Systemprogrammierung

Grundlage von Betriebssystemen

Sachwortverzeichnis

©Wolfgang Schröder-Preikschat

27. Januar 2017

Die in der Vorlesung (mündlich oder schriftlich) verwendeten Akronyme und Sachworte sind in der nachfolgenden Aufzählung zusammengefasst. Dabei werden die Akronyme nach den Sachworten, für die sie die Verkürzung bilden, aufgeschlüsselt. Für englischsprachige Sachwörter werden, soweit bekannt, die deutschsprachigen Entsprechungen angegeben. In der Beschreibung durch das ↑-Zeichen angeführte Sachwörter zeigen einen Kreuzverweis an. Jedes Sachwort wird erklärt, wobei dies im Zusammenhang mit dem hier relevanten Kontext der ↑Systemprogrammierung und in Bezug auf Betriebssysteme geschieht. Die Formulierungen erheben nicht den Anspruch auf Gültigkeit auch für andere Fachrichtungen in der Informatik. Ebenso erhebt die Aufzählung nicht den Anspruch auf Vollständigkeit für das in der Vorlesung behandelte Fachgebiet.

Der vorliegende Text ist ein *Nachschlagewerk*, dessen Lektüre, im Gegensatz zu einem Lehrbuch mit durchgehendem roten Faden, für gewöhnlich nicht von vorne nach hinten empfohlen ist. Vielmehr ist der Text Begleitmaterial zu den Vorlesungsfolien. Neben Systemprogrammierung sind hier insbesondere die Lehrveranstaltungen Betriebssysteme, Betriebssystemtechnik, Nebenläufige Systeme und Echtzeitsysteme eingeschlossen. Darüber hinaus kommen aber auch Lehrveranstaltungen zu den Themen Rechnerorganisation und Rechnerarchitektur als Bezugspunkt in Frage. Viele der dort (mündlich oder schriftlich) verwendeten Begriffe finden hier ihre Erklärung aus Sicht und mit Verständnis der systemnahen Programmierung. Ebenso dient der Text aber auch als Ergänzung zu (englisch- oder deutschsprachigen) Fach- oder Lehrbüchern zu diesen Themen, um dort verwendete Begriffe zu vertiefen, in einem größeren Kontext zu erfahren oder gar überhaupt erklärt zu bekommen.

Einige der behandelten Begriffe haben ihren Ursprung in der Erklärung eines anderen Begriffs, sie tauchen also nicht zwingend auch als Begriff in der jeweils verwendeten Primärliteratur auf. Diese Sachworte sind zwar nur durch Kreuzverweise in das Verzeichnis gelangt, haben aber sehr wohl einen thematischen Bezug zur Systemprogrammierung. Ein Beispiel dafür ist die ↑Abfangung, nämlich als bildhafte Darstellung einer Hilfskonstruktion (in ↑CPU und ↑Betriebssystem) zur Sicherung von einem ↑Programmablauf, der eine ↑synchrone Ausnahme hervorruft (↑*trap*).

Fragen, Hinweise, Korrekturen oder Kommentare nimmt der Autor gerne entgegen, am besten per ↑Nachricht an `wosch@cs.fau.de`.

ABB Abkürzung für (en.) *atomic basic block*, (dt.) ↑unteilbarer Grundblock.

Abfangung Konstruktion, die einen gegenwärtigen ↑Prozess gegen unkontrollierten Absturz sichert (in Anlehnung an das ↑Bauwesen). Die Fangstelle (↑*trap*) für eine ↑Aktion, bei der eine ↑Ausnahmesituation eingetreten ist. Der Prozess wird bei dieser Aktion, die er selbst verursacht, abgefangen: an der Fangstelle wird eine ↑synchrone Ausnahme erhoben.

abgesetzter Betrieb Bezeichnung für eine ↑Betriebsart, bei der die Ein-/Ausgabe getrennt von der damit im Zusammenhang stehenden Berechnung erfolgt.
Die Leistungsfähigkeit von einem ↑Rechensystem, in dem Berechnungen und Ein-/Ausgabe in „Personalunion“ durch ein einzelnes von der ↑CPU ausgeführtes ↑Programm bewerkstelligt wird, ist begrenzt durch die Geschwindigkeit, mit der die Ein-/Ausgabe durchgeführt werden kann. Grund dafür ist die im Vergleich zur CPU für gewöhnlich sehr viel langsamer arbeitende ↑Peripherie. So liegt die ↑Latenzzeit für den Zugriff auf ein Bandlaufwerk im Sekunden-/Minutenbereich, ein Festplattenlaufwerk bei 10 ms (über 2 Größenordnungen schneller), ein ↑SSD bei 25 μs (3 Größenordnungen schneller), den ↑RAM bei 50 ns (3 Größenordnungen schneller): das heißt, eine CPU könnte in der Zeit einen Durchsatz von mehr als 10^3 (SSD), 10^6 (Festplattenlaufwerk) oder 10^8 (Bandlaufwerk) Operationen pro Sekunde leisten (Stand 2016). Darüberhinaus ist in bestimmten Fällen gerade die Dauer für Eingabe oftmals nicht vorhersehbar und damit unbestimmt, etwa die Latenzzeit bis zu einem Tastendruck (Zeicheneingabe) oder Mausklick (Signaleingabe): also überall dort, wo ein unberechenbarer „externer Prozess“ (z.B. der Mensch) für die Eingabe verantwortlich ist.
Diese Ein-/Ausgabeabhängigkeit in der Rechenleistung wird wesentlich reduziert, wenn jeder für die CPU bestimmte Auftrag (↑*job*) immer über das schnellste ↑Peripheriegerät eingespeist und die eigentliche Zusammenstellung dieser Aufträge sowie die dazu benötigte Ein-/Ausgabe unabhängig von der CPU (*off-line*) bewerkstelligt wird. Das Rechensystem besteht dazu aus einerseits ↑Satellitenrechner für die eigentliche Bedienung der Peripherie und zur Durchführung der vergleichsweise langsamen Ein-/Ausgabe und andererseits dem ↑Hauptrechner zur Durchführung der eigentlichen Berechnungen. Die Ein-/Ausgabe (Satellitenrechner) wird abgesetzt von Berechnungen (Hauptrechner) betrieben oder umgekehrt. Dazu werden auf dem Satellitenrechner, über die jeweils benötigten und dort angeschlossenen Eingabegeräte, die zu bearbeitenden Aufträge samt aller Eingabedaten auf ein schnelles ↑Speichermedium überspielt, das dann zum Hauptrechner (manuell) übertragen wird, um dort die gestapelten Arbeitsaufträge nacheinander auszuführen. Für die Ausgabe wird umgekehrt verfahren: sie gelangt zunächst auf ein schnelles Speichermedium am Hauptrechner und wird danach an einen oder mehrere Satellitenrechner, an dem die benötigten Ausgabegeräte angeschlossen sind, (manuell) weitergegeben. Die Aufträge werden sodann vom Hauptrechner in ↑Stapelverarbeitung ausgeführt.

Ablagespeicher Bezeichnung für einen ↑Speicher zur mittelfristigen, dauerhaften Aufbewahrung von Informationen; Vorrichtung in einem ↑Rechensystem, wo eine ↑Datei abgelegt wird, ein ↑nichtflüchtiger Speicher. Die technische Speicherung erfolgt mitlaufend (*on-line*) zu dem ↑Prozess, der die Informationen ablegt oder auf (von ihm selbst oder anderen Prozessen) abgelegte Informationen zugreift. Auch ↑Sekundärspeicher genannt.

Ablaufinvarianz Merkmal von einem ↑Programm, mehr als einen ↑Handlungsstrang gleichzeitig zuzulassen, ohne sich in den daraus resultierenden zeitlich überlappenden Abläufen gegenseitig beeinflussen zu können. Eine solche Überlappung von Abläufen zeigt sich beispielsweise bei der ↑Ausnahmebehandlung, bei der nämlich der normale Ablauf eines Programms überlappt wird von dem unnormalen Ablauf durch einen ↑Ausnahmehandhaber. Variante davon ist das Programm zur ↑Unterbrechungsbehandlung, um im Betriebssystem auf eine ↑Unterbrechungsanforderung der Hardware reagieren zu können. Ähnliche Abläufe resultieren aus der Möglichkeit zur ↑Parallelverarbeitung ein und desselben Programms. Damit diese frei von gegenseitiger Beeinflussung sind, muss für den betreffenden Abschnitt ↑Eintrittsinvarianz gelten. Für einen ↑Prozess in diesem Abschnitt bedeutet dies: keine Verwendung statischer oder globaler Variablen, keine Veränderung an seinem ↑Text und kein Aufruf von einem nicht eintrittsinvarianten ↑Unterprogramm.

Ablaufplan Ergebnis der ↑Ablaufplanung.

Ablaufplanung Verfahren, das die Zuteilung von einen ↑Prozessor an einen ↑Prozess oder eine Gruppe von Prozessen regelt. Das Verfahren arbeitet rechnerunabhängig (*off-line*) oder mit-

laufend (*on-line*) zu den Prozessen, es liefert dementsprechend einen statischen oder dynamischen ↑Ablaufplan, der zur ↑Laufzeit der Prozesse unverändert befolgt wird oder aktuellen Bedürf- und Geschehnissen angepasst werden kann. Grundlage bildet ein auf das jeweilige Verfahren zugeschnittener ↑Einplanungsalgorithmus.

Ablaufsteuerung Steuerung von einem ↑Programmablauf mithilfe aufeinanderfolgender Befehle oder Bedingungen (Duden).

Abmeldung Prozedur, nach der eine Person oder ein externer ↑Prozess dem ↑Betriebssystem gegenüber die Teilnahme am Rechenbetrieb aufkündigt: das Sichabmelden aus einem ↑Rechner oder ↑Rechnernetz. Für gewöhnlich werden dabei alle während der individuellen Teilnahme dynamisch angeforderten ↑Betriebsmittel wieder freigegeben.

Abruf- und Ausführungszyklus Synonym zu ↑Befehlszyklus.

absolute Adresse Bezeichnung für eine ↑Adresse, die nicht relativ zu einem bestimmten Bezugspunkt steht, sondern direkt eine bestimmte ↑Speicherstelle benennt. Auch als ↑direkte Adresse bezeichnet. Derartige Adressen sind der ↑Relokation zu unterziehen, sollte das sie enthaltene ↑Programm im jeweiligen ↑Adressraum verschoben werden müssen.

absolute loader (dt.) ↑Absolutlader.

Absolutlader Bezeichnung für ein ↑Programm, das ein ausschließlich absolute Adressen verwendendes Programm in den ↑Hauptspeicher bringt; ein ↑Lader, der keine ↑Relokation durchführt. Durch solch einen Lader wird typischerweise das ↑Betriebssystem in den Hauptspeicher gebracht (↑*bootstrap*).

abstrakte Maschine Bezeichnung für eine Maschine, die nur gedanklich und nicht physisch existiert. Sie ist entweder ein als Modell geschaffenes Konstrukt, um komplexe Vorgänge innerhalb von Rechensystemen auf theoretischer Ebene zu untersuchen (Automat, Turing-Maschine) oder ein über eine wohldefinierte Schnittstelle zugängliches Softwaregebilde, das allein oder im Ensemble mit anderen Maschinen ihrer Art hilft, eine ↑semantische Lücke zu schließen (↑Betriebssystem). Letztere Variante meint ein ↑Programm, das zwar ein physisches ↑Betriebsmittel in Form von ↑Speicher belegt, als Software jedoch selbst ein „nicht technisch-physikalischer Funktionsbestandteil“ (Duden) darstellt.

access control list (dt.) ↑Zugriffskontrollliste.

access matrix (dt.) ↑Zugriffsmatrix.

accounting (dt.) ↑Buchführung.

ACET Abkürzung für (en.) *average-case* ↑*execution time*, mittlere anzunehmende ↑Ausführungszeit.

ACL Abkürzung für (en.) ↑*access control list*.

activation record (dt.) ↑Aktivierungsblock.

actual parameter (dt.) ↑tatsächlicher Parameter.

Adressabbildung Zuordnung ↑logische Adresse zu ↑reale Adresse. Bildet ein ↑seitennummerierter Adressraum die Grundlage, ergibt sich bei einer einstufigen Abbildung die reale Adresse a_r aus der logischen Adresse a_l wie folgt:

$$a_r = (pagetable[p].pageframe * sizeof(page)) + o$$

mit $p = a_l / sizeof(page)$ und $o = a_l - (p * sizeof(page))$, wobei hier „+“ als „bitweises oder“ und „−“ als „bitweises und“ (`|` bzw. `&` in ↑C) zu verstehen sind. Bei einer mehrstufigen

Abbildung ist zunächst die ↑Seitentabelle der letzten Stufe zu lokalisieren. Dies geschieht durch Indexoperationen auf vorgeschalteten Seitentabellen, wobei für jede Stufe eine Untergliederung der Seitennummer p vorgenommen wird, um die Indexwerte der Tabelle der jeweiligen Stufe zu erhalten. Ist ein ↑segmentierter Adressraum gegeben, geschieht die Abbildung gemäß:

$$a_r = segmenttable[s].base + a_l$$

mit $s = [0, 2^S - 1]$, ↑Segmentname. Die Abbildungsfunktion basiert hierbei auf eine Zweikomponentenadresse $a = (S, a_l)$ (zweidimensionaler Adressraum), wohingegen im seitennummerierten Fall die abzubildende Adresse als Einzelkomponente $a = a_l$ in Erscheinung tritt (eindimensionaler Adressraum).
Beide Techniken lassen sich kombinieren, wofür es grundsätzlich zwei Ansätze gibt. Zum einen kann die Seitentabelle als ↑Segment aufgefasst werden. Ihre Adresse und Größe im ↑Hauptspeicher findet sich dann in der ↑Segmenttabelle des mit Segmentname s bezeichneten Deskriptors. Das Segment, auf das sich Adresse a_l bezieht, bildet dann einen seitennummerierten Adressraum und a_l wird wie oben gezeigt abgebildet. Zum anderen wird die von der ↑Segmentadressierungseinheit gebildete Adresse a_r von der MMU als „logische Adresse“ a_l^r aufgefasst und durch die ↑Seitenadressierungseinheit geschickt, um die wirkliche reale Adresse a_r zu erhalten (↑x86 ab ↑i386). Es wird also entweder a_l (erster Ansatz) oder a_l^r (zweiter Ansatz) als logische Adresse eines seitennummerierten Adressraums verstanden, wobei dieser Adressraum im ersten Ansatz einen segmentlokalen und im zweiten Ansatz einen systemglobalen Bezug hat.
Zu beachten dabei ist der feine Unterschied, dass im ersten Ansatz die ↑Seite und im zweiten Ansatz immer noch das ↑Byte (↑Oktett) das Strukturelement eines Segments bildet: letzterer erlaubt somit die Aufteilung derselben Seite auf zwei Segmente, indem ein vorderer Abschnitt (in der letzten Seite) am Ende des einen Segments und der restliche hintere Abschnitt (in der ersten Seite) am Anfang des anderen Segments liegt — sofern von der MMU unterstützt (z.B. ↑i386 im 16-Bit Schutzmodus). In allen Fällen ist die Abbildung aber nur bei gültigen Indexwerten und gültigen Deskriptorattributen definiert. Im Fehlerfall fängt die ↑CPU die Generierung der realen Adresse ab (↑*segmentation fault*).

Adressbereich Abschnitt von linearen Adressen in einem ↑Adressraum.

Adressbreite Bitanzahl, die zur Darstellung einer ↑Adresse im ↑Dualsystem zur Verfügung steht. Typisch waren oder sind 16, 18, 20, 24, 32, 48 und 64 Bits pro Adresse (Stand 2016).

Adressbus Verbindungssystem in einem ↑Rechner, über das eine ↑reale Adresse übermittelt wird.

Adresse Nummer einer ↑Speicherstelle.

Adressierungsart Lokalisierung der/des Operanden von einen ↑Maschinenbefehl. Gängig sind: Registeradressierung, der Befehl enthält den Namen des dem Operanden zugeordneten Prozessorregisters; Direktwertadressierung, der Befehl enthält den Operanden selbst; direkte Adressierung, der Befehl enthält die Speicheradresse des Operanden; indirekte Adressierung, der Befehl enthält den Namen des die Speicheradresse des Operanden führenden Prozessorregisters; relative Adressierung, der Befehl enthält den auf eine Basisadresse bezogenen Abstand zum Operanden; indizierte Adressierung, der Befehl enthält den Namen des den zu einer Basisadresse bezogenen Abstand zum Operanden führenden Prozessorregisters.

Adressraum Menge von nichtnegativen ganzen Zahlen, endlich, wobei jedes Element darin eine ↑Adresse repräsentiert. Die Kardinalität dieser Menge ist bestimmt durch die ↑Adressbreite der (realen/virtuellen) Maschine.

Adressraumisolation Abkapselung von einem ↑Prozess, indem sein ↑Adressraum physisch von den jeweiligen Adressräumen anderer Prozesse im ↑Rechensystem getrennt gehalten wird. Technische Grundlage dafür bildet der ↑Speicherschutz, in aller Regel hardwarebasiert unter

Verwendung einer ↑MMU oder ↑MPU. Geschieht die Abkapselung *total*, betrifft sie also den kompletten Adressraum, gilt der Prozess als ↑schwergewichtiger Prozess. Greift sie dagegen *partiell*, bezieht sie sich nämlich nur auf den das ↑Stapelsegment eines Prozesses betreffenden ↑Adressbereich, ist der Prozess ein ↑leichtgewichtiger Prozess. Ein Prozess kann damit aus seinen isolierten Adressbereichen nicht ausbrechen und somit auch nicht in den Adressraum eines anderen Prozesses einbrechen.

Adresszähler Bezeichnung für einen Zeiger in ein in ↑Assembliersprache formuliertes ↑Programm, über den die einzelnen Programmanweisungen ihre jeweilige Speicheradresse erhalten. Dieser Zeiger ist eine Variable im ↑Assemblierer. Jeder Programmabschnitt (↑Text, ↑Daten, ↑BSS) besitzt einen eigenen, mit 0 initialisierten Zeiger. Bei der ↑Assemblierung einer Programmanweisung wird ihr der aktuelle Zeigerwert als Speicheradresse zugeordnet und der Zeiger wird um die Länge (in Bytes) dieser Anweisung erhöht. So erhält etwa der Name der Marke (d.h., ein ↑Symbol) einer Assembliersprachenanweisung einen Wert, der die ↑Adresse der Anweisung relativ zum Abschnittsbeginn repräsentiert. Der Zeiger kann als Operand in einer Assembliersprachenanweisung verwendet werden. Typisch ist das Dollarzeichen (`$`): einem Symbol vorangestellt liefert es dessen Adresswert als Operand.

after you, after you Sinnbild für einen möglichen Effekt bei der ↑Synchronisierung, der das ↑Verhungern von ↑Prozessen zur Folge haben kann. Der Überlieferung nach (Dijkstra) eilen zwei Gentleman zielstrebig auf einen Türdurchgang zu, der zu eng ist, dass beide ihn zugleich nebeneinander passieren könnten. Als Mann von Anstand gewährt jeder dem anderen selbstverständlich den Vorrang, um daraufhin nacheinander seines Weges gehen zu können. Dazu zieht einer seinen Bowler und bittet den anderen, durch ein höfliches „nach Ihnen“, voranzugehen. Letzterer steht seinem Konkurrenten in nichts nach, zieht gleichzeitig seinen Bowler und bittet ersteren mit einem ebenso höflichen „nach Ihnen“ vor. Beide nehmen an, der jeweils andere hält inne, was jedoch nicht der Fall ist. So kommt es wie es kommen musste: eng nebeneinander gehend prallen sie zusammen gegen die Türpfosten, erkennen ihren Irrtum, ziehen sich zurück und, anständig wie sie sind, gewähren sie dem jeweils anderen erneut den Vorrang. Blind gegenüber anderem Sozialverhalten, das nämlich Fortschritt verspricht, verfallen beide immer wieder demselben Ablauf — und wenn sie nicht gestorben sind, dann „kreiseln“ sie noch heute vor der Tür.

ageing (dt.) ↑Alterung.

Aktion Ausführung der Anweisung einer Maschine durch einen ↑Prozessor. Eine Anweisung, die eine Einzelaktion auf höherer Ebene beschreibt, kann als ↑Aktionsfolge auf tieferer Ebene stattfinden, beispielsweise: `ADD X,Y` ↦ `LOAD X; ADD Y; STORE X`.

Aktionsfolge Reihe von nacheinander oder gleichzeitig stattfindenden Aktionen. Dabei kann jede ↑Aktion gänzlich oder in Teilen auch demselben ↑Handlungsstrang zugehören.

aktives Register Bezeichnung für ein ↑Prozessorregister, das von einem ↑Handlungsstrang benutzt wird. Im Rahmen seines Optimierungslaufs versucht der ↑Kompilierer, den häufig verwendeten Werten (von Variablen) des zu übersetzenden ↑Programms schnellen ↑Registerspeicher zuzuteilen. Ein Register wird aktiviert, wenn der ↑Prozessor auf Veranlassung des Handlungsstrangs den ↑Maschinenbefehl zur Aufnahme (*load*) eines Werts (Konstante, Inhalt einer Programmvariablen) ausführt. Das Register bleibt für eine bestimmte Programmvariable aktiv, solange der Prozessor (auf Veranlassung des Handlungsstrangs) noch nicht den Maschinenbefehl zum Umspeichern (*store*) des Registerinhalts ausgeführt hat.

aktives Warten Verfahren, bei dem ein ↑Prozess tatkräftig ein bestimmtes ↑Ereignis erwartet: im Gegensatz zu ↑passives Warten, gibt der Prozess seinen ↑Prozessor während der gesamten ↑Wartezeit nicht ab und bleibt von sich aus laufend (↑Prozesszustand).
Ein solches ↑Warteverhalten (↑*busy waiting*) führt nur bedingt zur effizienten Nutzung der ↑CPU. Im Falle einer ↑Betriebsart, bei der die CPU zu einem Zeitpunkt grundsätzlich nur

ein ↑Programm ausführt (↑*uniprogramming*), wird unnötigerweise Energie für Nichtstun verbraucht (↑*idle*). Können dagegen mehrere Programme im ↑Arbeitsspeicher zugleich zur Ausführung bereitgestellt sein (↑*multiprogramming*), liegt die CPU trotz anstehender Arbeitslast unnötigerweise brach.
Einen Prozess tätig warten zu lassen, lohnt sich wenn überhaupt nur bei kurzer Wartezeit — die dem Prozess genau im Entscheidungsmoment zum Warten und damit vorher bekannt sein müsste, was sie für gewöhnlich jedoch nicht ist. Nur wenn das zeitliche Verhalten des externen Prozesses, der das Ereignis hervorruft, vorhersagbar ist, etwa die ↑Zwischenankunftszeit von Eingabedaten oder von Ausgabebestätigungen, lässt sich die Wartezeit des (internen) Prozesses bestimmen. Bei Maschinen oder allgemein physikalischen Prozessen mit periodischem Verhalten in Bezug auf Ein-/Ausgabe ist solch eine Vorhersage durchaus möglich, nicht jedoch bei aperiodischem Verhalten (z.B. wenn der Mensch selbst als externer Prozess ein Glied in der Verarbeitungskette bildet, nämlich eine Eingabe liefern oder die Ausgabe bestätigen muss).

Aktivierungsblock (en.) ↑*activation record*. Bezeichnung für den beim Aufruf eines ↑Unterprogramms jeweils benutzten Teilbereich auf dem ↑Laufzeitstapel einer ↑Prozessinkarnation. Dieser Teilbereich umfasst für gewöhnlich folgende Strukturelemente:

- die Argumente (↑*actual parameter*) für das Unterprogramm, sofern erforderlich,
- die Rücksprungadresse zu dem übergeordneten Unterprogramm, das das betreffende Unterprogramm aufgerufen hat,
- die Inhalte der ↑Prozessorregister, die im Unterprogramm verwendet werden, aber für das übergeordnete Unterprogramm invariant bleiben müssen (↑*callee-saved register*),
- Platzhalter für die lokalen Variablen (*automatic variables* in ↑C) und
- dem Unterprogramm nur vorübergehend bereitgestellter Behelfsspeicher, beispielsweise Platz für invariant zu haltende Inhalte von Prozessorregister, die jedoch ungesichert in einem anderen, aufzurufenden Unterprogramm verwendet werden dürfen (↑*caller-saved register*).

Die Größe und Reihenfolge der einzelnen Strukturelemente eines solchen Teilbereichs legt der ↑Kompilierer fest und schlägt sich unter anderem auch in der durch ihn definierten ↑Aufrufkonvention für Unterprogramme nieder. Dieser Bereich wird beim Aufruf in ein Unterprogramm automatisch aufgebaut und vor dem Rücksprung aus dem Unterprogramm automatisch zurückgebaut.
Der zur Argumentenübergabe erforderliche Bereich oben auf dem Stapel des aufrufenden Unterprogramms überlagert sich praktisch mit dem entsprechenden Bereich des aufgerufenen Unterprogramms. Für die Übergabe selbst gibt es zwei Herangehensweisen. In dem einen Ansatz legt das aufrufende Unterprogramm die Argumente zum Aufruf einfach auf den Stapel ab (*push*) und macht den Bereich nach Rückkehr aus dem Unterprogramm wieder frei (*pop*). Im anderen Ansatz verwendet das aufrufende Unterprogramm einen Übergabebereich fester (vom Kompilierer berechneter) Größe, der zur Aufnahme der Argumente dient.

Algol 60 Programmiersprache der Algol-Familie: imperativ, prozedural (1960). Abkürzung für *algorithmic language*, die 1963 ihre endgültige Fassung erhielt.

alignment (dt.) ↑Ausrichtung.

allgemeiner Semaphor Bezeichnung für einen ↑Semaphor, bei dem die ↑Ablaufsteuerung eine *Ganzzahlvariable* benutzt, der einen ganzzahligen elementaren Datentyp (`int`) besitzt. Die Operationen ↑P und ↑V sind zählende Operationen, daher auch *zählender Semaphor*. Zustandsänderungen in Bezug auf die enthaltene Ganzzahlvariable bilden jeweils eine ↑Elementaroperation, die für gewöhnlich aber nicht als ↑atomare Operation vorliegt: *Subtraktion* beziehungsweise *Addition*, um 1.
Die Ablaufsteuerung lässt einen ↑Prozess in P blockieren, wenn der Variablenwert vor der

Subtraktion 0 ist. Anderenfalls lässt P den Prozess voranschreiten. Die Subtraktion kann bedingt oder unbedingt erfolgen. Im ersten Fall entspricht der Semaphor einer nichtnegativen ganzen Zahl, da ein in die Blockierung geleiteter Prozess den Wert nicht weiter dekrementiert. Allein anhand des Zahlenwerts ist dann nicht feststellbar, ob ein Prozess in P blockiert und durch V gegebenenfalls zu deblockieren ist. Dies ist im zweiten Fall (ganze Zahl) möglich, da jeder blockierende Prozess den Zahlenwert weiter in den negativen Bereich bringt. Der Absolutwert liefert dann die Anzahl der in P blockierten Prozesse. Damit eröffnet sich ein größerer Spielraum für die technische Auslegung der ↑Warteschlange, also ob diese als Datenstruktur pro Semaphor ausgeprägt ist (damit aber auch Gefahr von ↑Interferenz in sich trägt) oder durch den ↑Planer berechnet werden soll (damit höhere ↑Latenzzeit mit sich bringt). Entspricht der Semaphor einer nichtnegativen ganzen Zahl, ist die Ausprägung als Datenstruktur üblich. In dem Fall kann V darüber dann einfach prüfen, ob Prozesse zur Deblockierung anhängig sind.
Typischer Anwendungsfall dieser Semaphorart ist die Verwaltung „zählbarer“ ↑Betriebsmittel, nämlich die Vergabeprozedur sowohl für ein ↑konsumierbares Betriebsmittel (Erzeuger-Verbraucher-Beziehung) als auch für ein ↑wiederverwendbares Betriebsmittel (begrenzter Puffer/Speicher; begrenzte Anzahl von Hardware-/Softwareeinheiten irgendwelcher Art).

Alpha Bezeichnung für den als Nachfolger zur ↑VAX von DEC entwickelten Mikroprozessor (1992): 64-Bit, ↑RISC, betrieben durch ↑OpenVMS und verschiedene ↑UNIX Dialekte.

Alterung (en.) ↑*ageing*. Bezeichnung für eine bestimmte Technik bei der ↑Prozesseinplanung, um möglichem ↑Verhungern vorzubeugen. Dabei fließt die ↑Wartezeit eines ↑Prozesses auf der ↑Bereitliste ein in die Entscheidungsfindung zur ↑Einlastung der ↑CPU. Typisches Beispiel dafür ist ↑HRRN und ↑MLFQ.

ambient noise (dt.) ↑Umgebungsrauschen.

AMD 64 Prozessorarchitektur, 64-Bit, abwärtskompatibel zu ↑x86, AMD (2000). Erste Produktfreigabe in 2003 (AMD Opteron „K8“).

AMP Abkürzung für (en.) ↑*asymmetric multiprocessing* und (en.) ↑*asymmetric multiprocessor*.

Angriffssicherheit Zustand des Geschütztseins einer ↑Entität in einem ↑Rechensystem vor Gefahr oder Schaden durch einen böswilligen ↑Prozess (in Anlehnung an den Duden). Um diesen Zustand zu gewährleisten, sorgt ein ↑Betriebssystem für die Unwirksamkeit jeder ↑Aktion, die zur Verletzung der ↑Schutzdomäne anderer Prozesse führen kann. Grundsätzlich bedeutet dies, unautorisiertes Eindringen eines Prozesses in eine fremde Schutzdomäne zu erkennen und abzufangen (↑Ausnahmesituation). Dazu ist einem Prozess mindestens der unautorisierte Zugriff auf bestimmte Entitäten zu verwehren, was im Falle von Zugriffen mittels dazu extra vorgesehenen Operationen (z.B. lesen/schreiben einer ↑Datei oder von einem ↑Speicherwort) noch vergleichsweise einfach ist. Ungleich schwieriger ist die Zugriffsabwehr, wenn ein Prozess mit legitimen Operationen Informationen, die nicht für ihn bestimmt sind, unbemerkt abgreifen kann (↑*covered channel*).
Jede ↑Schutzverletzung, die den Zustand oder die Funktion einer Entität (wie oben erwähnt) verändert, ist nicht nur illegitim, sonder kann insbesondere auch zur Beeinträchtigung der ↑Betriebssicherheit führen. Umgekehrt gilt dies nicht.

Ankunftszeit (en.) ↑*arrival time*. Bezeichnung für eine ↑Prozessgröße, die den Zeitpunkt festlegt, zu dem eine ↑Aufgabe zur Verarbeitung zur Verfügung steht (↑*release time*).

Anmeldung Prozedur, nach der sich eine Person oder ein externer ↑Prozess dem ↑Betriebssystem gegenüber zur Teilnahme am Rechenbetrieb und Aufnahme einer bestimmten Arbeit ankündigt. Die sich anmeldende ↑Entität wird authentifiziert und zugelassen, wenn sie berechtigt zur Teilnahme ist und ihr eine Mindestmenge an ↑Betriebsmittel in dem Moment bereitgestellt werden kann. Ist sie nicht berechtigt, wird ihr der Zugang zum ↑Rechensystem

verwehrt. Besteht temporärer Betriebssystemmangel, wird die Zulassung solange aufgeschoben, bis die Zusicherung der zur Arbeitsaufnahme benötigten Betriebsmittel in Aussicht steht (↑*long-term scheduling*). Nach erfolgter Zulassung wird (für den internen Prozess) eine ↑Prozessinkarnation angelegt, die die Entität im Rechensystem repräsentiert. Dieser Prozess nimmt seine Arbeit in einem bestimmten ↑Namensraum (↑*root file system*) auf, dort beginnend in seinem ↑Heimatverzeichnis.

Antwortzeit (en.) ↑*response time*. Zeitspanne zwischen Absetzen eines Auftrags durch einen ↑Prozess und der Entgegennahme der Antwort daraufhin im selben Prozess. Beispielsweise die zur ↑Laufzeit im ↑Maschinenprogramm anfallende ↑Latenzzeit einer per ↑Systemaufruf abgeforderten und im ↑Betriebssystem durchgeführten ↑Systemfunktion.

Anwenderprogramm Synonym zu ↑Anwendungsprogramm.

Anwendung (en.) ↑*application*. Bezeichnung für mindestens ein ↑Maschinenprogramm, das die für einen bestimmten ↑Anwendungsfall erforderlichen Funktionen festlegt. Dabei handelt es sich im Allgemeinen eben um nicht systemtechnische Funktionen, die nämlich nicht durch ↑Systemsoftware implementiert sind.

Im Falle von mehreren zusammengehörigen Maschinenprogrammen enthalten diese für gewöhnlich auch Anweisungen, die eine Kommunikation untereinander ermöglichen und der Kooperation bei der arbeitsteiligen Erfüllung von den mit den nicht systemtechnischen Funktionen verknüpften Aufgaben dienen. Dies bedeutet dann aber auch, dass mehr als ein ↑Prozess zugleich stattfinden muss, um die gewünschte nicht systemtechnische Funktionalität bereitzustellen. Allerdings sind dafür mehrere Programme alles andere als ein zwingendes Merkmal, denn die Anwendungsfunktionalität kann auch gut durch ein einzelnes ↑nichtsequentielles Programm beschrieben sein. Ebenso ist natürlich auch ein Mix aus sequentiellen und nichtsequentiellen Programmen möglich. Einzig die nicht systemtechnische Funktion bildet das charakteristische Merkmal und nicht etwa der Aspekt, wie diese Funktion systemtechnisch ausgelegt ist, also etwa ob sie als (ein- oder mehrelementige Menge von) ↑Faden, ↑Faser oder gar ↑Fäserchen in Erscheinung tritt.

Zu beachten ist, dass die Formulierung „nicht systemtechnische Funktion“ genau genommen relativistisch zu verstehen ist und oftmals ein bestimmtes, jedoch nicht immer explizit benanntes Bezugssystem vor Augen hat. So beziehen sich beispielsweise Programme, die eine Datenbankanwendung bilden, auf ein Datenbanksystem, dessen konstituierenden Programme ihrerseits eine Betriebssystemanwendung darstellen. Noch weiter oben in der Hierarchie kann die Datenbankanwendung selbst als ein System von Dienstnutzer/-leister (*client-server model*) ausgelegt sein, wobei letztere durch Programme implementiert sind, die im Verbund eine bestimmte Webanwendung ausmachen.

Anwendungsfaden Bezeichnung für einen ↑Faden im ↑Maschinenprogramm, der daselbst implementiert ist und nicht erst durch einen Faden im ↑Betriebssystemkern entsteht. Sämtliche für solch einen Faden erforderlichen Betriebsmittel sowie für seine Verwaltung nötigen Funktionen stellt das als ↑nichtsequentielles Programm ausgelegte Maschinenprogramm. Dies betrifft insbesondere den ↑Laufzeitkontext eines Fadens und die Funktionen zur ↑Ablaufplanung, ↑Einlastung und ↑Synchronisation.

Ein ↑Prozesswechsel, um innerhalb des Maschinenprogramms vom aktuellen zu einem anderen Faden umzuschalten, verläuft im lokalen ↑Adressraum und auch unter ↑Speicherschutz ohne ↑Systemaufruf. Jeder dieser Fäden ist vom Bautyp her ein ↑federgewichtiger Prozess, für den zur Steuerung und Überwachung keine kostspielige ↑Systemfunktion erforderlich ist und er selbst dazu auch keine benutzt. Für das ↑Betriebssystem ist ein solcher Faden kein ↑Objekt erster Klasse, das heißt, der durch den Faden konkretisierte ↑Prozess ist dem Betriebssystem gänzlich unbekannt.

Eine vom Maschinenprogramm aus beanspruchte Systemfunktion läuft damit nicht im Namen des effektiv beanspruchenden Fadens, sondern nur im Namen derjenigen ↑Entität ab, die im Betriebssystem das jeweils in Ausführung befindliche Maschinenprogrammen repräsentiert. Ist dies bespielsweise ein ↑schwergewichtiger Prozess oder ein ↑leichtgewichtiger

Prozess und wird dieser dann durch die Systemfunktion blockiert (↑Prozesszustand), blockieren implizit alle Fäden, die unbekannterweise eben auf diese eine ↑Prozessinkarnation durch eine ↑Aktion im Maschinenprogramm abgebildet wurden.

Anwendungsfall (en.) ↑*use case*. Beschreibung funktionaler und, wenn möglich, auch nichtfunktionaler Merkmale eines geplanten oder existierenden Systems. Dokumentiert wird insbesondere das nach außen (für den oder die Nutzer) sichtbare Verhalten dieses Systems. Je nach Detaillierungsgrad der Beschreibung sind Einzelheiten des Anwendungsverhalten mehr oder weniger abstrakt dargestellt. Mit zunehmend konkreter werdender Darstellung können Informationen gewonnen werden, die als ↑Vorwissen zu der jeweiligen ↑Anwendung in einem ↑Betriebssystem nutzbar sind. Beispielsweise kann dies Wissen über die Anzahl möglicher ↑Prozesse, der Art ihrer Interaktion (↑*shared memory*, ↑*message passing*), deren Begrenztheit, jeweilige Dauer (↑WCET), Periode oder Bedarf an ↑Betriebsmitteln, sowie Kenntnis über ↑Systemaufrufe und damit zu Art und Zahl der überhaupt zur Anwendungsunterstützung benötigten ↑Systemfunktionen umfassen.

Anwendungsprogramm (en.) ↑*application program*, ↑*user program*. Bezeichnung einer ↑Anwendung, die aus genau einem ↑Maschinenprogramm besteht.

Anwesenheitsbit Attribut einer ↑Seite oder von einem ↑Segment, gespeichert im entsprechenden Deskriptor der ↑MMU, das einen Hinweis über das Dasein (der Seite/des Segments) im ↑Prozessadressraum gibt. Eine Schaltvariable als Ausprägung eines booleschen Datentyps, die zwischen anwesend (`true`, Zugriff erlaubt) und abwesend (`false`, Zugriff verwehrt) unterscheidet. Die MMU signalisiert der ↑CPU, die durch den ↑Maschinenbefehl beschriebene ↑Aktion zu unterbrechen (↑*trap*), sobald dieser, spezifiziert durch die ↑Adressierungsart, Zugriffe auf abwesende Seiten/Segmente ausübt.
Ursprünglich wird durch dieses Attribut ↑virtueller Speicher in einem Prozessadressraum gekennzeichnet, nämlich alle für einen ↑Prozess gültige Seiten/Segmente, die zur Zeit jedoch nicht im ↑Hauptspeicher liegen. Allerdings kann es auch Verwendung finden, wann immer ein ↑Betriebssystem den Zugriff auf einen bestimmten ↑Adressbereich sofort in Erfahrung bringen möchte, unabhängig davon, ob die diesem Bereich zugeordneten Seiten/Segmente im Hauptspeicher residieren — mehr dazu aber in SP2 (*copy on reference*).

App Kurzbezeichnung für (en.) ↑*application*. Ursprünglich nur das Kürzel für eine ↑Anwendung für programmierbare Mobiltelefone (*smartphone*). Mittlerweile aber auch die Bezeichnung für ein ↑Anwendungsprogramm für einen gewöhnlichen ↑Rechner.

application (dt.) ↑Anwendung.

application program (dt.) ↑Anwenderprogramm, ↑Anwendungsprogramm, ↑Benutzerprogramm.

Applikationsprogramm Synonym zu ↑Anwendungsprogramm.

Arbeitsentzug (en.) ↑*work stealing*. Bezeichnung für eine Variante der ↑Einlastung, bei der sich (bildlich gesprochen) ein ↑Prozessor selbst um ↑Prozesse bemüht, um nicht in den ↑Leerlauf eintreten zu müssen. Im Gegensatz zu ↑Arbeitsteilung wird ein Prozess aus der ↑Bereitliste eines anderen Prozessors „gestohlen“, sollte die eigene Bereitliste im Moment eines sonst anstehenden ↑Prozesswechsels leer sein. Sollte kein Prozess zum „Stehlen“ gefunden werden können, bedeutet dies, dass auch insgesamt keine ↑Aufgaben zur Bearbeitung anstehen und ein Prozessor daher nicht unnötigerweise untätig wird. Das bei Arbeitsteilung bestehende Problem, solche Phasen der Untätigkeit eines Prozessors nicht sicher abwenden zu können, stellt sich hier nicht.

Arbeitsmenge Minimalmenge des im ↑Hauptspeicher zu einem Zeitpunkt vorzuhaltenden Bestands von ↑Text und ↑Daten von einem ↑Prozess, damit dieser effizient stattfinden und seine Arbeit verrichten kann (↑*working set*). Zur Aufbewahrung des Gesamtbestands wird ↑seitennummerierter virtueller Speicher als Grundlage genommen. Die Bestandsgröße ist

ganzzahlige Vielfache einer ↑Seite.
Die Erfassung solcher Seiten dient der Modellierung von Prozessverhalten und damit auch einer Abschätzung der im ↑Rechensystem voraussichtlich noch zu erwartenden Vorgänge in Bezug auf den Hauptspeicher sowie dessen Nutzung. In dem Modell wird davon ausgegangen, dass in einem ↑Maschinenprogramm keine Anweisungen kodiert sind, um dem ↑Betriebssystem per ↑Systemaufruf Hinweise über den im Hauptspeicher erforderlichen Seitenbestand zu übermitteln. Vielmehr ist es Aufgabe des Betriebssystems eigenständig diesen Bestand zu ermitteln. Grundlage dafür ist die zurückliegende ↑Referenzfolge eines Prozesses, das heißt, die sich daraus ergebende Menge der im letzten ↑Zeitfenster referenzierten Seiten (↑Betriebsseite). Diese Betriebsseiten bilden eine Teilmenge des im Hauptspeicher liegenden Seitenbestands (↑*resident set*), Zugriffe darauf haben keine ↑Seitenfehler zur Folge. Ziel ist es, die Anzahl der residenten Seiten (d.h., den für einen Prozess erforderlichen Hauptspeicherbedarf) zu minimieren und dadurch den Grad an ↑Mehrprogrammbetrieb (d.h., die Anzahl von Prozessen) zu maximieren.
Um Hauptspeicherplatz für andere Prozesse zu schaffen, lässt die ↑Seitenumlagerung einzelne Betriebsseiten unangetastet: eine Arbeitsmenge wird niemals auseinandergerissen, da sonst das Flattern (↑*thrashing*) von Betriebsseiten droht. Stattdessen wird diese Menge (Betriebsseiten) gegebenenfalls komplett umgelagert (*working-set* ↑*swapping*) und der zugehörige Prozess wird für die Dauer, während all seine Betriebsseiten ausgelagert sind, suspendiert. Zur Fortsetzung des Prozesses werden all diese Seiten im Verbund eingelagert (↑*pre-paging*).

Arbeitsmodus Art und Weise des Handelns, Tätigwerdens, von einem ↑Prozessor (↑CPU oder ↑Rechenkern) im Rahmen der Ausführung von einem ↑Programm; (lat.) *modus operandi* (in Anlehnung an den Duden). Zu einem Zeitpunkt handelt der Prozessor entweder für das ↑Betriebssystem oder für ein ↑Maschinenprogramm, aber niemals für beide zugleich — indem er jedoch für das Betriebssystem handelt, kann dadurch sehr wohl die Ausführung des Maschinenprogramms voranschreiten (↑partielle Interpretation).
Ist das Betriebssystem aktiv (↑*system mode*), handelt der Prozessor privilegiert, das heißt, er führt jeden korrekt kodierten ↑Maschinenbefehl aus (↑*privileged mode*). Synonyme Bezeichnungen für diese Arbeitsweise fokussieren auf das jeweils aktive Subsystem innerhalb eines Betriebssystems, das heißt, sie beziehen sich auf das ↑Hauptsteuerprogramm (↑*supervisor mode*) oder den ↑Betriebssystemkern (↑*kernel mode*). In diesem Modus kann insbesondere durch Setzen einer ↑Unterbrechungssperre der Prozessor dazu veranlasst werden, zeitweilig ungestört von einer ↑Unterbrechung zu handeln (↑*uninterruptible mode*).
Ist das Betriebssystem inaktiv, aber Arbeit ohne ↑Leerlauf zu leisten, handelt der Prozessor für das Maschinenprogramm (↑*user mode*) und damit nicht privilegiert (↑*unprivileged mode*). Der Wechsel in den Systemmodus geschieht bei jeder ↑Ausnahme, einschließlich ↑Systemaufruf. Der Wechsel zurück in den Benutzermodus erfolgt mit Beendigung einer ↑Ausnahmebehandlung, etwa wenn die partielle Interpretation eines Maschinenbefehls durch das Betriebssystem abgeschlossen ist.
Die Inbetriebnahme von dem ↑Rechner findet ebenfalls im Systemmodus statt. Zur erstmaligen Aufnahme der Ausführung eines Maschinenprogramms ist demzufolge ein (durch einen offensichtlich nicht abgesetzten Systemraufruf hinterlassener) Systemzustand nachzubilden und aufzusetzen, um vom System- in den Benutzermodus wechseln zu können.

Arbeitsspeicher Bezeichnung für den zur Ausführung von einem ↑Programm und nur zu dessen ↑Laufzeit benötigten ↑Speicher. Im Wesentlichen ↑Hauptspeicher, eventuell jedoch erweitert um einen peripheren Speicher zur Ablage von zeitweilig für die Programmausführung nicht erforderliche Bestände von ↑Text oder ↑Daten. Die Erweiterung ist funktional transparent oder intransparent, das heißt, Zugriffe auf den Erweiterungsspeicher sind in den Programmen selbst nicht formuliert (↑virtueller Speicher) oder explizit durch spezielle Anweisungen zum Ausdruck gebracht (↑Überlagerungsspeicher).
Die Verwendung des Begriffs als Synonym zu Hauptspeicher ist mit Vorsicht zu betrachten, sie ist unpräzise. Ein ↑Prozess muss zwar im Hauptspeicher stattfinden, damit er seine Arbeit verrichten kann, jedoch kann er bei ↑Speichervirtualisierung weit mehr als nur die direkt

über den Hauptspeicher verfügbaren Text- und Datenbestände bearbeiten. Hinter Hauptspeicher steht sowohl in logischer als auch in physischer Hinsicht ein anderer Speicherbegriff: logisch, da ohne ihn grundsätzlich kein (von Neumann) ↑Rechner funktioniert, aber ein solcher Rechner ohne oben genannten Erweiterungsspeicher seine Funktion sehr wohl erfüllen kann; physisch, da seine Größe durch die ↑Speicherbestückung, nicht etwa die ↑Adressbreite des Rechners oder sein ↑realer Adressraum, limitiert ist.

Arbeitsteilung (en.) ↑*work sharing*. Bezeichnung für eine Variante der ↑Einplanung von ↑Prozessen, bei der die zu bearbeitenden ↑Aufgaben auf ↑Prozessoren verteilt werden. Im Gegensatz zu ↑Arbeitsentzug wird die einem Prozessor jeweils zugeordnete ↑Bereitliste durch einen übergeordneten ↑Planer mit Prozessen befüllt.
Solange mehr Aufgaben zur Bearbeitung anstehen als Prozessoren verfügbar sind, ist es das Ziel, die Befüllung der Bereitliste eines Prozessors zeitlich so zu gestalten, dass es nicht zum ↑Leerlauf des betreffenden Prozessors kommt und er stattdessen immer etwas (sinnvolles) zu tun hat. Dazu ist die Aufgabenverteilung rechtzeitig anzustoßen, nämlich bevor eine der Bereitlisten leer ist, insgesamt aber noch genug Aufgaben anstehen. Der Zeitpunkt dafür ist bestimmt aus (a) der zur Aufgabenverteilung zu erwartenden ↑Ausführungszeit einerseits und (b) der zur Aufgabenbearbeitung zu erwartenden ↑Bedienzeiten pro Prozessor andererseits: die Differenz $(b) - (a)$ bestimmt die relative ↑Startzeit für die Aufgabenverteilung, um einen drohenden Leerlauf zu verhindern. Da diese Zeiten für gewöhnlich nur Schätzwerte sind und nicht selten überhaupt Unbekannte bleiben, wird der Leerlauf von Prozessoren nicht sicher verhindert werden können.

Arbeitsverzeichnis Bezeichnung für ein ↑Verzeichnis, das den gegenwärtigen ↑Namenskontext für einen ↑Prozess bildet (↑*current working directory*). In ↑UNIX ist diesem Verzeichnis der Name „.“ (↑*dot*) zugeordnet. Der Eintrag erlaubt die einfache Bestimmung vom eigenen Namenskontext, ohne den wirklichen, im ↑Elterverzeichnis eingetragenen, Kontextnamen kennen zu müssen.

Architektur Anordnung der Teile eines Ganzen zueinander; gegliederter Aufbau, innere Gliederung (Duden). Nach ↑Vitruv ist eine damit gemeinte Konstruktion auf drei Prinzipien begründet: *venustas*, Anmut, Schönheit, Wunsch; *firmitas*, Solidität, Stabilität, Wesentlichkeit; *utilitas*, Zweckmäßigkeit, Nützlichkeit, Funktion, Gut.

Architekturmerkmal Eigenschaft einer ↑Rechnerarchitektur. Beispielswiese die Fähigkeit des ↑Prozessors zur ↑Parallelverarbeitung, die Art oder der Grad von ↑Speicherkonsistenz im System, das Spezifikum eines ↑Zwischenspeichers und ob dieser bei einem ↑Multiprozessor beziehungsweise ↑Mehrkernprozessor ↑Speicherkohärenz sicherstellt oder die Besonderheit funktional dedizierter Verarbeitungseinheiten. Darüber hinaus aber auch jede charakteristische Eigenheit der Hardware, die im ↑Programmiermodell sichtbar ist und Auswirkungen auf die Ausführung von ↑Programmen haben kann. Neben dem ↑Befehlssatz beispielsweise die ↑Bytereihenfolge oder die Fähigkeit, einzelnen oder gar allen ↑Rechenkernen eine eigene ↑Taktfrequenz geben zu können, damit dann verschieden schnell zu laufen und unterschiedlich viel Energie zu verbrauchen.

Archivspeicher Bezeichnung für einen ↑Speicher zur langfristigen, dauerhaften Aufbewahrung von Informationen; geordnete Sammlung von Software und Daten jeglicher Art in digitaler Form; ↑nichtflüchtiger Speicher. Die Speicherung erfolgt rechnerunabhängig (*off-line*). Zum Zugriff auf die gespeicherten Informationen ist der auf ↑Wechseldatenträger vorrätiger Speicher, manuell oder maschinell (Roboter), zuerst an das ↑Rechensystem anzuschließen. Auch ↑Tertiärspeicher genannt.

arithmetischer Überlauf Bezeichnung für die Überschreitung der bei endlicher Zahlendarstellung definierten Kapazitätsgrenze des Operanden einer arithmetischen Operation. In technischer Hinsicht ist dieser Operand das ↑Exemplar eines ↑Datentyps fester Breite, jeder seiner

möglichen Werte ist als Dualzahl mit einer festen Anzahl von Bits repräsentiert. Beispielsweise deckt in ↑C der Datentyp `int` einen Wertebereich von mindestens $[-32767, +32767]$ beziehungsweise höchstens $[-2147483647, +2147483647]$ ab — der jeweilige negative Grenzwert ist der im Standard erlaubten Darstellung eines Zahlenwerts als Einerkomplement geschuldet, das gebräuchliche Zweierkomplement verschiebt diese untere Grenze um einen weiteren Zähler. Ein Exemplar dieses Typs ist damit durch 16 oder 32 Bits zusammenhängend im ↑Speicher dargestellt. Der Typmodifizierer `short` wird gemeinhin in dem Zusammenhang genutzt, um die kleinere (d.h., 16 Bits breite) Darstellung festzulegen. Für größere Darstellungen ist der Modifizierer `long` anzugeben, nämlich einfach (mind. 32 Bits) oder zweifach (mind. 64 Bits). Eine kleinere Darstellung wird durch `char` (mind. 8 Bits) erreicht. Tritt bei einer Berechnung bezogen auf einen bestimmten Datentyp ein Überlauf auf, wird dies im ↑Statusregister angezeigt (*overflow bit*).

ARPANET Abkürzung für (en.) *Advanced Research Projects Agency Network*, 1969–1990. Ursprünglich im Auftrag der US-Luftwaffe am MIT entwickeltes ↑Rechnernetz. Vorläufer des heutigen Internet.

arrival time (dt.) ↑Ankunftszeit.

ASCII Abkürzung für (en.) *American standard code for information interchange* (1963). Ein 7-Bit Kode, der 128 Zeichen definiert: 33 Steuerzeichen (nicht druckbar: $[\mathtt{00}_{16}, \mathtt{1F}_{16}]$, $\mathtt{7F}_{16}$) und 95 druckbare Zeichen ($[\mathtt{20}_{16}, \mathtt{7E}_{16}]$).

ASM86 Bezeichnung für einen ↑Assemblierer, Intel, für ↑x86. Erste Version 1981.

Assemblierer Softwareprozessor, der ein in ↑Assembliersprache formuliertes ↑Programm in ein semantisch äquivalentes Programm in ↑Maschinensprache umwandelt.

Assembliersprache Programmiersprache, um alle Bestandteile, aus denen sich ein ↑Maschinenprogramm schließlich zusammensetzt, zu versammeln (*assemblieren*). Die Sprachelemente dienen der Platzierung und Anordnung von ↑Text und ↑Daten in einem gemeinsamen ↑Adressraum, der symbolischen Darstellung von einem ↑Maschinenbefehl, der Vergabe von Namen für eine ↑Adresse, der Zuordnung von Attributen zu den Namen (für den ↑Binder) und der Deklaration eines solchen Namens als global sichtbar außerhalb der ↑Übersetzungseinheit. Eine Anweisung in dieser Sprache ist unterteilt in vier Komponenten wie folgt:

```
[label] mnemonic [operands] [comment]
```

Optionale Angaben sind (hier) durch eckige Klammern kenntlich gemacht. Ansonsten bezeichnen die Einzelkomponenten eine Marke (*label*), die dem an dieser Stelle geltenden Adresswert (↑Adresszähler) einen Namen gibt; ein als Gedächtnisstütze dienendes ↑Mnemon (*mnemonic*), das entweder den Operationsteil von einem ↑Maschinenbefehl oder einen ↑Pseudobefehl für den ↑Assemblierer benennt; der ebenfalls mnemonisch formulierte Operandenteil (*operands*) des Maschinenbefehls; sowie ein Kommentar (*comment*), um die Anweisung im semantischen Zusammenhang zu erklären. Ein Beispiel für ↑GAS und ↑x86 ist folgendes ↑Unterprogramm, das als Funktionswert den ↑PC-Inhalt von dem ↑Prozess liefert, der dieses Unterprogramm aufgerufen hat:

```
    .text                       # apply/add following stuff to text segment
    .p2align 4,,15              # enforce 16 byte alignment
    .globl whereami             # make symbol globally visible
whereami:                       # assume call instruction for invocation
    movl (%esp), %eax           # fetch program counter saved by call
    rts                         # return to where I came from
    .size whereami, .-whereami  # determine length of this function
```

Die hier durchgängig genutzten Kommentarfelder erläutern die Bedeutung jeder einzelnen

Anweisung. Dabei ist (für GAS) jedes Kommentarfeld durch das Rautensymbol (`#`) gekennzeichnet, wodurch der nach diesem Symbol auf der Zeile stehende Text vom Assemblierer ignoriert wird.
Pseudobefehle sind dem Assemblierer durch den anführenden Punkt kenntlich gemacht. Zunächst wird durch `.text` deklariert, dass der Bezug aller nachfolgenden Anweisungen zum ↑Textsegment ausgelegt ist. Der Befehl `.p2align` sorgt für die ↑Ausrichtung des nachfolgenden Programmabschnitts. Dieser Befehl hat drei Operanden: der erste Operand (`4`) gibt an, bis zu welcher Zweierpotenz der ↑Adresszähler weitergeschaltet und das Textsegment an dieser Stelle aufgefüllt werden soll; der zweite Operand (leer) definiert den Wert, der als Füllung verwendet werden soll; der dritte Operand (`15`) steht für eine Byteanzahl, um die der Adresszähler höchstens weitergeschaltet wird. Der Effekt dieser Anweisung ist, dass der vom Assemblierer bestimmte Wert des Symbols `whereami` nächstes ganzzahlig Vielfaches relativ zum aktuellen Adresszählerwert sein wird: hat der Adresszähler beispielsweise den Wert 42, wird `whereami` den Wert $2^4 \times 3 = 48 \geq 42$ erhalten und damit eine Lücke von sechs Bytes im Textsegment geschaffen. Der Befehl `.global` stellt die globale Sichtbarkeit von Symbolen (hier: `whereami`) her. Ohne diese Angabe hätten alle in einer Übersetzungseinheit deklarierten Symbole nur lokale Bedeutung, sie wären für andere Übersetzungseinheiten effektiv „unsichtbar“: der ↑Binder verwendet den Wert eines lokalen Symbols nicht, um eine ↑unaufgelöste Referenz gleichen Namens, die in einem anderen ↑Objektmodul vermerkt ist, aufzulösen. Indem `whereami` als globales Symbol ausgezeichnet ist, kann die Funktion dieses Namens im gesamten ↑Maschinenprogramm letztlich auch aufgerufen werden. Der Befehl `.size`, schließlich, gibt einem Symbol ein Größenattribut. Konkret führt der Assemblierer hier folgende Berechnung durch: $loc(.) - loc(whereami)$, wobei *loc* für einen Adresszählerwert steht und mit dem Punkt (in GAS) auf den jeweils aktuellen Adresszählerwert Bezug genommen wird. Ergebnis ist die von der Funktion namens `whereami` im Textsegment belegte Anzahl von Bytes, das heißt, den für diesen Textabschnitt nötigen ↑Speicherbereich.

Assemblierung Vorgang der Übersetzung von einem ↑Programm durch einen ↑Assemblierer.

AST Abkürzung für (en.) ↑*asynchronous system trap*.

asymmetric multiprocessing (dt.) ↑asymmetrische Simultanverarbeitung.

asymmetric multiprocessor (dt.) asymmetrischer ↑Multiprozessor, auch ↑asymmetrisches Multiprozessorsystem.

asymmetric scheduling (dt.) ↑asymmetrische Planung.

asymmetrische Planung (en.) ↑*asymmetric scheduling*. Modell der ↑Ablaufplanung, die wenigstens einen ↑Multiprozessor oder ↑Mehrkernprozessor zur Grundlage hat. Dabei sorgen die Verfahren bewusst nicht für eine gleichmäßige Verteilung der ↑Prozesse über die einzelnen ↑Prozessoren (↑*asymmetric multiprocessing*).
Der natürliche Grund dafür kann ein ↑asymmetrisches Multiprozessorsystem und der damit gegebene asymmetrische Aufbau der zugrunde liegenden Hardware sein. So ist der von einem Prozess generierte Befehlsstrom immer prozessorabhängig. Dies gilt für gewöhnlich für das gesamte ↑Programm, das den Prozess definiert, kann jedoch auch abschnittweise festgelegt sein: in jedem Fall ist aber dem Prozess ein Prozessor zuzuteilen, der den von diesem Prozess jeweils generierten Befehlsstrom auszuführen vermag. Die ↑Einplanung muss auf die charakteristischen Merkmale der Prozesse und ihrer Prozessoren Rücksicht nehmen.
Andererseits kann allein das ↑Betriebssystem diese Form der Ablaufplanung vorgeben, insbesondere bei symmetrischer Auslegung der Hardware. Hier ist vor allem die ↑Architektur des Betriebssystem zu nennen, die etwa aus Gründen der ↑Synchronisation und, in dem Zusammenhang, zur Reduzierung von ↑Interferenz die Prozessoren logisch getrennt voneinander betreibt. Dazu ist jedem Prozessor eine eigene ↑Bereitliste zugeordnet, das heißt, auf die diese Liste implementierende Datenstruktur wird nicht von verschiedenen Prozessoren aus zugegriffen. Die Prozesse, die diese Listen letztlich füllen und leeren, sind nicht gekoppelt,

sie agieren rein lokal, müssen sich nicht mit Prozessen anderer Prozessoren synchronisieren und können ungestört voranschreiten. Der Nachteil dieser Organisation besteht dann jedoch darin, dass ein Prozessor in den ↑Leerlauf eintreten kann, obwohl ein anderer Prozessor noch mehr als genug ↑Aufgaben zu erledigen hat.
Zur Abhilfe dieses Problems unausgewogener Arbeitslasten gibt es zwei Ansätze, ↑Arbeitsteilung und ↑Arbeitsentzug, die zwar nicht zu einer zwingend anhaltenden Kopplung der auf die Listen zugreifenden Prozesse führen, jedoch die betreffenden Prozesse nicht mehr ohne gegenseitige Beeinflussung stattfinden lassen: erfolgen Arbeitsteilung oder -entzug gleichzeitig zu sonst lokalen Operationen, die den Zustand einer Bereitliste verändern können, müssen die beteiligten Prozesse sich synchronisieren und werden sich dadurch auch zwingend gegenseitig beeinflussen. Diese mögliche Beeinflussung greift jedoch erst, wenn (1) die Untätigkeit einzelner Prozessoren droht und (2) mehr Aufgaben im ↑Prozesszustand bereit sind als Prozessoren zur Verfügung stehen.
Ein anderer Aspekt, der zur asymmetrischen Nutzung einer sonst symmetrisch ausgelegten Hardware führt, ist die Reservierung von Prozessoren für Prozesse, die ganz bestimmte Funktionen ausüben. Dies kann Prozesse sowohl einer ↑Anwendung als auch des Betriebssystems betreffen, die an einen Prozessor verankert und diesen gegebenenfalls sogar exklusiv nutzen werden. An sich gibt es wenige ↑Systemfunktionen, die nicht derart auf eigene Prozessoren platziert werden könnten. Entscheidend ist die strukturelle Einbindung dieser Funktionen im Betriebssystem, wie häufig sie aktiviert werden und wie kostspielig dann ihr „Fernaufruf“ im Vergleich zu ihrer ↑Ausführungszeit ist. Beispielsweise kann es sich ab einer bestimmten Anzahl wettstreitiger Prozesse lohnen, wenn bereits ein ↑kritischer Abschnitt einen eigenen Prozessor zur Ausführung zugeteilt bekommt. Der sequentielle Durchlauf der wettstreitigen Prozesse sorgt für starke Lokalität der ↑Daten im ↑Zwischenspeicher, wodurch die Systemleistung erheblich verbessert werden kann.

asymmetrische Simultanverarbeitung (en.) ↑*asymmetric multiprocessing*. Eine Variante der ↑Simultanverarbeitung, die ein ↑asymmetrisches Multiprozessorsystem zur Grundlage hat.

asymmetrisches Multiprozessorsystem (en.) ↑*asymmetric multiprocessor* (↑AMP). Bezeichnung für ein ↑Rechensystem, das aus wenigstens einen ↑Multiprozessor oder ↑Mehrkernprozessor aufgebaut ist und dessen Recheneinheiten asymmetrisch ausgelegt sind. Die Asymmetrie ist in der ↑Heterogenität der Hardware begründet, etwa wenn der ↑Befehlssatz der Prozessoren uneinheitlich ist und demzufolge Prozesse nur auf die für sie geeigneten Prozessoren stattfinden sollten (↑*asymmetric scheduling*). Typisches Beispiel dafür ist ein aus ↑CPU und ↑GPU bestehendes Rechensystem, wobei jede Klasse dieser ↑Prozessoren jeweils durch mehrere ↑Exemplare ausgeprägt sein kann.

asynchrone Ausnahme Bezeichnung für eine ↑Ausnahme, die von dem betrachteten ↑Prozess nicht selbst verursacht wurde. Der durch diesen Prozess beschriebene ↑Programmablauf wird unterbrochen (↑*interrupt*), dadurch verzögert, aber zu einem späteren Zeitpunkt fortgesetzt. Verursacher einer solchen Ausnahme ist ein zum Bezug genommenen Programmablauf externer Prozess auf einem anderen ↑Prozessor oder in der ↑Peripherie. Die auf dem Prozessor dieses Programmablaufs entstehende ↑Ausnahmesituation ist unvorhersehbar und nicht reproduzierbar. Typisches Beispiel dafür ist eine ↑Unterbrechungsanforderung (↑IRQ, ↑NMI), aber auch ein ↑Seitenfehler, nämlich wenn ↑virtueller Speicher eine ↑globale Ersetzungsstrategie benutzt.

asynchronous system trap (dt.) asynchrone ↑Ausnahme im ↑Betriebssystem. Bezeichnung sowohl für einen Mechanismus in einem (realen/abstrakten) ↑Prozessor als auch für eine Art von ↑Ereignis auf ↑Systemebene: der plötzliche Aufruf eines ↑Unterprogramms im Betriebssystem außerhalb des Hauptausführungsstrangs. Die Idee besteht darin, als Folge einer bestimmten Operation (↑*system mode*) einen Vorgang asynchron im Betriebssystem stattfinden zu lassen, bevor die zuvor unterbrochene Ausführung (↑*user mode*) eines ↑Maschinenprogramms wieder aufgenommen und dadurch die ↑Benutzerebene reaktiviert wird. Ein in

↑RSX 11 erstmalig verwirklichtes Konzept für den ursprünglichen Zweck, eine in Bearbeitung befindliche ↑Aufgabe über das Auftreten eines bestimmten Ereignisses (z.B. Beendigung der ↑Ein-/Ausgabe) zu informieren.
Der Mechanismus ist dem herkömmlichen Aufruf eines ↑Unterprogramms nicht ganz unähnlich, nur dass dieser eben asynchron zugestellt wird (↑DPC) und einen speziellen ↑Ausnahmehandhaber aktiviert. Auslöser ist eine auf Systemebene und durch Software initiierte ↑Unterbrechungsanforderung, die aber erst behandelt wird, wenn das Betriebssystem seine Aufgabe (z.B. einen ↑Systemaufruf oder eine ↑Unterbrechungsbehandlung) beendet hat und beabsichtigt, wieder in Bereitschaft zu gehen (d.h., sich zu deaktivieren) und zur Benutzerebene zurückzukehren. Typisches Beispiel für solch eine Funktion ist die Auslösung einer Unterbrechungsbehandlung zweiter Stufe (↑SLIH).
Mit der ↑VAX, später mit dem ↑Alpha, wurde der Mechanismus direkt durch Hardware unterstützt. Um einen ↑AST auszulösen muss die Nummer der gewünschten Privilegebene (auch Zugriffsmodus) in ein spezielles ↑Prozessorregister (`PR$_ASTLVL`: *processor AST level register*) eingetragen werden. Zusätzlich wird ein ↑Deskriptor, der den AST-Handhaber (SLIH) identifiziert, in eine ↑Warteschlange eingereiht. Sobald die CPU aus der Unterbrechungsbehandlung zurückkehrt (`REI`: *return from exception or interrupt*), prüft sie den dabei zu reaktivierenden Zugriffsmodus. Hat der aktuelle Modus höhere Privilegien als der zu reaktivierende Modus und ist ein AST anhängig, aktiviert die CPU automatisch den entsprechenden Unterbrechungshandhaber und startet diesen auf ↑Unterbrechungsprioritätsebene 2. Dieser Handhaber arbeitet alle in der AST-Warteschlange enthaltenen Aufgaben ab.
Die ↑PDP 11 Familie bot rudimentäre Unterstützung durch im ↑PSW enthaltene Statusinformationen: dem T-Bit (für ↑*trap*) und einer Angabe darüber, welcher ↑Arbeitsmodus vor der Unterbrechung aktiv war (*previous mode*). Kommt es zu einer Unterbrechung, sichert die CPU automatisch unter anderem das aktive PSW auf den ↑Laufzeitstapel der Systemebene und wechselt den Arbeitsmodus entsprechend. Das T-Bit kann in jedem Arbeitsmodus gesetzt werden, um nach der Ausführung eines ↑Maschinenbefehls eine ↑Ausnahme hervorzubringen. Für gewöhnlich wird damit die Fehlersuche (*debugging*) unterstützt. Um nun einen AST auf der Systemebene auszulösen, muss das T-Bit im gesicherten PSW gesetzt werden. Dazu prüft das Betriebssystem vorher, in welchen Arbeitsmodus bei Beendigung der Unterbrechungsbehandlung (FLIH) zurückgekehrt wird. Ist im aktiven PSW die Benutzerebene (*user mode*) als voriger Arbeitsmodus ausgewiesen, wird das T-Bit im gesicherten PSW gesetzt. Im nächsten Schritt muss das Betriebssystem die CPU mittels `RTI` (*return from interrupt*) — nicht jedoch mit `RTT` (*return from trap*) — aus der Unterbrechungsbehandlung zurückkehren lassen, um nämlich bei gesetztem T-Bit im dann vom Laufzeitstapel wieder hergestellten PSW noch vor Ausführung des nächsten Maschinenbefehls (auf Benutzerebene) sofort wieder eine Unterbrechungsbehandlung aufzunehmen.
Hardwarehilfe, die bei Rückkehr zur Benutzerebene und den damit erfolgten vollständigen Abbau des Laufzeitstapels automatisch eine ↑Ausnahmebehandlung einleitet, aber fehlt, lässt sich ansatzweise in Software nachbilden, um einen AST-ähnlichen Mechanismus im Betriebssystem bereitstellen zu können. Hierzu ist der Aufruf des Unterbrechungshandhabers erster Stufe (↑FLIH) in eine spezielle ↑Mantelprozedur einzubetten. Diese Prozedur führt beispielsweise Buch über die aktiven Handhaberinkarnationen, das heißt, zählt deren Verschachtelungen, und löst den AST aus, wenn keine verschachtelte Inkarnation mehr vorliegt. In dem Fall existiert nur noch eine Handhaberinkarnation, nämlich diejenige, die der Ausgang für die Kaskadenbildung war. Normalerweise erfolgt bei Auflösung dieser ersten/letzten Inkarnation die Rückkehr zur Benutzerebene, ausnahmsweise wird in dem Moment jedoch bei anhängigem AST eine zusätzliche Unterbrechungsbehandlung durchgeführt.
Das Problem einer solchen Nachbildung ist es, die Verbuchung einer jeden Handhaberinkarnation sicherzustellen. Auch wenn dies durch eine einfache Zähloperation bewerkstelligt werden kann, ist jedoch bei weitem nicht garantiert, dass jede Zählung erfolgt: kritisch ist hier die ↑kaskadierte Unterbrechung. Sogar wenn die Zähloperation der erste Maschinenbefehl eines jeden Unterbrechungshandhabers ist, hängt es von der CPU ab, ob dieser Befehl unter allen Umständen zur Ausführung kommt. Erlaubt die CPU bereits die nächste Un-

terbrechung vor dem ersten Befehl im Unterbrechungshandhaber, wird die neue Inkarnation nicht gezählt. Demzufolge kann mehr als eine Inkarnation aktiv sein, ein AST dürfte somit nicht behandelt werden. Allerdings ist dieser Umstand aufgrund einer noch nicht geschehenen Zählung nicht immer erkennbar, weshalb fälschlicherweise ein AST im falschen Moment zur Behandlung kommt. Wenn dann zu dieser Behandlung noch eine niedrigere Unterbrechungsprioritaetsebene eingestellt wird, was für gewöhnlich der Fall ist, kann der Laufzeitstapel im Betriebssystem überlaufen, wodurch möglicherweise ↑Panik entsteht.

Exkurs Für den ↑x86 kann trotz ↑NMI für gewöhnlich eine sichere Nachbildung erreicht werden. Zum einen deshalb, weil die CPU die Kaskadierung eines NMI nicht zulässt: erst wenn der Handhaber für den NMI seine Ausführung vollständig beendet hat (`iret`), wird der nächste NMI zugelassen. Zum anderen profitiert die Nachbildung von der Tatsache, dass die CPU, je nach Modell, kaskadierte Unterbrechungen der Art ↑IRQ entweder nicht unterstützt oder programmierbar macht. Unterbrechungsprioritätsebenen führt erst der ↑PIC ein, x86 und damit die CPU kennt nur eine Ebene. Vor diesem Hintergrund startet die Unterbrechungsbehandlung wie folgt:

```
#define __attribute__ ((interrupt)) void irqstub_t

irqstub_t irq000(train_t *this) { guide(flih000, this); }
irqstub_t irq001(train_t *this) { guide(flih001, this); }
/* ... */
irqstub_t irq256(train_t *this) { guide(flih256, this); }
```

Für jeden IRQ ist ein ↑Unterbrechungshandhaberstumpf definiert, der lediglich den Aufruf an die zentrale Mantelprozedur (`guide`) kodiert, um den weiteren Verlauf in die entsprechenden Bahnen zu lenken. Die Aufrufparameter identifizieren den eigentlichen Unterbrechungshandhaber (FLIH) und den auf dem Laufzeitstapel gesicherten (minimalen) ↑Prozessorstatus des unterbrochenen ↑Prozesses. Die dem entsprechende Mantelprozedur für einen Unterbrechungshandhaber erster Stufe, die einen AST, nämlich den Unterbrechungshandhaber zweiter Stufe, letztlich durch das System leitet, sei wie folgt ausgelegt:

```
__attribute__ ((fastcall)) void guide(flih_t flih, train_t *this) {
    stage(this, ENTER);                  /* enter IRQ handler incarnation */

    admit(IRQ | EDGE);             /* take OS priority, if edge triggered */
    flih(this);                                    /* invoke actual FLIH */
    avert(IRQ | EDGE);       /* return to CPU priority, if edge triggered */

    if (stage(this, PROVE | USER)) {    /* on the way back to user level? */
        while (backlog(annex())) {               /* yes, any pending AST? */
            admit(IRQ);                          /* yes, take OS priority */
            flush(annex());                       /* forward pending ASTs */
            avert(IRQ);                         /* return to CPU priority */
        }
    }

    stage(this, LEAVE);                  /* leave IRQ handler incarnation */
}
```

Für gewöhnlich beginnt die Ausführung dieser Prozedur mit implizit gesetzter ↑Unterbrechungssperre (*interrupts disabled*), das heißt, der durch die Hardware in dem Moment vorgegebenen Unterbrechungsprioritätsebene der CPU. Gegebenenfalls ist diese Prioritätsebene in der CPU oder dem PIC auch programmierbar, woraufhin die totale Unterbrechungssperre nicht zwingend ist. Keinesfalls darf jedoch eine unkontrollierte Rücknahme dieser vorgegebenen Sperre geschehen, da sonst, insbesondere bei ↑Pegelsteuerung, der Laufzeitstapel überlaufen und dadurch Panik hervorgerufen werden könnte. Den Prozedurrumpf klammert eine Operation (`stage`) ein, um bei fehlender Hardwarehilfe für einen AST dennoch (z.B. durch Zählen, s.o.) die Verbuchung von Handhaberinkarnationen zu ermöglichen. Bei gege-

bener Hardwarehilfe (x86 ab ↑i286) ist diese „Operationsklammer“ wirkungslos.
Nach dem Betreten der Handhaberinkarnation startet die eigentliche Unterbrechungsbehandlung erster Stufe durch Aufruf (`flih`) der dem FLIH entsprechenden Prozedur. Liegt dem IRQ ↑Flankensteuerung zugrunde, darf vor dem Aufruf — aber erst nach eventuell erforderlicher Inkarnationsverbuchung (`stage`: `ENTER`) — die Unterbrechungssperre aufgehoben, das heißt, zur Prioritätsebene des Betriebssystems gewechselt werden. Nach Rückkehr aus dem FLIH wird wieder zur Prioritätsebene der CPU zurück gegangen, um nämlich dem Problem (↑*lost wake-up*) vorzubeugen, einen beim Verlassen der Unterbrechungsbehandlung eventuell anhängig werdenden AST zu verpassen.
Darf ein AST geliefert werden, weil die Rückkehr zur Benutzerebene ansteht (`stage`: `PROVE`), wird der Auftragsbestand (`backlog`) geprüft und gegebenenfalls die Auftragswarteschlange (`annex`) abgearbeitet. Beim x86, und zwar ab dem Modell i286, kann für diese Prüfung der auf dem Laufzeitstapel beim ↑Unterbrechungszyklus gesicherte (und hier durch `this` erreichbare) ↑Segmentwähler für das ↑Textsegment genutzt werden. Die Benutzerebene ist normalerweise mit den geringsten Privilegien (↑DPL 3) ausgestattet. Gibt der gesicherte Segmentwähler eben diese Priviligstufe (mit dem Wert 3) vor, wird angenommen, dass bei Rückkehr aus der Mantelprozedur (`guide`) der Wechsel zur Benutzerebene stattfindet. Sind mangels Hardwarehilfe die Handhaberinkarnationen zu zählen, zeigt ein Zählerwert von 1 diesen Zustand an. In dem Fall ist jedoch vorauszusetzen, dass mittels NMI gestartete und damit kaskadierte Unterbrechungshandhaber nicht durch dieselbe Mantelprozedur gelenkt werden (s.o.: Problem der Nachbildung durch Zählen), sondern nur welche, die einen IRQ zur Ursache haben.
Die Abarbeitung der Auftragswarteschlange (`flush`) geschieht auf der für das Betriebssystem definierten Prioritätsebene, weshalb im Falle des x86 die Unterbrechungssperre aufgehoben wird. Da während der Abarbeitung jederzeit ein weiterer AST ausgelöst werden kann, ist vor Rückkehr aus der Unterbrechungsbehandlung der Auftragsbestand erneut zu prüfen. Hierzu ist aus zuvor bereits erwähntem Grund erneut die Prioritätsebene der CPU einzunehmen, die Unterbrechungssperre also zu setzen.
Gemeinhin ist ein Unterbrechungshandhaber erster Stufe einer Prozedur im programmiersprachlichen Sinn (↑*C function*) sehr ähnlich, allerdings erfolgen Aufruf von und Rückkehr aus dem entsprechenden Unterprogramm auf anderem Wege. Der ↑Aktivierungsblock dieses Unterprogramms unterscheidet sich stark von dem einer normalen Prozedur (keine Argumente, gesicherter Prozessorstatus anstatt gesicherte Rücksprungadresse), weshalb es auch nicht mit einem Prozedur- (`ret`), sondern Ausnahmerücksprungbefehl (`iret`) beendet wird. Da dieser Unterbrechungshandhaber aber durch eine Mantelprozedur gesteuert und ebenfalls als Prozedur aufgerufen wird, besitzt er einen ganz normalen Aktivierungsblock und führt am Ende auch einen Prozedurrücksprung aus:

```
void flih000(train_t *this) {
    static dpc_t slih = { 0, slih000, 42 };              /* AST descriptor */
    /* ... */
    defer(annex(), &slih);                                  /* release AST */
}
```

Die hier wichtigen Punkte sind einerseits die Programmierung eines ↑Deskriptors für einen AST, um den dementsprechenden Unterbrechungshandhaber zweiter Stufe zu erfassen, und anderseits Bereitstellung eben dieses Deskriptors, um den AST beziehungsweise SLIH zu gegebener Zeit auszulösen. Letzteres geschieht durch Zurückstellung (`defer`) eines Prozeduraufrufs (DPC). Ob überhaupt ein SLIH stattfindet und daher ein AST ausgelöst wird, ist Ermessenssache des FLIH. Darüberhinaus kann ein FLIH mehr als einen SLIH nach sich ziehen, indem er mehr als einen AST auslöst beziehungsweise Deskriptor zurückstellt.
Der Unterbrechungshandhaber zweiter Stufe ist ebenfalls eine gewöhnliche Prozedur. Es bietet sich an, diese Prozedur mit einem „generischen Argument“ zu versehen, durch das der FLIH Parameter an den SLIH übergeben kann. Diese Parameter (im gezeigten Beispiel der Wert `42`) sind ebenfalls im Deskriptor für den SLIH aufgeführt und werden ihm automatisch

übergeben, wenn der betreffende Deskriptor (durch `flush`) der AST-Warteschlange entnommen wird. Nachfolgend das Skelett eines solchen SLIH:

```
void slih000(data_t data) {
    /* ... */
}
```

Als AST ausgelöst wird diese Prozedur asynchron zum gegenwärtigen Hauptausführungsstrang im Betriebssystem gestartet. Gegebenenfalls sind die von ihr angestoßenen ↑Aktionen daher mit dem sonst im Betriebssystem noch stattfindenden ↑Prozessen zu synchronisieren. Auch wenn mit einem AST letztlich die ↑Sequentialisierung bestimmter Vorgänge einher geht, finden damit längst nicht alle Vorgänge im Betriebssystem sequentialisiert statt.

atomare Operation Operation, für die ↑Unteilbarkeit gilt. Eine solche Operation kann weder durch eine ↑Unterbrechung aufgespalten werden, noch durch ↑Parallelverarbeitung gleichzeitig mehrfach zur Ausführung gelangen.

Aufgabe Etwas, was einem ↑Prozess zu tun aufgegeben ist (in Anlehnung an den Duden). Ein Auftrag, eine bestimmte Funktion zu erfüllen. Dieses Etwas kann implementiert sein als ein ↑Unterprogramm, also keinen eigenständigen ↑Handlungsstrang definieren oder, umgekehrt, sehr wohl durch einen eigenständigen Handlungsstrang realisiert sein, der dann ein oder mehrere Unterprogramme autonom und in einer bestimmten Reihenfolge ausführt. Im letzteren Fall repräsentiert solch ein Unterprogramm eine ↑Teilaufgabe (↑*job*), die dann auch als kleinste Ausführungseinheit verstanden wird. Aufgabe wie auch Teilaufgabe sind Einheiten einer ↑Ablaufplanung, in der dann ein Prozess eben durch solch eine Einheit bestimmt ist.

Aufrufkonvention Methode, nach der ein ↑tatsächlicher Parameter einem ↑Unterprogramm übergeben wird. Ist das Unterprogramm eine Funktion, legt die Methode zusätzlich die Art und Weise der Rückgabe des Funktionswerts fest. Platzhalter für die Über- beziehungsweise Rückgabewerte sind Prozessorregister oder liegen auf dem ↑Laufzeitstapel, mit fester Zuordnung der Parameterposition zu einem Prozessorregister oder Festlegung der Reihenfolge beim Stapeln der Parameter (↑cdecl: von rechts nach links). Darüber hinaus wird bestimmt, welche Prozessorregister innerhalb des Unterprogramms frei verwendbar sind. Hierzu erfolgt eine Differenzierung zwischen ↑flüchtiges Register und ↑nichtflüchtiges Register, wobei nur die Inhalte letzterer invariante Merkmale bei der Unterprorgammausführung darstellen.

Auftrag Weisung; einem ↑Rechensystem zur Erledigung übertragene ↑Aufgabe (in Anlehnung an den Duden). Die damit verbundene Aufforderung wird von einem ↑Kommandointerpreter entgegengenommen, der die zu erledigende Aufgabe prüft und, falls gültig beziehungsweise zulässig, alle dazu benötigten Handlungen veranlasst. Einerseits ist eine solche Aufgabe durch ein eigenständiges ↑Programm implementiert, das dann durch einen eigenen ↑Prozess zur Ausführung gebracht wird. Andererseits kann die Aufgabe aber auch als ein ↑Unterprogramm des Kommandointerpreters ausgelegt sein, dass dann im Kontext eines bereits stattfindenden Prozesses (nämlich dem ↑Interpreter) aufgerufen wird und abläuft (↑*built-in command*). Die Aufgabe kann darin bestehen, ↑Betriebsmittel anzufordern, damit ein vorgesehenes Programm ausgeführt werden, dieses Programm zu starten und mit bestimmten Geräten der ↑Peripherie zu verknüpfen, Zwischenergebnisse für ein nachgeschaltet auszuführendes Programm zu speichern, schließlich gestartete Programme zu beenden und angeforderte Betriebsmittel wieder freizugeben. Eine derart komplexe Aufgabe ist für gewöhnlich als Anweisungsfolge beschrieben und wird durch ↑Stapelverarbeitung ausgeführt.

Auftragssteuersprache Synonym zu ↑Kommandosprache.

Ausführungszeit (en.) ↑*execution time*. Bezeichnung für eine ↑Prozessgröße, die die effektive Zeitspanne für die Durchführung einer ↑Aufgabe quantifiziert. Grundsätzlich wird zwischen der minimalen (↑BCET) und maximalen (↑WCET) Zeitspanne unterschieden. Darüber hinaus kann die Zeitspanne (ggf. auch zwischen diesen beiden Extremen, soweit diese bekannt

sind) auch einem Durchschnittswert (↑ACET) entsprechen.
Bezugspunkt in allen Fällen ist die Anzahl von Taktzyklen, die bei der Aufgabendurchführung von der ↑CPU benötigt werden. Diese Anzahl lässt sich durch Laufzeitmessung oder Programmanalyse bestimmen: sie ist einerseits durch den kürzesten und längsten Pfad des diese Aufgabe entsprechenden ↑Programms gegeben und anderseits durch die Anzahl von ↑Unterbrechungen, den der betreffende ↑Prozess unterworfen sein kann. Muss die Aufgabe in ↑Echtzeit durchgeführt werden, ist, je nach ↑Echtzeitbedingung, für jeden dabei anfallenden ↑Befehlszyklus ↑Vorwissen zur maximalen Anzahl der zu erwartenden ↑Prozessortakte wünschenswert oder unabdinglich. Verrechnet mit der ↑Taktfrequenz der CPU ergibt sich daraus die für die Ausführung der Aufgabe zu erwartende absolute Zeit.

Auslastung Grad der anhaltenden Nutzung von einem ↑Betriebsmittel; der Bruchteil der Zeit, den dieses Betriebsmittel nicht brachliegt.

Ausnahme Sonderfall, eine Abweichung vom normalen ↑Programmablauf. Der ↑Prozess wird unterbrochen und gegebenenfalls nach erfolgter ↑Ausnahmebehandlung wieder aufgenommen.

Ausnahmebehandlung Vorgang zur Bearbeitung eines Sonderfalls bei der Programmausführung. Ursache ist eine ↑Ausnahmesituation in der (realen/virtuellen) Maschine, auf der ein ↑Programm abläuft: die in diesem Programm direkt/indirekt kodierte ↑Aktion oder ↑Aktionsfolge, beispielsweise ein Befehl der ↑CPU oder eine Betriebssystemfunktion ausgelöst durch einen ↑Systemaufruf, kann nicht normal vollendet werden. Für die Bearbeitung eines solchen Sonderfalls ist ein Programm (↑Ausnahmehandhaber) vorzusehen, das auf der Maschine, deren Aktion/Aktionsfolge die Ausnahme erhebt, zur Ausführung kommt. Typisches Beispiel ist eine Routine zur ↑Unterbrechungsbehandlung in oder zur Signalbehandlung auf einem Betriebssystem, also ein Programm für eine reale beziehungsweise ↑virtuelle Maschine.

Ausnahmehandhaber (en.) ↑*exception handler*. Bezeichnung für einen speziellen, ausschließlich zur ↑Ausnahmebehandlung ausgelegten und vorgesehenen ↑Handhaber.

Ausnahmesituation Umstand, der eine ↑Aktion dazu bringt, eine ↑Ausnahme zu erheben. Im Falle der in einem ↑Maschinenprogramm beschriebenen Aktion beispielsweise ein ungültiger ↑Maschinenbefehl beziehungsweise ↑Systemaufruf, eine ↑Schutzverletzung oder ein ↑Seitenfehler beim ↑Abruf- und Ausführungszyklus.

Ausnahmezuteiler (en.) ↑*exception dispatcher*. Bezeichnung für eine spezielle ↑Mantelprozedur, die durch eine ↑Ausnahme aktiviert wird, daraufhin den zuständigen ↑Ausnahmehandhaber identifiziert und aufruft und gegebenenfalls, nämlich in Abhängigkeit von der Art der Ausnahme, für die Fortsetzung des ausnahmsweise unterbrochenen ↑Programmablaufs sorgt. Für gewöhnlich übernimmt diese Prozedur für eine Gruppe solcher Handhaber zentrale Verwaltungsaufgaben, beispielsweise die ↑Zustandssicherung und -wiederherstellung für den unterbrochenen ↑Prozess, die Protokollierung seiner ↑Benutzerzeit und ↑Systemzeit, sonstige ↑Buchführung oder gar die ↑Sequentialisierung der ↑Unterbrechungsbehandlung zweiter Stufe (↑SLIH).

Exkurs Typische Art einer solchen Mantelprozedur ist der ↑Systemaufrufzuteiler ....

Ausrichtung Linienführung einer ↑Adresse entsprechend der Größe von einem an einer bestimmten ↑Speicherstelle liegenden ↑Exemplar von ↑Text oder ↑Daten. Die Führung kann durch die ↑CPU vorgegeben oder empfohlen sein, da sonst der Zugriff scheitert (↑*trap*) oder langsamer verläuft (Zwischenzyklus). Maßgeblich ist die ↑Informationsstruktur (insb. ↑Wortbreite) der CPU. Eine ↑Assembliersprache bietet daher (für gewöhnlich) Anweisungen, um eine bestimmte Anordnung von Text oder Daten festlegen zu können (↑Pseudobefehl, z.B. `.p2align`). Typischerweise erzeugt ein (optimierender) ↑Kompilierer solche Anweisungen und sorgt somit dafür, dass beispielsweise Zugriffe auf Datentypexemplare immer mit einer zur Exemplargröße ausgerichteten Adresse geschehen: `char` gerade/ungerade, 8-Bit; `short` gerade, 16-Bit; `int`/`long` gerade, 32-Bit; `long long` gerade, 64-Bit.

automatisierte Rechnerbestückung ↑Betriebsart, bei der das ↑Rechensystem gesteuert durch einen ↑Kommandointerpreter nacheinander mit Arbeitspaketen bestückt wird, wobei jedes einzelne Paket durch ein ↑Programm beschrieben ist, von dem zu einem Zeitpunkt stets nur eins zur Ausführung bereit im ↑Hauptspeicher liegt (↑*uniprogramming*). Das Rechensystem führt automatisch und ohne Eingriffe einer Bedienperson (↑*operator*) alle erforderlichen Schritte aus, um die zur Ausführung bestimmten Programme nacheinander einzulesen, zu übersetzen, zu laden und auszuführen. Die Bedienperson sorgt lediglich dafür, einen ganzen ↑Stapel von Rechenaufgaben (↑*batch job*), einen ↑Auftrag, am Eingang in Empfang zu nehmen, in das Rechensystem einzuspeisen, das in aller Regel auf ↑Tabellierpapier ausgedruckte Ergebnis der durchgeführten Berechnungen dem Drucker zu entnehmen und in den Ausgang zur Abholung zu legen.
Bedeutet ↑manuelle Rechnerbestückung noch, jeden einzelnen (oben genannten) Schritt durch Bedienpersonal auszulösen, ist der Mensch nunmehr weitestgehend aus der Verarbeitungskette herausgenommen. Der dadurch wesentlich gesteigerte ↑Durchsatz erfordert allerdings einen für ein Systemprogramm (↑*resident monitor*) reservierten Platz im Hauptspeicher, um solche Programmstapel verarbeiten zu können. Da zudem nicht jeder Auftrag dieselbe Reihenfolgebildung von Programmen impliziert, ist diese für den jeweiligen ↑Stapelauftrag immer wieder explizit durch Anweisungen einer ↑Auftragssteuersprache (↑JCL) festzulegen. Darüber hinaus muss für eine in Bezug auf nachfolgende Programme im Stapel vernünftige Fehlerbehandlung gesorgt sein, sollte nämlich ein ↑Programmablauf scheitern. Alle dafür notwendigen Softwarevorkehrungen, in Ergänzung zum Kommandointerpreter und zu grundlegenden Ein-/Ausgabeprozeduren, resultieren in Programme, die in ihrer Gesamtheit ein „embryonales ↑Betriebssystem“ bilden (↑FMS).

background noise (dt.) ↑Hintergrundrauschen.

bad block (dt.) fehlerhafter, ↑defekter Block.

bad-block management (dt.) ↑Defektblockverwaltung.

base/limit register (dt.) ↑Segmentregister.

batch (dt.) ↑Stapel.

batch job (dt.) ↑Stapelauftrag.

batch mode (dt.) ↑Stapelbetrieb.

batch monitor (dt.) ↑Stapelmonitor.

batch process (dt.) ↑Stapelprozess.

batch processing (dt.) ↑Stapelverarbeitung.

Bauwesen Gesamtheit dessen, was mit dem Errichten von Bauten zusammenhängt (Duden). Bei dem hier gemeinten Bau handelt es sich um das ↑Betriebssystem.

BCET Abkürzung für (en.) *best-case* ↑*execution time*, kleinste anzunehmende ↑Ausführungszeit.

Bearbeitungszeit (en.) ↑*processing time*. Bezeichnung für eine ↑Prozessgröße, die die Zeitspanne bis zur vollständigen Bearbeitung einer ↑Aufgabe quantifiziert. Häufig auch gleichgesetzt mit der ↑Durchlaufzeit von dem ↑Prozess, der diese Aufgabe entweder allein oder mit mehreren anderen (kooperierenden) Prozessen zusammen bearbeitet. Genau genommen ist diese Zeitspanne allerdings nur die Summe aller bei der Aufgabenbearbeitung durch die direkt beteiligten Prozesse anfallenden ↑Bedienzeiten: in der Durchlaufzeit eines Prozesses sind nämlich auch all seine ↑Wartezeiten enthalten. In logischer Hinsicht entspricht die kumulierte Zeit einem (relativen) Zeitraum, dessen Anfang und Ende auf einer (relativen) Zeitskala durch Zeitpunkte scheinbar derart festlegt ist, als wenn der Prozess bis zur Fertigstellung der Aufgabe durchläuft (↑*run to completion*).

Bedienstation Bezeichnung einer Verarbeitungseinheit (↑Prozessor), die in der Lage ist, einen an sie gestellten ↑Auftrag zu verarbeiten. Für gewöhnlich sind die Aufträge pro Station gleichartig, sie bestimmen sich durch die funktionalen Merkmale der mit der Station (in Hard-/Software) assoziierten ↑Entität. So ist beispielsweise ein ↑Prozess ein Auftrag an die ↑CPU als Entität, ein ↑Programm zu verarbeiten: er befindet sich dazu im ↑Prozesszustand bereit. Demgegenüber ist ein blockierter Prozess als solch eine Entität mit einem Auftrag verknüpft, der ihn von einem anderen (internen/externen) Prozess zur Verarbeitung erst noch zugestellt werden muss: etwa ein Auftrag, der die Beendigung einer Ein-/Ausgabeoperation (↑Ein-/Ausgabestoß) anzeigt, mit dem ein ↑kritischer Abschnitt wieder zugänglich gemacht wird oder der ein ↑konsumierbares Betriebsmittel zu Verfügung stellt.
Sind zu einem Zeitpunkt mehr Aufträge anhängig als Stationen zur Verfügung stehen, bildet sich eine ↑Warteschlange von Aufträgen, die nicht sofort der Verarbeitung zugeführt werden können. Nur in dieser Situation ist die ↑Einplanung von Aufträgen erforderlich, die damit über die Reihenfolge der ↑Einlastung einer jeweiligen Verarbeitungseinheit entscheidet.

Bedienzeit (en.) ↑*service time*. Bezeichnung für eine ↑Prozessgröße, die die Zeitspanne quantifiziert, in der eine ↑Aufgabe an einer ↑Bedienstation durchgeführt wird. Handelt es sich bei der Aufgabe um einen ↑Prozess und bei der Bedienstation um die ↑CPU, dann entspricht diese Zeitspanne dem ↑Rechenstoß des Prozesses bis zum nächsten ↑Prozesswechsel.

Bedingungssynchronisation Synonym zu ↑unilaterale Synchronisation.

Bedingungsvariable Synchronisationsvariable, deren gespeicherter Wert unzugänglich ist für einen ↑Prozess; Programmierkonvention. Kommt in einem ↑Monitor zur Anwendung, um den gegenwärtigen Prozess auf ein ↑Ereignis zu synchronisieren und bei der Blockierung implizit den wechselseitigen Ausschluss aufzuheben. Macht die ↑unilaterale Synchronisation von Prozessen möglich, unter der Prämisse, dass zu einem Zeitpunkt höchstens ein Prozess im Monitor sein darf.
Auf der Variablen sind nur zwei Operationen definiert: `wait` (a. `await`), um das durch die Variable repräsentierte Ereignis zu erwarten, den Prozess zu blockieren, und `signal` (a. `cause`), um eben dieses Ereignis anzuzeigen, einen Prozess zu deblockieren. Erwartet kein Prozess das Ereignis, ist `signal` wirkungslos. Dies bedeutet aber auch, dass eine Ereignisanzeige in Form eines Zustands nicht in der Variablen gespeichert wird. Im Gegensatz dazu speichert `wait` den Zustand, dass einer oder mehrere Prozesse in Erwartung auf ein Ereignis blockiert sind. In dem Fall ist mit jedem ↑Exemplar einer solchen Variablen eine ↑Warteschlange von Prozessen assoziiert
Untrennbar verbunden mit dem Monitor, das heißt, einem Konzept der *Typisierung* in höheren Programmiersprachen, indem nämlich die korrekte Verwendung von Operationen zur ↑Synchronisation durch einen ↑Kompilierer abprüfbar wird und so Programmierfehler vermieden werden können. Die zur ↑Bedingungssynchronisation erforderlichen Anweisungen erzeugt der Kompilierer, wie auch die Instruktionen, um den Monitor zeitweilig verlassen zu können, ohne diesen in der Zwischenzeit gesperrt zu halten. Mangels Sprachunterstützung ist dieses Konzept jedoch viel mehr zur Programmierkonvention entartet — das jedoch immer noch erleichtert, ein ↑nichtsequentielles Programm zu formulieren.

Befähigung Gabe von einem ↑Subjekt, eine bestimmte Operation auf ein ↑Objekt durchzuführen. Ein übertragbares, fälschungssicheres Merkmal, das einen ↑Prozess zu einer bestimmten ↑Aktion ermächtigt. In technischer Hinsicht ist diesem Merkmal ein bestimmter Wert zugeschrieben, der die ↑Referenz auf ein Objekt einschließlich ↑Zugriffsrecht repräsentiert. Dieser Wert ist in einer Variablen gespeichert, die das den betreffenden Prozess festlegende ↑Programm definiert: die Gabe ist damit ein physischer Bestandteil des Prozesses, sie ist mit dem Subjekt verknüpft und gegebenenfalls im jeweiligen ↑Prozessadressraum daselbst gespeichert (im Gegensatz zur ↑ACL). Ein Prozess kann über mehrere solcher Gaben für dasselbe Objekt oder verschiedene Objekte verfügen. Mit jeder dieser Gabe ist der Prozess sodann autorisiert, die damit jeweils verbundene Aktion stattfinden zu lassen. Bei Auslösung der Aktion

bezogen auf ein bestimmtes Objekt muss der Prozess seine Gabe dazu explizit machen, er trägt den Schlüssel (*key*) für die gewünschte Aktion bei sich. Ist die Gabe unzureichend, tritt eine ↑Ausnahmesituation ein.
Anders als bei einer ↑Zugriffskontrollliste ist der Widerruf (*revocation*) eines Zugriffsrechts schwierig, da der Inhalt der Schlüsselvariablen zu invalidieren ist: die ↑Adresse dieser Variablen gehört für gewöhnlich einem fremden ↑Adressbereich an (d.h., die Variable liegt in einem anderen ↑Adressraum isoliert von der widerrufenden Autorität) und ist gegebenenfalls sogar unbekannt. Noch komplizierter erweist sich der Widerruf aller Zugriffsrechte eines Prozesses auf ein gegebenes Objekt. Wird die ↑Adresse einer Schlüsselvariablen nicht im ↑Betriebssystem verbucht, ist in sämtlichen Prozessen eine ↑Ausnahme zu erheben (↑*broadcast*), deren jeweilige Behandlung die Invalidierung des Inhalts der Schlüsselvariablen zur Folge haben muss. Der Weg über die ↑Ausnahmebehandlung ist ebenfalls notwendig, sollte nur das Zugriffsrecht eines einzelnen Prozesses widerrufen werden müssen.

Befehlssatz Menge der Instruktionen, die ein ↑Prozessor ausführen kann. Jede einzelne Instruktion wird auch als ↑Maschinenbefehl bezeichnet.

Befehlszähler Register der ↑CPU, das ihrer Befehlsfolgesteuerung (*instruction sequencer*) die gegenwärtige Position im ablaufenden ↑Programm anzeigt; auch ↑PC genannt. Diese Positionangabe ist eine ↑Adresse im ↑Arbeitsspeicher, an der der ↑Abruf- und Ausführungszyklus der CPU einen ↑Maschinenbefehl erwartet und die pro Zyklus implizit um die Befehlslänge (heute (2016): der Anzahl von Bytes, die der komplette Befehl umfasst) inkrementiert wird. Bei ↑CISC hängt diese Länge vom jeweiligen Befehl ab, bei ↑RISC ist sie normalerweise für alle Befehle gleich.
Je nach CPU geschieht die Weiterschaltung des PC vor (*pre-increment*) oder nach (*post-increment*) dem Abrufen eines Maschinenbefehls: das heisst, im Falle einer ↑Ausnahmesituation bei der Befehlsausführung zeigt der PC entweder auf den Befehl, der die ↑Ausnahme hervorgerufen hat (*pre*: ab Motorola 68020), oder auf seinen Nachfolger (*post*: Motorola 68000/10, IA-32) im Befehlsstrom. Letzterer Fall macht eine ↑Ausnahmebehandlung, die zur Wiederaufnahme der Befehlsausführung führt, gegebenenfalls unmöglich, da der dann im ↑Unterbrechungszyklus gesicherte Inhalt des PC den Nachfolger des gescheiterten Befehls adressiert und eine Rückrechnung der Adresse nur möglich wäre, wenn alle Befehle der CPU gleich lang sind — was für CISC (IA-32) im Allgemeinen nicht zutrifft und auch bei RISC nicht sein muss. Abhilfe schafft hier eine Präventivmaßnahme der CPU, in der sie nämlich den jeweiligen Inhalt des PC vor Weiterschaltung vermerkt und in Abhängigkeit von der Ausnahme den vermerkten oder den aktuellen Wert im Unterbrechungszyklus sichert, sie also nach der Art der Ausnahme unterscheidet (*fault-class exceptions*, Intel 64 und IA-32).
Zusätzlich zur impliziten Veränderung durch ein Inkrement, kann der PC auch explizit durch spezielle Maschinenbefehle einen Wert erhalten. Bei einer CPU, die den PC zum ↑Programmiermodell zugehörig definiert (↑PDP 11 Register R7, ARM7 Register R15), kann die Veränderung sogar durch normale Schreibbefehle geschehen. In beiden Fällen erfolgen damit Sprünge im Programmablauf, womit Fallunterscheidungen, Schleifen, Aufrufe von einem ↑Unterprogramm (ebenso ↑Unterbrechungshandhaber) und Rückkehr daraus erst möglich werden.

Befehlszyklus (en.) ↑*instruction cycle*. Hauptschleife von einem ↑Interpreter: (1) den nächsten Befehl aus dem ↑Programm lesen, (2) ihn dekodieren, gegebenenfalls in Einzeloperationen zerlegen, alle benötigten Operanden entsprechend der jeweils spezifizierten ↑Adressierungsart laden und (3) die Operation/en durchführen. Schließlich (4) die Abfrage, ob während dieser Befehlsausführung eine ↑Ausnahme erhoben wurde und gegebenenfalls den ↑Unterbrechungszyklus einleiten. Führt eine ↑Aktion herbei.

Befestigungspunkt Stelle im ↑Namensraum von einem ↑Dateisystem zum ↑Einhängen eines weiteren Dateisystems. Typischerweise der ↑Name von einem ↑Verzeichnis, an dessen Stelle sodann das ↑Wurzelverzeichnis des eingehängten Dateisystems tritt.

Benutzerbereich Bezeichnung für den ↑Adressbereich, der einem ↑Maschinenprogramm vom ↑Betriebssystem zugestanden wird. Bezugsrahmen bildet ein bestimmter ↑logischer Adressraum oder ↑virtueller Adressraum, dessen Modell im Betriebssystem verankert ist. In diesem Gebiet handelt die ↑CPU für das Maschinenprogramm (↑*user mode*), darüberhinaus verfügt ein ↑Prozess darin nur über eingeschränkte Rechte (↑*unprivileged mode*).

Benutzerebene (en.) ↑*user level*. Bezeichnung für die Stufe, sowohl im Sinne von Privilegien (↑*unprivileged mode*) als auch in Bezug den Kontext (↑Maschinenprogramm), auf der ein ↑Prozess gegenwärtig stattfindet.

Benutzerkennung Zeichen zur eindeutigen Identifizierung derjenigen ↑Entität, die das ↑Rechensystem benutzen darf oder benutzt; für gewöhnlich eine Nummer. Die Kennung wird bei der ↑Anmeldung initial zugeordnet und kann gegebenenfalls im weiteren Verlauf von einem ↑Prozess für sich selbst geändert werden (`setuid(2)`). Der mögliche ↑Kennungswechsel ist hochgradig abhängig vom ↑Betriebssystem.

Benutzerprogramm Synonym zu ↑Anwendungsprogramm.

Benutzerzeit (en.) ↑*user time*. Bezeichnung für die Zeitspanne, die ein ↑Prozess im ↑Maschinenprogramm (↑*user mode*) verbracht hat. Ein über die gesamte Lebenszeit eines Prozesses akkumulierter Wert. Gegensatz zur ↑Systemzeit.

Benutzthierarchie Synonym zu ↑funktionale Hierarchie.

Bereitliste Aufstellung der von einem ↑Prozessor zu einem bestimmten Zeitpunkt zu leistenden Arbeit, wobei eine einzelne Arbeitseinheit durch einen ↑Prozess zu verrichten ist. Jeder der Prozesse auf dieser Liste steht bereit (↑Prozesszustand) zur ↑Einlastung von der ↑CPU beziehungsweise einem ↑Rechenkern sind.
Eine vom ↑Planer verwaltete ↑Warteschlange, die im Falle mitlaufender (*on-line*) ↑Ablaufplanung von ihm auch fortgeschrieben wird. Ob diese Aufstellung als statische (Tabelle) oder dynamische (einfach/doppelt verkettete Liste) Datenstruktur oder in Kombination von beiden (Vorrang- oder Prioritätenliste) implementiert ist, hängt ganz vom ↑Einplanungsalgorithmus ab. Konkretes Listenelement in all diesen Varianten ist der ↑Prozesskontrollblock, der direkt (ein Verkettungsglied enthaltend) oder indirekt (referenziert durch ein Verkettungsglied) in die Liste aufgenommen wird.

Bereitzeit (en.) ↑*release time*. Bezeichnung für eine ↑Prozessgröße, die den Zeitpunkt festlegt, zu dem die Durchführung einer ↑Aufgabe frühestens beginnen kann.

best fit (dt.) beste Passung: ↑Platzierungsalgorithmus.

Betriebsart Art und Weise der durch ein ↑Betriebssystem ermöglichten Betriebsführung in einem ↑Rechensystem. Die verschiedenen Ansätze unterscheiden unter anderem:

- die Anzahl der Teilnehmer am Rechnerbetrieb (↑*single-user mode*, ↑*multi-user mode*),
- die Anzahl der ↑Programme, die ein Teilnehmer gleichzeitig in Betrieb haben kann (↑*uniprogramming*, ↑*multiprogramming*),
- die Anzahl der ↑Prozesse, die ein Programm zugleich zulässt (uni-, ↑*multithreading*),
- die Art der Verschränkung der Prozesse, nämlich ob sie pseudo- oder echt parallel stattfinden (↑*parallel processing*),
- Zeitgarantien, die diesen Prozessen eingeräumt werden (keine, ↑*real-time mode*),
- den Grad der Ansprechempfindlichkeit von Prozessen, abgestuft für eine interaktionslose oder interaktive Bedienung und Nutzung des Systems (↑*batch mode*, ↑*conversational mode*) oder
- Techniken zur Optimierung von Betriebsabläufen (↑*remote operation*, ↑*overlapped I/O*, ↑*single-stream batch monitor*, ↑*multi-stream batch monitor*).

Jede dieser Optionen, einzeln für sich genommen oder in geeigneten Kombinationen, begründet eine eigene Rechnerbetriebsart, für die das Betriebssystem jeweils bestimmte Systemfunktionen bereitstellen muss.

Betriebslast (en.) ↑*overhead*. ↑Gemeinkosten durch überschüssigen oder indirekten Bedarf beispielsweise an Rechenzeit, Speicher, Bandbreite oder Energie, um eine bestimmte Aufgabe zu erledigen.

Betriebsmittel ↑Ressource, die ein ↑Prozess zur Vollbringung der ihm gemäß ↑Programm auferlegten Aufgabe benötigt. Ein Mittel, das in Hardware (↑Prozessor, ↑Speicher, ↑Peripherie) oder Software (Bestände von ↑Text oder ↑Daten) vorliegen kann, begrenzt verfügbar und wiederverwendbar ist oder unbegrenzt zur Verfügung stehen kann und dann aber (in einer Erzeuger-Verbraucher-Beziehung) als konsumierbar gilt.

Betriebsmittelkontrolle Überprüfung, der ein ↑Prozess bei Zuteilung, Nutzung und Rückgabe von einem ↑Betriebsmittel durch das ↑Betriebssystem unterzogen wird; kontinuierliche Überwachung durch eine dedizierte Steuerung im Betriebssystem, der Prozess und Betriebsmittel unterstehen (↑*resource management*).

Betriebsmittelverwaltung ↑Systemfunktion zur ↑Betriebsmittelkontrolle — mehr aber dazu in SP2.

Betriebsmodus ↑Arbeitsmodus, in dem das ↑Rechensystem betrieben wird. Normalerweise wird ein ↑Maschinenprogramm direkt durch die ↑CPU ausgeführt (↑*unprivileged mode*). Demgegenüber kommt ein ↑Betriebssystem nur außer der Reihe zur Ausführung (↑*privileged mode*), als ↑Ausnahme (↑*trap*, ↑*interrupt*): zwingend zur Inbetriebnahme des Rechensystems (↑*power-on reset*) und als Option während des Rechenbetriebs, nämlich wenn ↑Adressraumisolation zur Wahrung der ↑Integrität des Betriebssystems vorgeschrieben ist. So erfolgt der ↑Moduswechsel immer auch nur ausnahmebedingt: vom unprivilegierten in den privilegierten Modus und, nach erfolgter ↑Ausnahmebehandlung, wieder zurück. Dies schließt insbesondere jede Form von ↑Systemaufruf mit ein. Gleiches gilt für die Inbetriebnahme, bei der für den Urwechsel „zurück“ in den unprivilegierten Modus der Zustand einer vermeintlich vor dem Einschalten („Urknall“) ausnahmsweise unterbrochenen und jetzt wieder aufzunehmenden Ausführung eines Maschinenprogramms aufgesetzt wird.

Betriebsseite Element (↑Seite) einer ↑Arbeitsmenge.

Betriebssicherheit Zustand des Geschütztseins der Umgebung von einem ↑Prozess vor Gefahr oder Schaden durch ihn selbst (in Anlehnung an den Duden). Dabei ist mit Umgebung ein bestimmter Kreis von Prozessen gemeint, die intern (↑Programmablauf) oder extern (physikalisch-technischer Prozess) von dem jeweils als Bezugssystem betrachteten ↑Rechensystem stattfinden. Um diesen Zustand zu gewährleisten, sorgt ein ↑Betriebssystem für die Unwirksamkeit jeder ↑Aktion, die zur Verletzung der ↑Schutzdomäne anderer Prozesse führen kann. Grundsätzlich bedeutet dies, unautorisiertes Verlassen der eigenen Schutzdomäne eines Prozesses zu erkennen und abzufangen (↑Ausnahmesituation). Im Falle von ↑Echtzeitbetrieb ist darüber hinaus noch dafür Sorge zu tragen, dass abnormales Zeitverhalten eines beliebigen Prozesses die ↑Echtzeitbedingung eines strikt zeitabhängigen Prozesses nicht verletzt.
Nicht selten ist der Nachweis zu erbringen, dass ein solcher Zustand niemals durch Prozesse, die innerhalb des Rechensystems stattfinden, aufgehoben werden kann. Dazu finden formale Methoden Verwendung, die die formale Korrektheit der Software prüfen und gegebenenfalls bestätigen. Für gewöhnlich durchläuft die Software dazu eine aufwändige Zertifizierung, mit der die Einhaltung spezifizierter Anforderungen durch eine autorisierte und unabhängige Zertifizierungsstelle (z.B. der TÜV) nachgewiesen wird.

Betriebssystem Bezeichnung für ein ↑Programm oder eine Menge von Programmen, die Programme oder Anwendungen unterstützen und die Programmierung oder Bedienung von einem ↑Rechensystem erleichtern sollen, die Ausführung von Programmen überwachen und

steuern, das Rechensystem nach einer bestimmten Art und Weise für einen Anwendungsfall betreiben, eine ↑abstrakte Maschine (auch: ↑virtuelle Maschine) implementieren. Eine ↑Softwareschicht, die die ↑Betriebsmittel eines Rechensystems verwaltet, die Betriebsmittelvergabe steuert und die Betriebsmittel nach gewissen Kriterien an mitbenutzende Rechenprozesse verteilt.

Betriebssystemkern Herzstück von einem ↑Betriebssystem. Definiert als wesentliche Funktion das ↑Operationsprinzip für eine ↑abstrakte Maschine, die durch ein Betriebssystem implementiert wird. Das Operationsprinzip bestimmt letztlich die Architektur des Betriebssystems und damit auch den Funktionsumfang seines Kerns. Darüberhinaus stellt die für den jeweiligen Anwendungsfall vom Betriebssystem zu unterstützende ↑Betriebsart gegebenenfalls weitere Anforderungen, die sich auch in zusätzlicher und vom Kern zu erbringender Funktionalität niederschlagen kann. So lässt sich ohne Angabe der Betriebsart der tatsächliche Funktionsumfang eines solchen Kerns ebenso wenig festlegen wie der eines Betriebssystems. Fordert die Betriebsart beispielsweise eine Form von ↑Simultanverarbeitung, wird der Kern wenigstens Funktionen zur ↑Prozesseinplanung, ↑Unterbrechungsbehandlung und ↑Synchronisierung umfassen. Läuft es zusätzlich noch auf ↑Mehrprogrammbetrieb hinaus, werden im Kern zumindest grundlegende Funktionen zum ↑Speicherschutz und zur ↑Ausnahmebehandlung sowie ein ↑Systemaufrufzuteiler hinzukommen.

Betriebssystemlatenz Bezeichnung für die von einem ↑Betriebssystem zur Durchführung einer bestimmten ↑Systemfunktion anfallenden ↑Latenzzeit. Je nach der ↑Betriebsart beziehungsweise dem ↑Operationsprinzip des Betriebssystems ist damit die Verzögerung, die ein ↑Handlungsstrang im ↑Maschinenprogramm erfahren kann, gegebenenfalls unbestimmt und nicht berechenbar. Ist die Stelle, an der eine solche Verzögerung wirkt, im Fall von einem ↑Systemaufruf im Maschinenprogramm noch bestimmbar, macht demgegenüber eine ↑Unterbrechung eine exakte Vorhersage unmöglich. Bestenfalls kann dann noch die ↑Zwischenankunftszeit von Unterbrechungen abgeschätzt werden, um Phasen der Unterbrechungs- und damit Verzögerungsfreiheit zu veranschlagen.

Bibliothek Raum mit Ablagen; ↑Programmbibliothek.

binärer Semaphor Bezeichnung für einen ↑Semaphor, bei dem die ↑Ablaufsteuerung eine *binäre Variable* benutzt. Entweder realisiert als ↑allgemeiner Semaphor mit Wertebereich $[0, 1]$ oder speziell ausgelegt als boolescher elementarer Datentyp (`bool`). Bei letzterer Variante sind die in den Operationen ↑P und ↑V erfolgenden Zustandsänderungen in Bezug auf die enthaltene *boolesche Variable* implizit jeweils eine ↑atomare Operation: *Zuweisung* von `false` (P) beziehungsweise `true` (V).
Die Ablaufsteuerung lässt einen ↑Prozess in P blockieren, wenn der Variablenwert `false` ist. Anderenfalls setzt P die (`true` enthaltene) Variable auf `false` und lässt den Prozess voranschreiten. Allein anhand des Wahrheitswerts ist dann nicht feststellbar, ob ein Prozess in P blockiert und durch V gegebenenfalls zu deblockieren ist. In dem Fall hätte V nur die Wahl, die Variable auf `true` zu setzen und den ↑Planer die ↑Prozesstabelle ablaufen zu lassen (damit höhere ↑Latenzzeit mit sich zu bringen), um gegebenfalls zu deblockierende Prozesse fest- und wieder bereitzustellen. Verwendet P jedoch eine semaphoreigene Datenstruktur als ↑Warteschlange, kann V darüber einfach prüfen, ob Prozesse zur Deblockierung anhängig sind. Ist die Warteschlange leer, setzt V den Variablenwert auf `true`. Anderenfalls entnimmt V der Warteschlange den nächsten Prozess (birgt damit dann aber auch Gefahr von ↑Interferenz mit dem Planer) und deblockiert ihn, ohne den Wahrheitswert zu verändern: dieser bleibt `false`, solange noch Prozesse am Semaphor warten. Typischer Anwendungsfall dieser Art von Semaphor ist ein ↑kritischer Abschnitt oder ↑Monitor beziehungsweise überall dort, wo ↑wechselseitiger Ausschluss erforderlich ist.

Bindefehler ↑Ausnahmesituation beim Zugriff auf ein ↑Text oder ↑Daten enthaltendes ↑Segment im ↑Arbeitsspeicher. Die von einem ↑Prozess beim Zugriff verwendete ↑symbolische Adresse des Segments ist gültig, jedoch im Moment des Zugriffs hat es noch keinen Platz im

Arbeitsspeicher dieses Prozesses. Die Bindung des Segmentnamens (↑Symbol) an eine Segmentadresse geschieht erst zur ↑Laufzeit von dem betreffenden ↑Maschinenprogramm, sie ist dynamisch. Das Betriebssystem enthält dazu einen ↑Binder, der die Auflösung des Namens in eine ↑Adresse und damit verbunden die Platzierung des Segments im Rahmen einer ↑Ausnahmebehandlung durchführt und den beim Zugriff abgefangenen Prozess danach weiter fortsetzt: ↑partielle Interpretation des Zugriffs.

Binden Vorgang, um eine ↑Bindung einzurichten. Ein ↑Binder liest ein ↑Objektmodul nach dem anderen ein, analysiert jedes darin verwendete ↑Symbol nach seinem Geltungsbereich (d.h., ob es lokal oder global sichtbar ist) und versucht jede gegebenenfalls bestehende ↑unaufgelöste Referenz zu löschen. Jedes Objektmodul, das ein global sichtbares (exportiertes) und in einem anderen Objektmodul benutztes (importiertes) Symbol enthält, fließt in das aufzustellende ↑Lademodul ein. Dabei wird entweder das gesamte Objektmodul oder nur die darin enthaltene und exportierte ↑Entität berücksichtigt, um das ↑Text- und ↑Datensegment für das am Ende gebildete ↑Maschinenprogramm jeweils sukzessive aufzubauen. Im ersten Fall kann damit das Lademodul mehr Entitäten als benötigt (d.h. referenziert) aufweisen und entsprechend größeren Platz im ↑Arbeitsspeicher beanspruchen. Die Module liegen als ↑Datei vor oder werden einer ↑Programmbibliothek entnommen.
Je nach ↑Bindezeit operiert ein Binder unterschiedlich. Für komplette Bindungen vor ↑Ladezeit eines Maschinenprogramms darf im Lademodul keine ↑Referenz mehr unaufgelöst sein, anderenfalls endet der Bindevorgang mit einer Fehlermeldung (*unresolved references*) und das betreffende ↑Programm gilt als nicht ablauffähig. Für die Erstellung einer Bindung zur Ladezeit kommt zusätzlich ein ↑bindender Lader ins Spiel, der die noch offenen Bestandteile des Maschinenprogramms einfügt und dabei direkt in den Arbeitsspeicher bringt. In dem Fall hat der zur Generierung des Lademoduls genutzte (statische) Binder die Bindung in Bezug auf eine unaufgelöst gebliebene Referenz allerdings gültig markiert, das heißt, die referenzierte Entität ist bekannt im ↑Rechensystem und darf auch zur Komplettierung des Maschinenprogramms hinzugezogen und nachgeladen werden. Sehr ähnlich wird auch für die Erstellung einer Bindung zur ↑Laufzeit des Maschinenprogramms verfahren: die zur Laufzeit bestehenden unaufgelösten Referenzen sind gültig, sie werden jedoch erst im Moment ihrer erstmaligen Benutzung (d.h., wenn ein ↑Prozess in eine ↑Bindungsfalle tappt) endgültig gebunden. Hier greift ein ↑dynamischer Binder ein und komplettiert die Bindung, indem er die angeforderte Entität nachlädt.

bindender Lader Bezeichnung für einen ↑Lader, der eine in einem ↑Lademodul gegebenenfalls noch enthaltene ↑unaufgelöste Referenz selbst auflöst. Dabei bildet jede dieser Referenzen eine ↑symbolische Adresse von ↑Text oder ↑Daten vorrätig in einem ↑Objektmodul in einer ↑Bibliothek. Die Bibliothek mit dem Objektmodul, das die gesuchte symbolische Adresse in der ↑Symboltabelle listet, wird zur Herstellung einer ↑Bindung herangezogen. Steht in keiner Bibliothek ein solches Objektmodul zur Verfügung, scheitert der Ladevorgang.

Binder ↑Dienstprogramm, das mehrere Objektmodule zu einem ↑Objektmodul oder ↑Lademodul zusammenfügt.

Bindezeit Zeitpunkt, zu dem ein ↑Programm gebunden wird, das heisst, für jedes von diesem Programm verwendete ↑Symbol die ↑Bindung mit einer ↑Adresse stattfindet. Die Bindung wird eingerichtet, wenn das mit dem Symbol bezeichnete Objekt (↑Modul, ↑Unterprogramm, ↑Exemplar eine Datentyps) vom ↑Binder gefunden und der Art (↑Text, ↑Daten, ↑BSS) entsprechend dem jeweiligen ↑Segment hinzugefügt wurde, wodurch das Segment an Größe zunimmt. Gemäß der (zur Segmentbasis) relativen Objektlage in dem Segment, sowie der absoluten Lage des Segments im vom ↑Betriebssystem für ein ↑Maschinenprogramm vorgesehenen ↑Adressraum, kann schließlich das jeweilige Symbol in eine konkrete (reale, logische, virtuelle) ↑Ladeadresse aufgelöst werden: auf jede zur Basis 0 relativen Adresse eines solchen Objekts wird die Anfangsadresse des das Objekt enthaltenen Segments als ↑Relokationskonstante addiert. Abschließend wird an jeder Stelle im Maschinenprogramm, an der eine solche Objektreferenz verwendet wird, der korrigierte (verschobene) Adresswert eingetragen. Diese

Stellen sind in der ↑Relokationstabelle verzeichnet, die zusammen mit der ↑Symboltabelle beiden wichtigsten und pro ↑Objektmodul vom ↑Assemblierer für den Binder erstellten Datenstrukturen. Auch ein ↑bindender Lader richtet solche Bindungen ein, jedoch zur ↑Ladezeit eines Programms. Ein Objektmodul nach dem anderen wird nach den zu aufzulösenden Symbolen abgesucht, wobei diese Module eben erst von einem ↑Übersetzer erzeugt wurden oder einer ↑Bibliothek entnommen werden.

Bindung Beziehung zwischen einem ↑Symbol und einer ↑Adresse, repräsentiert durch einen Eintrag in der ↑Symboltabelle.

Bindungsfalle Bezeichnung für eine spezielle ↑Abfangung im ↑Betriebssystem, um bei Bedarf ↑dynamisches Binden durchzusetzen, sobald ein ↑Prozess nämlich eine für ihn zwar gültige jedoch noch ↑unaufgelöste Referenz benutzen sollte (↑*link trap*).

BKL Abkürzung für (en.) *big kernel lock*. Die Bezeichnung einer in ↑Linux anfänglich benutzten ↑Umlaufsperre, durch die ↑wechselseitiger Ausschluss für das gesamte ↑Betriebssystem sichergestellt wurde. In aktuellen Versionen ist diese Sperre nicht mehr vorhanden.

Blattprozedur ↑Unterprogramm, das kein weiteres Unterprogramm im selben ↑Adressraum aufruft, auch nicht sich selbst.

Block Abschnitt fester Größe im ↑Ablagespeicher und ↑Archivspeicher. Die Größe eines solchen Blocks hängt vom Bezugsrahmen ab, beispielsweise: 512 B, ↑Gerätetreiber (Platte: ↑Peripheriegerät); 1 KiB, ↑Betriebssystemkern (↑Linux: `vmstat(8)`); 4 KiB, ↑Dateisystem. Letzteres ist ein typischer Wert für eingangs genannte Arten von ↑Speicher.

blockierende Synchronisation Art der ↑Synchronisation, bei der ein ↑Prozess im Wettstreitfall (↑*contention*) darauf warten muss, bis er an der Reihe ist, eine bestimmte ↑Aktion oder ↑Aktionsfolge durchzuführen. Diese Aktion/Aktionsfolge ist als ↑kritischer Abschnitt beschrieben. Auch als ↑pessimistische Nebenläufigkeitssteuerung bezeichnet.

Blocknummer Zahl, die einen ↑Block auf einem ↑Datenträger eindeutig kennzeichnet.

boot block (dt.) ↑Startblock.

boot sector Synonym zu ↑*boot block*.

bootstrap (dt.) ↑Urlader, urladen.

bootstrap loader (dt.) ↑Urladeprogramm.

bootstrapping (dt.) ↑Ureingabe.

bounds register (dt.) ↑Grenzregister.

broadcast (dt.) ↑Rundruf.

BSD Mehrplatz-/Mehrbenutzerbetriebssystem. Abkürzung für (en.) *Berkeley Software Distribution*, Variante von ↑UNIX, erste Installation 1977 (↑PDP 11).

BSS Abkürzung für (en.) *block started by symbol*, (dt.) Block eingeleitet durch ein ↑Symbol. Der Begriff steht für ausgespartem Raum in einem in ↑Assembliersprache formulierten ↑Programm. Der jeweilige Raum wird durch eine Marke (Symbol) gekennzeichnet, er erhält einen Namen. Seine Größe erhöht den ↑Adresszähler bei der ↑Assemblierung entsprechend, ohne jedoch einen Inhalt an seiner Stelle zu hinterlassen: der Raum bleibt leer. Erst beim ↑Laden wird das diesen Raum beanspruchende ↑Datensegment im ↑Arbeitsspeicher initialisiert, indem jedes ↑Speicherwort in diesem Segment den Wert 0 erhält.

Buchführung Aufzeichnung der von einem ↑Prozess oder einer ↑Prozessinkarnation genutzten ↑Betriebsmittel. Je nach Art des Betriebsmittels werden dabei unterschiedliche Werte erfasst. Für ein ↑wiederverwendbares Betriebsmittel ist etwa die Nutzungsdauer ein wichtiger Wert, wohingegen für ein ↑konsumierbares Betriebsmittel eher die Anzahl relevant ist. Die über die Zeit akkumulierten Werte werden entweder direkt im ↑Prozesskontrollblock gehalten oder indirekt darüber in eigenen Datenstrukturen verwaltet. Die aufgezeichneten ↑Daten dienen einerseits bestimmten strategischen Verfahren in einem ↑Betriebssystem, um einen effizienten Rechnerbetrieb zu gewährleisten, und andererseits aber auch kommerziellen Zwecken, wenn nämlich die Inanspruchnahme allgemein von dem ↑Rechensystem nur gegen Entgelt erfolgt.

build management (dt.) Erstellungsprozess. Vorgang bei der Softwareentwicklung, durch Konfigurierung, Generierung, Übersetzung (↑Kompilation, ↑Assemblierung) und ↑Binden ein ↑Programm automatisch zu erzeugen. Eine der einfachsten Varianten davon ist `make(1)`.

built-in command (dt.) ↑integrierter Befehl.

burst mode (dt.) ↑Stoßbetrieb.

bus error (dt.) ↑Busfehler.

Busfehler Bezeichnung einer ↑Ausnahme, die durch eine für den ↑Adressbus ungültige ↑Adresse hervorgerufen wird. Zu dieser Adresse gibt es keinen Adressaten, das heißt, weder ein ↑Speicherwort noch ein ↑Peripheriegerät kann dadurch von der ↑CPU angesprochen werden.

busy waiting (dt.) ↑geschäftiges Warten.

Byte Maßeinheit für die Anzahl der Bits zur Kodierung eines einzelnen Schriftzeichens in einem ↑Rechensystem. Gebräuchliche Längen waren/sind 5, 6, 7 (ASCII), 8 (EBCDIC), 9 oder 12 Bits. Darüberhinaus gab es ↑Rechner mit $\approx 5,644$ (Radix-50) oder 1–36 Bits (einstellbar, DEC PDP-10) pro Schriftzeichen. Wenn nicht anders angegeben, dann werden heute (2016) typischerweise 8 Bits zur Zeichenkodierung angenommen, auch als ↑Oktett bezeichnet.

Bytereihenfolge (en.) ↑*endianness*. Reihung der einzelnen ↑Bytes in einem ↑Speicherwort, das als Einheit aus mehr als einem Byte zusammengesetzt ist. So kann entweder das höchstwertigste (*big endian*, Motorola-Format) oder das niederwertigste (*little endian*, Intel-Format) Byte in diesem Wort zuerst gespeichert sein und somit an der kleinsten ↑Adresse in diesem Wort liegen, gefolgt von den jeweils nächsten Bytes desselben Worts. Bei einem 32 Bits breitem Speicherwort von zwei 16-Bit Halbworten, je zwei Bytes, kann die Reihung sowohl in Bezug auf die Halbwörter als auch den Bytes darin verschieden sein (*middle endian*, PDP-Format). Einige ↑Prozessoren erlauben die Einstellung der Reihung (*bi-endian*, z.B. ↑PowerPC).
Zur Verdeutlichung sei angenommen, einer Variablen `value` vom Datentyp `long` (↑C) wird die Dezimalzahl 3 735 943 886 zugewiesen, hexadezimal `DEADFACE` (↑*hexspeak*). Je nach Art der Reihung kann das 32-Bit Speicherwort, das als Platzhalter für die Variable `value` verwendet wird, damit folgende Belegungsmuster aufweisen:

Adresse	*endianness*		
	big	*little*	*middle*
BAFF0000	DE	CE	AD
BAFF0001	AD	FA	DE
BAFF0002	FA	AD	CE
BAFF0003	CE	DE	FA

Wenn nun dieses beispielsweise von einem „Großender“ beschriebene Speicherwort von ihm selbst und darüber hinaus auch von einem „Kleinender“ und einem „Mittelender“ gelesen wird, deuten die betreffenden Prozessoren den Inhalt des Speicherworts unterschiedlich: nämlich als den Zahlenwert 3 735 943 886 (*big*, richtig), 3 472 535 006 (*little*, falsch) und 2 917 060 346 (*middle*, falsch). Sollten also auf gemeinsame ↑Daten zugreifende ↑Prozesse

auf Prozessoren mit solch unterschiedlichen Sichten stattfinden, müssen die betreffenden ↑Programme selbst sicherstellen, dass derartige Missdeutungen ausgeschlossen sind.

C Programmiersprache: imperativ, prozedural (1969).

C++ Programmiersprache: imperativ, objektorientiert (1979). Erweiterung von ↑C.

cache (dt.) ↑Zwischenspeicher.

cache coherence (dt.) Zwischenspeicherkohärenz: allg. ↑Speicherkohärenz.

cache line (dt.) ↑Zwischenspeicherzeile.

cache miss (dt.) ↑Zwischenspeicherfehlzugriff.

callee-saved register Synonym zu (en.) ↑*non-volatile register*.

caller-saved register Synonym zu (en.) ↑*volatile register*.

capability (dt.) ↑Befähigung.

card punch (dt.) ↑Kartenstanzer.

card reader (dt.) ↑Kartenleser.

cascaded interrupt (dt.) ↑kaskadierte Unterbrechung.

CCITT Abkürzung für (fr.) *Comité Consultatif International Téléphonique et Télégraphique*.

CD Akürzung für (en.) *compact disc*, (dt.) Kompaktscheibe.

cdecl ↑Aufrufkonvention von ↑C.

channel (dt.) ↑Kanal.

channel program (dt.) ↑Kanalprogramm.

circular first fit Synonym zu ↑*next fit*.

CISC Abkürzung für (en.) *complex instruction set computer*, (dt.) ↑Rechner mit vielschichtigem (komplexen) ↑Befehlssatz. Der ↑Prozessor in solch einem Rechner kann einen einzelnen ↑Maschinenbefehl als mehrere Einzeloperationen (z.B. Operanden laden, Berechnung durchführen, Ergebnis speichern) auf niedriger Ebene direkt ausführen oder ist zu mehrstufigen Operationen oder ↑Adressierungsarten innerhalb eines einzelnen Maschinenbefehls in der Lage. Für gewöhnlich variieren die Befehlslängen, einerseits wegen verschieden langer Befehlskodes und andererseits wegen der für einen Befehl erlaubten Adressierungsarten. Entsprechend variiert die Anzahl der bei Ausführung benötigten Taktzyklen von Befehl zu Befehl. Anders als beim ↑RISC ist der Grundgedanke hinter der ↑Rechnerarchitektur, die ↑semantische Lücke auch durch Maßnahmen in Hardware zu verkleinern und dafür in Bezug auf ↑Performanz eher einen Kompromiss einzugehen. Typisches Beispiel ist ↑Intel 64.

CLI Abkürzung für (en.) *command line interpreter*. Ein besonderer ↑Kommandointerpreter.

command line (dt.) ↑Kommandozeile.

compiler (dt.) Kompilierer.

completion time (dt.) ↑Endzeit.

computer (dt.) ↑Rechner.

computer installation Synonym zu ↑*computer system*.

computer system (dt.) ↑Rechenanlage.

concurrency control (dt.) ↑Nebenläufigkeitssteuerung.

condition variable (dt.) ↑Bedingungsvariable.

condition-code register (dt.) ↑Statusregister.

constant folding (dt.) ↑Konstantenfaltung.

constant propagation (dt.) ↑Konstantenpropagation.

contention (dt.) Wettbewerb, Wettstreit, ↑Konkurrenzsituation.

continuation (dt.) ↑Fortsetzung.

control loop (dt.) Regelschleife, ↑Regelkreis.

conversational mode (dt.) ↑Dialogbetrieb.

cooperative scheduling (dt.) ↑kooperative Planung.

coroutine (dt.) ↑Koroutine.

covered channel (dt.) ↑verdeckter Kanal.

CPU Abkürzung für (en.) *central processing unit*, (dt.) ↑Zentraleinheit.

CPU bound (dt.) ↑CPU gebunden. Bezeichnung für eine rechenintensive ↑Aufgabe, einen nicht interaktiven ↑Prozess. Gegenteil von ↑*I/O bound*.

CPU *burst* (dt.) ↑Rechenstoß.

CPU protection (dt.) ↑CPU Schutz.

CPU Schutz (en.) ↑*CPU protection*, Vorkehrung in einem ↑Betriebssystem, durch die die Monopolisierung der ↑CPU durch einen einzelnen ↑Prozess verhindert wird. Dazu erhält jeder Prozess die CPU nur für maximal eine ↑Zeitscheibe zugeteilt. Durch Ablauf der Zeitscheibe kommt eine ↑präemptive Planung zur Wirkung, die im weiteren Verlauf dem dadurch jeweils unterbrochenen Prozess die CPU entziehen kann.

CR Abkürzung für (en.) *carriage return*, (dt.) ↑Wagenrücklauf. Steuerzeichen, kodiert als $\texttt{0D}_{16}$ in ↑ASCII, ↑EBCDIC und ↑*Unicode*.

critical section (dt.) ↑kritischer Abschnitt.

cross-cutting concern (dt.) ↑Querschnittsbelang.

CSIM Abkürzung für (en.) *complete software interpreter machine*, (dt.) vollständig in Software realisierter ↑Interpreter.

CTSS Mehrplatz-/Mehrbenutzerbetriebssystem. Abkürzung für „*Compatible Time-Sharing System*“. Führte das ↑Zeitteilverfahren ein, um einem ↑Prozess die lllusion vom eigenen ↑Prozessor zu verschaffen (↑partielle Virtualisierung) und so mehrere Prozesse zugleich stattfinden lassen zu können (↑Simultanverarbeitung). Das System war kompatibel mit ↑FMS. Erste Installation im Jahr 1961 (modifizierte IBM 7094).

current working directory (dt.) ↑Arbeitsverzeichnis.

cursor (dt.) ↑Schreibmarke.

cycle stealing (dt.) ↑Taktentzug.

cycle time (dt.) ↑Durchlaufzeit.

Dämon Bezeichnung für einen ↑Prozess, der anhaltend im Hintergrund stattfindet und dabei eine bestimmte ↑Systemfunktion erfüllt (z.B. `cron(8)` oder `lpd(8)`). Der Prozess findet in einem dedizierten ↑Maschinenprogramm, also oberhalb eines Betriebssystems, oder in dem Betriebssystem selbst statt.

dark silicon (dt.) dunkles Silizium. Menge von Schaltkreisen einer integrierten Schaltung, die aufgrund des vorgegebenen Grenzwerts für die thermische Verlustleistung (*thermal design power*, TDP) elektrischer Bauteile nicht mit der nominellen Betriebsspannung (d.h., oberen Nennspannung) versorgt sein darf: Bereiche von Transistoren in einem ↑Prozessor, die durch Überhitzungsschutz abzuschalten oder abgeschaltet sind. Für eine als ↑Mehrkernprozessor ausgelegte ↑CPU bedeutet dies beispielsweise, dass nicht jeder ↑Rechenkern durchgängig in Betrieb ist. Die jeweils abgeschalteten Kerne bilden eine Kühlfläche für angeschaltete Kerne, deren Abwärme dann über diese Fläche abgeführt werden kann.

Darwin Mehrplatz-/Mehrbenutzerbetriebssystem, von ↑UNIX abstammend. Erste Installation März 1999 (PowerPC), Basis für ↑macOS. Programmiert in ↑Objective-C, ↑C++ und ↑C.

data (dt.) ↑Daten.

data processing system (dt.) Datenverarbeitungssystem, ↑Rechensystem.

Datei Kunstwort, von Datenkartei; Bestand von sachlich zusammenhängenden ↑Daten, der dauerhaft im ↑Speicher vorrätig sein kann. Je nach Art der Daten, grob unterschieden in *ausführbare* und *nichtausführbare Datei*. Erstere speichert ein ↑Programm für einen ↑Interpreter. Letztere enthält Daten, die erst durch einen ↑Prozess der Verarbeitung unterzogen werden. Beide Arten sind von einer auch als „echte Datei“ bezeichneten Klasse, im Gegensatz zur ↑Pseudodatei, die nicht der ↑EDV im eigentlichen Sinne dient.

Dateibaum Struktur, die der ↑Namensraum in einem ↑Dateisystem bildet (↑hierarchischer Namensraum). Die Besonderheit dabei ist die Darstellung als mit der Wurzel (↑*root directory*) nach oben an einer Befestigung (↑*mounting point*) eingehängter Baum.

Dateiname ↑Name, der eine ↑Datei in einem ↑Dateisystem identifziert. Der Name ist für gewöhnlich eindeutig immer nur in Bezug auf den für ihn gültigen ↑Namenskontext. Dieselbe Datei kann (i) innerhalb desselben Namenskontextes über verschiedene oder (ii) in verschiedenen Namenskontexten über denselben Namen identifiziert werden (↑*hard link*).

Dateisystem Prinzip, nach dem Informationen im ↑Ablagespeicher und ↑Archivspeicher gegliedert und geordnet sind; Einheit von (dauerhaft) zu speichernde Informationen und ↑Datenträger. Das grundlegende, interne Strukturelement für einen solchen Datenträger ist der ↑Block, aber das nach außen sichtbare ist die ↑Datei, in der ↑Nutzdaten zu beliebigen Zwecken gespeichert sind. Um diese Daten jedoch auf dem Datenträger so zu organisieren, dass sie in effizienter Weise wieder abrufbar und gegebenenfalls auch aktualisierbar sind, bedarf es ↑Verwaltungsdaten. Zu den Verwaltungsdaten zählt eine Zusammenstellung des auf dem Datenträger selbst genutzten Gebiets (↑*partition*) von Blöcken, der in diesem Gebiet (i) freien Blöcke für Nutz- aber auch weitere Verwaltungsdaten und (ii) reservierten Blöcke für Dateidatenstrukturen (↑*inode table*) und Listen freier Einzelstücke dieser Strukturen wie auch fehlerhafter Blöcke (↑*bad block*). Derartige Informationen, zumeist indirekt über Verweise (d.h., Anfangsblock und Blockanzahl in dem Gebiet) zum Ausdruck gebracht, sind Attribute einer zentralen Datenstruktur (↑*superblock*), die redundant über den Datenträger abgelegt ist, um im Fehlerfall die Nutz- und Verwaltungsdaten weiterhin verfügbar zu haben.

Daten Zahlenwerte oder (zweckdienlich) kodierte Informationen, die auf Beobachtungen, Messungen, Erhebungen, Angaben oder Berechnungen beruhen und elektronisch (digital) in einem ↑Speicher abgelegt werden können.

Datenbus Verbindungssystem in einem ↑Rechner, über das ↑Daten übermittelt werden.

Datensegment Bezeichnung für ein ↑Segment, das die ↑Daten von einem ↑Programm speichert.

Datenträger Bezeichnung für ein ↑Speichermedium.

Datentyp Beschreibung einer bestimmten Menge von ↑Daten derselben Struktur und der auf diesen Daten definierten Operationen. Jedes einzelne Datum als Element dieser Menge ist Beispiel (↑*instance*), ↑Exemplar, einer konkreten Ausprägung derselben Art, desselben Typs. Vor oder zur ↑Laufzeit wird sichergestellt, dass die für das Datum beabsichtigte Operation gültig, typkonform ist. Im Fehlerfall, scheitert die ↑Kompilation von dem betreffenden ↑Programm (vor Laufzeit) oder es tritt eine ↑Ausnahmesituation ein (zur Laufzeit).

dauerhaftes Betriebsmittel Synonym zu ↑wiederverwendbares Betriebsmittel.

deadline (dt.) ↑Frist, ↑Termin.

Defektblockverwaltung Funktion im ↑Dateisystem oder in der Firmware einer ↑Plattensteuerung, mit der ein ↑defekter Block ausgeblendet und durch einen fehlerfreien Block ersetzt wird. Dazu werden in einem reservierten Bereich auf dem ↑Datenträger Ersatzblöcke vorgehalten. Eine ebenfalls auf diesem Medium liegende Zurodnungstabelle bildet die defekten auf die fehlerfreien Blöcke ab. Diese Tabelle ist entsprechend zu verwalten.

defekter Block Bezeichnung für einen ↑Block, der unbrauchbar (geworden) ist. Ursache können im ↑Datenträger manifestierte Fertigungsfehler sein, aber auch Fehler die durch Abnutzung oder Alterung auftreten. Ein solcher bei einer Ein-/Ausgabeoperation erkannter Block wird ausgeblendet und durch einen fehlerfreien Block ersetzt (↑*bad-block management*).

deferred procedure call (dt.) ↑zurückgestellter Prozeduraufruf.

Defragmentierung Verfahren zur Auflösung der ↑Fragmentierung. Die im ↑Hauptspeicher belegten Abschnitte werden geeignet verschoben, um einen einzigen zusammenhängenden freien Abschnitt zu schaffen. Voraussetzung dafür ist ein ↑logischer Adressraum für jeden ↑Prozess, dessen im Hauptspeicher liegende Anteile von ↑Text und ↑Daten von der Verschiebung betroffen sind: die ↑reale Adresse eines jeden verschobenen Abschnitts ändert sich, nicht jedoch dessen ↑logische Adresse. Nach jedem Verschiebevorgang wird jeder ↑Seitendeskriptor oder ↑Segmentdeskriptor, der einen verschobenen Abschnitt referenzierte, mit der neuen realen Adresse programmiert. Am Ende des Vorgangs hat die ↑Freispeicherliste nur noch einen Eintrag.

demon (dt.) ↑Dämon.

descriptor privilege level (dt.) durch einen ↑Deskriptor definierte Privilegstufe. Gemeinhin die für ↑x86 (ab Modell ↑i286) gebräuchliche Bezeichnung für ein bestimmtes Vor- oder Sonderrecht, das den Zugriffsmodus eines ↑Subjekts auf ein bestimmtes ↑Objekt festlegt. Für x86 sind vier Stufen möglich, sehr ähnlich zur ↑VAX, wobei Stufe 0 mit den höchsten (↑*system level*: ↑*kernel mode*) und Stufe 3 mit den niedrigsten (↑*user level*) Privilegien verknüpft ist. Jede dieser Stufen ist gleichbedeutend mit einem ↑Schutzring.

Deskriptor Kenn- oder Schlüsselwort, durch das der Inhalt einer Information charakterisiert wird und das zur Bestimmung von ↑Text oder ↑Daten im ↑Speicher von einem ↑Rechensystem dient (in Anlehnung an den Duden).

desktop (dt.) Arbeitsoberfläche. Grundlage der „Schreibtischmetapher“, bei der die für einen ↑Rechner dargestellte Arbeitsfläche einem Schreibtisch nachempfunden ist und die hinterste Ebene bildet, über die dann einzelne Fenster als Bildschirmelemente dargestellt sind.

deterministic scheduling (dt.) ↑deterministische Planung.

deterministische Planung (en.) ↑*deterministic scheduling*. Modell der ↑Ablaufplanung, die eine kausale Vorbestimmung allen Geschehens oder Handelns der ↑Prozesse vornimmt. Die Prozessreihenfolge ist durch Vorbedingungen im Voraus eindeutig festgelegt: sie ist, anders als ↑probabilistische Planung, zufallsunabhängig.
Charakteristisches Merkmal ist die ↑vorlaufende Planung, die nämlich über das nötige ↑Vorwissen verfügt, um einen ↑Ablaufplan liefern zu können, der während des Betriebs zu jedem Zeitpunkt bestimmt, mit welchem Prozess im ↑Rechensystem weitergefahren wird. Anderenfalls lassen sich weder kausal vorbestimmte Prozessfolgen herleiten noch eindeutige Aussagen zum Zusammenspiel dieser Prozesse treffen. Das Vorwissen betrifft die Anzahl der einzuplanenden Prozesse, ihre jeweilige ↑Dringlichkeit, eventuelle Abhängigkeiten untereinander sowie Art und Anzahl der jeweils benötigten ↑Betriebsmittel. Für jeden Prozess ist zudem seine ↑Ankunftszeit, ↑Bearbeitungszeit und ↑Frist bekannt. Typisches Beispiel ist ↑RM.

device file (dt.) ↑Gerätedatei.

Dialogbetrieb (en.) ↑*conversational mode*. Bezeichnung für eine ↑Betriebsart, bei der der Austausch von Fragen und Antworten zwischen einem Menschen und dem ↑Rechensystem über eine ↑Dialogstation im Vordergrund steht. Die dabei stattfindende Mensch-Maschine-Interaktion wird durch einen ↑Dialogprozess geführt. Ziel für das ↑Betriebssystem ist es, die ↑Antwortzeit der jeweils gestellten Anfrage wie auch die ↑Durchlaufzeit der damit verbundenen Arbeitsaufträge zu minimieren. Gegebenenfalls ist auch ↑Termineinhaltung zu gewährleisten, wenn nämlich die Antwort zu einer Anfrage innerhalb einer bestimmten Frist vorliegen muss. Letzteres bedeutet, dass das Betriebssystem dann einer vorgegebenen ↑Echtzeitbedingung zu genügen hat. Eine gewisse ↑Vorhersagbarkeit des Systemverhaltens bei Ausführung der Anfragen ist zumindest wünschenswert, in bestimmten Anwendungsfällen sogar gefordert.

Dialogprozess ↑Programmablauf in einem ↑Rechensystem, der die wechselseitige Kommunikation mit einen typischerweise in Gestalt eines Menschen erscheinenden „externen Prozess“ durch Verwendung einer ↑Dialogstation führt. Kontrolliert wird der Programmablauf durch einen ↑Kommandointerpreter, der die Anfragen an das Rechensystem entgegennimmt und ausführt. Dabei kann die Bedienoberfläche textorientiert (↑*shell*), grafisch (↑*desktop*) oder in geeigneter Kombination beider Arten ausgelegt sein.

Dialogstation ↑Peripheriegerät zur wechselseitigen Kommunikation (Absetzen von Anfragen, Entgegennahme von Antworten) zwischen Mensch und ↑Rechensystem. Für gewöhnlich zwei Geräte vereint: einerseits die Rechnertastatur als Eingabegerät und andererseits der Rechnerbildschirm (↑*terminal*) oder -zeilendrucker (↑TTY) als Ausgabegerät. Je nach technischer Ausführung sind dafür im ↑Betriebssystem gegebenenfalls auch zwei ↑Gerätetreiber erforderlich und nicht nur einer.

Dienst Bündelung von Funktionen in einem ↑Rechensystem nach einer bestimmten fachlichen, thematischen Ausrichtung. Das „Funktionsbündel“ verfügt über eine klar definierte Schnittstelle, durch die ein ↑Auftrag an das Rechensystem abgegeben und das Ergebnis der Auftragsausführung entgegengenommen werden kann. Beispiele sind Informations-, Datenbank-, Buchungs-, Bezahl-, Abrechnungs-, Web-, Netzwerk- oder Systemdienste.

Dienstprogramm Bezeichnung von einem ↑Programm für systemnahe Aufgaben, meist zum ↑Betriebssystem gehörend. Solche Aufgaben bestehen beispielsweise darin, Objektmodule zu einem ↑Lademodul zusammenzubinden (`ld(1)`), Dateiverzeichnisse aufzulisten (`ls(1)`), Dateien zu kopieren (`cp(1)`), Druckaufträge „aufzuspulen“ (`lp(1)`), den Zustand von Prozessen darzustellen (`ps(1)`), Parameter der Netzwerkschnittstelle zu konfigurieren (`ifconfig (8)`) oder Statusdaten im Betriebssystem zu manipulieren (`sysctl (8)`).

directory (dt.) ↑Verzeichnis.

directory entry (dt.) ↑Verzeichniseintrag.

direkte Adresse Synonym zu ↑absolute Adresse.

Direktzugriffsspeicher Bezeichnung für einen ↑Speicher mit wahlfreiem/direktem Zugriff auf jedes ↑Speicherwort über eine jeweils eindeutig zugeordnete ↑Adresse.

dirty bit Synonym zu ↑*modified bit*.

disc controller (dt.) ↑Plattensteuerung.

disc memory (dt.) ↑Plattenspeicher.

dispatcher (dt.) ↑Umschalter, Zuteiler.

dispatching (dt.) ↑Einlastung.

dispatching latency (dt.) ↑Einlastungslatenz.

distributed shared memory (dt.) ↑verteilter gemeinsamer Speicher.

DLL Abkürzung für (en.) *dynamic link library*. Kurzbezeichnung für eine ↑dynamische Programmbibliothek mit Bezugspunkt ↑Windows.

DMA Abkürzung für (en.) *direct memory access*, (dt.) ↑Speicherdirektzugriff.

DMA *controller* (dt.) Steuereinheit für ↑DMA.

dot (dt.) ↑Name vom ↑Arbeitsverzeichnis (↑UNIX).

dot dot (dt.) ↑Name vom ↑Elterverzeichnis (↑UNIX).

DPC Abkürzung für (en.) ↑*deferred procedure call*.

DPL Abkürzung für (en.) ↑*descriptor privilege level*.

DRAM Abkürzung für (en.) *dynamic RAM*, (dt.) dynamischer ↑RAM.

Dringlichkeit Grad, in dem die Bearbeitung eines Auftrags drängt, in dem ein ↑Prozess den ↑Prozessor zugeteilt bekommen oder abgeschlossen werden muss.

drum address (dt.) ↑Trommeladresse.

drum machine (dt.) ↑Trommelmaschine.

drum memory (dt.) ↑Trommelspeicher.

DSM Abkürzung für (en.) ↑*distributed shared memory*.

Dualsystem Zahlensystem, das nur zwei Ziffern (0, 1) zur Darstellung aller Zahlen als Potenzen von 2 verwendet.

Durchlaufzeit (en.) ↑*cycle time*, ↑*through-put time*. Zeitspanne zwischen der Ankunft von einem ↑Prozess und seiner vollständigen Beendigung. Bezugspunkt dabei ist der ↑Planer, der nämlich bestimmt, wann ein Prozess das erste Mal auf die ↑Bereitliste kommt, wann dieser Prozess zur ↑Einlastung von dem ↑Prozessor ansteht, wie oft der Prozess einer ↑Verdrängung unterzogen wird und wann der Prozess die ↑Ablaufplanung verlässt.

Durchsatz Anzahl von Aufträgen, die ein ↑Rechensystem, ↑Betriebssystem oder ↑Prozess pro Zeiteinheit verarbeiten kann. Beispielsweise die Anzahl von Ein-/Ausgabeaufträgen, abgesetzt von einem einzelnen Prozess oder einer bestimmten Gruppe von Prozessen.

DVD Abkürzung für (en.) *digital versatile disc*, (dt.) Universalscheibe für digitale Daten.

dynamic binding (dt.) ↑dynamisches Binden.

dynamische Programmbibliothek Bezeichnung für eine ↑Programmbibliothek, aus der heraus Bestände von ↑Text oder ↑Daten mitlaufend mit dem betreffenden ↑Prozess in den ↑Arbeitsspeicher geladen werden, das heißt, die ↑Bindezeit von einem ↑Maschinenprogramm entspricht seiner ↑Ladezeit oder ↑Laufzeit. Zur Ladezeit kommt ein ↑bindender Lader ins Spiel, der automatisch die im ↑Lademodul noch als unaufgelöst geltenden Text- oder Datenreferenzen feststellt, die entsprechenden Bestände einer solchen Bibliothek entnimmt und sie mit dem Maschinenprogramm zusammen in den Arbeitsspeicher lädt, wenn sie nicht bereits geladen wurden. Zur Laufzeit setzt ein in dem (noch nicht komplett gebundenen) Maschinenprogramm stattfindender Prozess explizit einen ↑Systemaufruf ab, um die benötigte Programmbibliothek im Arbeitsspeicher zugänglich zu machen (↑UNIX: `dlopen(3)`; ↑Windows: `LoadLibrary`). Die dabei anfallende ↑Aktionsfolge bedeutet jedoch kein ↑dynamisches Binden, wo ein Fehlzugriff auf Text oder Daten den Auslöser bildet und die Einbindung einer solchen ↑Entität durch ↑partielle Interpretation des Zugriffs bewerkstelligt wird. Beide Fälle (Lade-/Laufzeit) führen für gewöhnlich zur Mitbenutzung einer schon im Hauptspeicher liegenden Bibliothek oder Teile davon durch mehrere Prozesse (↑*shared library*) und erfordern damit den Zugriff auf denselben ↑Speicherbereich (↑*shared memory*) durch den jeweiligen ↑Prozessadressraum.

dynamischer Binder Bezeichnung für einen ↑Binder, der als ↑Systemfunktion integriert in einem ↑Betriebssystem den Dienst vollbringt, eine ↑Bindung zu einem ↑Text- oder ↑Datensegment herzustellen (↑dynamisches Binden). Der Binder behandelt den in einem ↑Prozess aufgetretenen ↑Bindefehler und gliedert ein dem ↑Ablagespeicher entnommenes Text- oder Datensegment in den ↑Adressraum des Prozesses ein. Aktiviert wird der Binder bei Bedarf, im Rahmen der ↑Ausnahmebehandlung bei einem ↑Hauptspeicherfehlzugriff über eine noch als ↑symbolische Adresse ausgelegte und spätere ↑logische Adresse oder ↑virtuelle Adresse von dem betreffenden ↑Segment.

dynamischer Speicher Bezeichnung für einen ↑Speicher, dessen räumliches Fassungsvermögen (Kapazität) veränderlich ist; Gegenteil von ↑statischer Speicher. Typische Beispiele dafür sind ↑Stapelspeicher und ↑Haldenspeicher. Ersterer wird implizit (automatisch) mit jedem Aufruf von einem ↑Unterprogramm vergrößert und bei Rückkehr davon wieder verkleinert. Letzterer wird explizit in Anspruch genommen, in ↑C beispielsweise durch `malloc(3)` und `free(3)`. Beiden Speichern gemeinsam ist die Zugehörigkeit zu einem ↑Programm, wobei sie aber selbst erst durch einen ↑Prozess dieses Programms, also zur ↑Laufzeit, in Erscheinung treten und zur Wirkung kommen.

dynamisches Binden ↑Bindung herstellen, und zwar zur ↑Laufzeit von dem betreffenden ↑Programm selbst. Ein ↑Prozess in diesem Programm hat eine ungebundene ↑symbolische Adresse verwendet und dadurch eine ↑Ausnahmesituation herbeigeführt: die ↑CPU unterbricht die Ausführung von dem ↑Maschinenbefehl, der diese ↑Adresse hervorgebacht hat (↑*trap*) und löst dessen ↑partielle Interpretation durch das ↑Betriebssystem aus. In der Folge lokalisiert ein ↑dynamischer Binder das adressierte ↑Text-/↑Datensegment, durchläuft die ↑Platzierungsstrategie, um das betreffende ↑Segment in den ↑Prozessadressraum einzublenden und eine ↑Ladeadresse zu erhalten und erstellt mit ihr die ↑Bindung. Zum Abschluss wird die CPU instruiert, die Ausführung des betroffenen Maschinenbefehls zu wiederholen (↑*rerun*).
Grundlage für diese Technik bildet zumindest ein ↑logischer Adressraum, denn das in Frage kommende Segment ist in dem Programm durch ein ↑Symbol repräsentiert und damit auch in der, in dem ↑Lademodul dieses Programms enthaltenen, ↑Symboltabelle vermerkt. Dieses Symbol wurde bei der ↑Übersetzung des Programms erfasst: es steht beispielsweise für ein ↑Unterprogramm, auf das durch einen Maschinenbefehl zum Prozeduraufruf (`call`, x86) kodiert im Programm Bezug genommen; oder es steht etwa für ein ↑Exemplar eines beliebigen Datentypen, auf das lesend/schreibend durch einen Maschinenbefehl (`mov`, x86) kodiert im Programm zugegriffen wird. Die Gültigkeit der damit gegebenen symbolischen Adresse wurde also vor Laufzeit des Programms im Rahmen der in dieser Phase anfallenden Übersetzungs- und (statischen) Bindevorgänge bestätigt, nicht jedoch die Präsenz von ↑Text

oder ↑Daten im Lademodul dazu.
Bildet ein ↑virtueller Adressraum die erweiterte Grundlage, so lässt sich letztendlich der ↑Arbeitsspeicher eines Prozesses über den kompletten ↑Ablagespeicher im ↑Rechensystem ausdehnen. Die dynamisch eingebundenen und durch gewöhnliche Adressen referenzierten Text- und Datensegmente werden bereits im Falle eines logischen Adressraums automatisch und für den Prozess funktional nicht wahrnehmbar aus dem Ablagespeicher geholt, sie liegen dort vielleicht jeweils in einer eigenen ↑Datei vor und wurden womöglich eben keiner ↑Programmbibliothek entnommen. Allerdings erfolgte der „transparente Zugriff“ nur lesend. Durch den virtuellen Adressraum erscheint dieser gesamte ↑Speicher jedoch als ↑virtueller Speicher, womit der „transparente Zugriff“ darauf sodann lesend und schreibend geschehen kann. Dies bedeutet insbesondere, dass jede Datei ein gewöhnliches und direkt im Programm adressierbares Segment darstellt: sie wird ohnehin für gewöhnlich symbolisch adressiert, dann eben implizit bei Zugriffen dynamisch eingebunden und damit ohne expliziten ↑Systemaufruf (`open(2)`, `read(2)`, `write(2)`, `close(2)`) zugänglich. Pionierarbeit dazu leistete ↑Multics, das diesen Ansatz erstmalig umgesetzt hat und so jede im Rechensystem gespeicherte Information jedem Prozess in kontrollierter Weise (durch partielle Interpretation der Speicherzugriffe, scheinbar) direkt zugänglich machte — ein Merkmal, das heutige (2016) Betriebssysteme bei all ihrer Komplexität in anderen Belangen vermissen lassen.

E/A (en.) ↑*I/O*, Abkürzung für ↑Ein-/Ausgabe.

EBCDIC Abkürzung für (en.) *extended binary coded decimal interchange code*, IBM (1963/64). Ein 8-Bit Kode, dessen 256 mögliche Kodewörter jedoch nicht alle verwendet werden. Die ersten 64 Kodewörter sind Steuerzeichen. Seine Besonderheit ist der enge Bezug zum Standardformat einer ↑Lochkarte (IBM: 80 Zeichen pro Zeile, 12 Zeilen) bei der die Buchstaben A–I, J–R und S–Z jeweils in der numerischen Zone (d.h., die Ziffern 0–9) kodiert werden. Der Kode wurde mit ↑IBM System/360 weitläufig eingeführt und findet vornehmlich in der Großrechnerdomäne (↑*mainframe*) Verwendung.

Echtzeit (en.) ↑*real time*. Bezeichnung für die tatsächlich in der Realität verstreichende Zeit, genauer: die einem ↑Prozess im ↑Rechensystem vorgegebene Zeit, die er in der Realität verbrauchen darf. Damit der Prozess seine Zeitvorgaben einhalten kann, ist ↑Echtzeitbetrieb erforderlich. Das bedeutet: Betrieb von einem Rechensystem, bei dem ein ↑Programm zur Verarbeitung anfallender Daten ständig betriebsbereit ist, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen (nach DIN ISO/IEC 2382).

Echtzeitbedingung Umtand in Bezug auf die vorgegebene Zeit, die ein ↑Prozess in der Realität verbrauchen darf. Für den Prozess ist ein Termin festgelegt, zu dem ein durch ihn selbst herbeigeführtes Berechnungsergebnis vorliegen sollte oder muss. Dieser Termin hat eine bestimmte *Kritikalität* und ist als weich (*soft*), fest (*firm*) oder hart (*hard*) qualifiziert; Synonyme für weich und hart sind schwach beziehungsweise strikt:

- Ein weicher/schwacher Termin darf von dem Prozess überschritten werden. Der Prozess wird bei Terminüberschreitung nicht abgebrochen. Ein von ihm verspätet geliefertes Berechnungsergebnis wird nicht verworfen, jedoch nimmt die Bedeutung dieses Ergebnisses mit zunehmender Verspätung immer weiter ab.
- Ein fester Termin darf von dem Prozess überschritten werden. Der Prozess wird bei Terminüberschreitung abgebrochen, aber erneut für den nächsten Durchlauf (Periode) bereitgestellt. Ein Berechnungsergebnis wird nicht geliefert.
- Ein harter/strikter Termin darf von dem Prozess nicht überschritten werden. Eine gegebenenfalls dennoch stattfindende Terminüberschreitung durch den Prozess ist eine

↑Ausnahmesituation, deren Behandlung im Kontext der Anwendung (↑Maschinenprogramm) zu erfolgen und das System in einen sicheren Zustand zu bringen hat. Ein Berechnungsergebnis wird nicht geliefert.

In jedem Fall überprüft das ↑Betriebssystem fortlaufend für jeden solchen zeitabhängigen Prozess, ob eine Terminüberschreitung vorliegt und leitet entsprechend der jeweiligen Kritikalitätsstufe die erforderlichen Maßnahmen ein. Nur im Falle der Überschreitung eines harten Termins übergibt das Betriebssystem die Zuständigkeit für den weiteren Rechenbetrieb an das Maschinenprogramm, da nur die Anwendung selbst den sicheren Zustand des Systems zu erkennen und herbeizuführen vermag — wenn überhaupt. Wird das System nicht in einen sicheren Zustand überführt, kann dies eine *Katastrophe* zur Folge haben: eine größere Gefährdungs- und Gefahrenlage oder ein Schadensereignis materieller oder wirtschaftlicher Natur oder bezogen auf ein oder mehrere Lebewesen (Mensch, Tier).
Hintergrund ist für gewöhnlich der ↑Echtzeitbetrieb, dass heißt, wenn die Ausführung von Maschinenprogrammen eine Abhängigkeit vom physikalischen Zeitpunkt der Erzeugung und Verwendung der Berechnungsergebnisse definiert. Solche Maschinenprogramme interagieren typischerweise mit „externen Prozessen“ im Rahmen einer Regelschleife (↑*control loop*). Für gewöhnlich handelt es sich dabei um technische (physikalische/chemische) Prozesse, die in technischen Anlagen oder Maschinen stattfinden. Aber auch der Mensch ist oft in solch einer Regelschleife eingebunden und kann Schritte auslösen, die in Echtzeit zu bearbeiten sind. Beispiel ist eine manuell ausgelöste Reaktorschnellabschaltung (*scram*, für „verduften“), wenn das Bedienpersonal die Überschreitung kritischer Grenzwerte erkennt. Ein weiteres Beispiel ist die Verarbeitung von Datenströmen, hier insbesondere Audio- oder Videoströme (*streaming media*). Je nach Anwendungsfall gelten für solch eine Verarbeitung sehr unterschiedliche Kritikalitätsstufen: weich/fest für digitales Fernsehen (Einzelbildverlust ist unkritisch, wegen möglicher Qualitätseinbußen aber unerwünscht) und hart für autonomes Fahren (Einzelbildverlust ist kritisch, sollten dadurch Brems- und Lenkvorgänge zu spät ausgelöst werden).

Echtzeitbetrieb Bezeichnung für eine ↑Betriebsart, die einen ↑Prozess im ↑Rechensystem in ↑Echtzeit stattfinden lässt. Ein oder mehrere Prozesse operieren unter wenigstens einer bestimmten ↑Echtzeitbedingung, die durch die jeweilige Umgebung des Rechensystems vorgegeben sind. Zeit stellt damit keine intrinsische Eigenschaft des Rechensystems dar, sondern ist durch die in der Umgebung gültigen Zeitskala definiert. Nach dieser Zeitskala haben sich bestimmte Abläufe innerhalb des Rechensystems zu richten.
Die Verarbeitung der zeitabhängigen Prozesse (↑*process control*) geschieht zeit- oder ereignisgesteuert. Bei zeitgesteuerter Verarbeitung finden die betreffenden Prozesse immer nur zu definierten Zeitpunkten statt (↑*time-triggered system*). Diese Zeitpunkte definiert zwar das Rechensystem (↑*timer*), jedoch sind sie passend zu der in der Umgebung des Rechensystems existierenden Zeitskala gewählt. Die Zeitintervalle sind für gewöhnlich gleich lang (äquidistant) und fest. Die Ablaufplanung für die Prozesse geschieht rechnerunabhängig (*off-line*), sie liefert einen zur ↑Laufzeit statischen ↑Ablaufplan. Demgegenüber finden bei einer ereignisgesteuerten Verarbeitung die Prozesse entsprechend ihrer jeweiligen ↑Dringlichkeit zeitnah zu dem ihnen jeweils zugeordneten ↑Ereignis statt, die genauen Zeitpunkte für diese Prozesse sind jedoch unbekannt (↑*event-triggered system*). Die Ablaufplanung erfolgt mitlaufend (*on-line*) zu den jeweils einzuplanenden Prozessen, sie resultiert in einen dynamischen Ablaufplan, der zur Laufzeit variiert. Aus der Dringlichkeit eines Prozesses wird seine ↑Priorität abgeleitet, die ihm eine bestimmte Position auf der ↑Bereitliste einräumt und, in Abhängigkeit vom ↑Verdrängungsgrad, mehr oder weniger zügig und bestimmt (vorhersagbar) die ↑CPU zuteilt (↑*preemption*).
Die ereignisgesteuerte Verarbeitung von Prozessen bietet höhere Flexibilität, erschwert jedoch die ↑Vorhersagbarkeit der Abläufe im Rechensystem. Umgekehrt die zeitgesteuerte Verarbeitung, die aufgrund des statischen Ablaufplans kaum Flexibilität zeigt, dafür aber mit starker Vorhersagbarkeit dieser Abläufe aufwarten kann. Welche Verarbeitungsart zu bevorzugen oder gar gefordert ist, steht und fällt mit dem jeweiligen Anwendungsfall und

dem jeweils verfügbaren Anwendungswissen. Zeitgesteuerte Verarbeitung erfordert *Vorabinformationen* zu den Prozessen: zu ihrer jeweiligen Dauer (↑WCET) wie auch zur jeweiligen Länge ihrer eventuellen Periode, zu bestehenden Abhängigkeiten zwischen den Prozessen und über die von den Prozessen belegten ↑Betriebsmittel. Ist dieses *a priori* Wissen nicht verfügbar, bleibt nur die ereignisgesteuerte Verarbeitung der Prozesse. Ist dieses Wissen jedoch verfügbar, hat zeitgesteuerte Verarbeitung zudem den großen Vorteil, einen vergleichsweise schlanken Rechnerbetrieb zu ermöglichen — auch wenn vorhersagbares Verhalten keine Rolle spielt und daher ereignisgesteuerte Verarbeitung ebenfalls Option wäre.

Echtzeitsystem Bezeichnung von einem ↑Rechensystem, für das ↑Echtzeitbetrieb durchzusetzen ist.

EDF Abkürzung für (en.) *earliest deadline first*. Ein bestimmter ↑Einplanungsalgorithmus in einem ↑Betriebssystem. Zeitbasiertes Verfahren, das dem ↑Prozess die höchste ↑Priorität gibt, dessen Termin als nächster ansteht. Für ↑Echtzeitbetrieb (↑*event-triggered system*) bedeutet dies insbesondere auch ↑Verdrängung, vorzugsweise mit einem ↑Verdrängungsgrad von 100 % (sog. volle Verdrängung, (en.) *full preemption*) und konstanter ↑Latenz.

edge-triggered interrupt (dt.) flankengesteuerte ↑Unterbrechung, ↑Flankensteuerung.

EDV Abkürzung für *elektronische Datenverarbeitung*. Sammelbegriff für den Umgang mit ↑Daten in einem ↑Rechensystem, um dadurch Informationen zu gewinnen und darzustellen und die gewonnenen Informationen für die Erzeugung weiterer Daten zu nutzen. Die zu verarbeitenden Daten liegen rechnerunabhängig (*off-line*) oder mitlaufend (*on-line*) zu einem ↑Prozess in einem ↑Speicher im Rechensystem vor.

EEPROM Abkürzung für (en.) *electrically erasable PROM*, (dt.) elektrisch löschbarer ↑PROM.

Eigenvariable (en.) ↑*own variable*. Bezeichnung einer Variablen, deren Wert beim Verlassen des Grundblocks, in dem sie deklariert wurde, namentlich unbekannt wird, aber erhalten bleibt und beim Wiedereintritt in diesen Block automatisch zur Verfügung steht. Geht auf ein mit ↑Algol 60 eingeführtes Konzept zurück, das solche Variablen durch den Zusatz `own` bei der Deklaration auszeichnet. In ↑C wird gleiches durch `static` als Zusatz bei der Deklaration der lokalen Variablen einer Funktion erreicht.

Ein-/Ausgabe Bezeichnung einerseits für den Vorgang zur Kommunikation der innerhalb eines ↑Rechensystems stattfindenden ↑Prozesse mit ihrer Außenwelt und andererseits für die Gesamtheit von ↑Daten, Informationen, die einem Prozess eingegeben und von ihm verarbeitet werden (*input*) beziehungsweise das Arbeitsergebnis eines Prozesses (*output*) bilden.

Ein-/Ausgaberegister Behältnis in einem ↑Peripheriegerät um ↑Daten ein-/auszugeben und die Ein-/Ausgabe zu steuern und zu überwachen. Anzahl und Bedeutung dieser Register sind gerätespezifisch. Typischerweise wird (logisch) unterschieden zwischen Kontroll-, Status- und Datenregister. Diese Register sind zugänglich entweder über eine gewöhnliche ↑Adresse (↑*memory-mapped I/O*) oder über einen speziellen Anschluss (↑*port-mapped I/O*).

Ein-/Ausgabestoß Häufung von Aktivität in Bezug auf die ↑Peripherie (↑*I/O burst*). Die Peripherie führt eigenständig eine oder mehrere Ein-/Ausgabeoperationen für einen ↑Prozess und damit unabhängig von der ↑CPU durch. Diese Operationen sind logisch, aber nicht zwingend physisch verknüpft mit dem Prozess, der sie ausgelöst hat. Der Prozess kann während der Zeit einen beliebigen ↑Prozesszustand haben, das heißt, er ist laufend, bereit oder blockiert. Im letzteren Fall kann es sein, dass der Prozess die Beendigung einer Ein-/Ausgabeoperation erwartet (↑passives Warten) und dadurch physisch mit der Ein-/Ausgabe gekoppelt ist. Eine solche Kopplung kann allerdings auch bestehen, wenn der Prozess laufend ist (↑aktives Warten). In den anderen Fällen findet der Prozess losgelöst von der oder den laufenden Ein-/Ausgabeoperationen statt, die er gegebenenfalls abgesetzt hat, er ist aber logisch mit seiner Ein-/Ausgabe gekoppelt. Wartet ein Prozess nicht die Beendigung seiner

Ein-/Ausgabeoperation(en) ab, kann er weitere an dasselbe oder ein anderes ↑Peripheriegerät abgeben. All diese Operationen geschehen nebenläufig zum ↑Rechenstoß des Prozesses.

Einadressraummodell Art von ↑Schutz in einem ↑Rechensystem, die sicherstellt, dass kein ↑Prozess eine ihm fremde ↑Adresse (innerhalb einer bestimmten Zeitspanne) in Erfahrung bringen kann. Die Annahme dabei ist (a) ein allen Prozessen gemeinsamer, einziger und riesengroßer ↑logischer Adressraum und (b) dass jede vom ↑Betriebssystem zu vergebene ↑logische Adresse als Funktion der ↑Platzierungsstrategie randomisiert wird. Damit kann keine dieser Adressen von einem Prozess erraten werden und bloßes Ausprobieren einer Adresse ist wegen der Adressraumgröße impraktikabel. Dieser Ansatz ist gangbar bei einer ↑Adressbreite ab 48 Bits beziehungsweise einen 2^{48} verschiedene logische Adressen umfassenden ↑Adressraum: mit einer ↑Zugriffszeit von 1 ns pro ↑Byte würde ein kompletter Lauf über den gesamten Adressraum etwa 78 Stunden dauern, was für eine ↑Betriebsart mit kurzlebigen (interaktiven) Prozessen ausreichend Sicherheit bieten kann. Auf Grundlage heutiger (2016) 64-Bit-Technologie würde ein solcher Lauf etwa 585 Jahre dauern und damit allgemein vor Brachialgewalt (*brute force*) schützen, sollte ein Prozess fremde Bestände von ↑Text und ↑Daten zerstören wollen. So ist Schutz vor unautorisierten Zugriffen gegeben, ohne jedoch erlaubte Zugriffe (z.B. nur lesen) weiter qualifizieren und unerlaubte Zugriffe (z.B. alle, außer lesen) abwehren zu können. Für letzteres ist eine Adresse zusätzlich als ↑Befähigung auszulegen. Bei solch einem Ansatz wird eine ↑MMU schließlich nur zur ↑Adressabbildung (d.h., ↑Relokation zur ↑Laufzeit) benötigt, nicht aber zur ↑Adressraumisolation (im Gegensatz zum ↑Mehradressraummodell).

Einbenutzerbetrieb Variante von ↑Dialogbetrieb, bei der das ↑Betriebssystem zu einem Zeitpunkt höchstens einen Teilnehmer den Rechenbetrieb ermöglicht (Gegenteil von ↑Mehrbenutzerbetrieb). Allerdings ist ↑Mehrprogrammbetrieb damit nicht ausgeschlossen: je nach Auslegung des Betriebssystems kann ein Teilnehmer sehr wohl in die Lage versetzt werden, während der ↑Sitzung mehr als ein ↑Programm zur Ausführung zu bringen (↑Teilnehmerbetrieb, ↑Teilhaberbetrieb).

Einfriedung Schutzmechanismus, bei dem ein ↑realer Adressraum (↑Hauptspeicher) an einer bestimmten ↑Adresse, der ↑Gatteradresse, in zwei Zonen aufgeteilt wird: eine geschützte Zone für das ↑Betriebssystem (↑*resident monitor*) und eine ungeschützte Zone für das ↑Maschinenprogramm. Generiert ein ↑Prozess der ungeschützten Zone eine Adresse der geschützten Zone, wird er abgefangen (↑*trap*). Umgekehrt gilt dies nicht zwingend, das heißt, das Betriebssystem hat potentiell freien Zugriff auf beide Zonen. Bevor die von dem Prozess im Maschinenprogramm gebildete effektive ↑reale Adresse zum ↑Adressbus geht, wird sie von der ↑CPU mit der Gatteradresse verglichen. Die vom Prozess generierte (effektive) Adresse geht nur dann zum Adressbus, das heißt, der Zugriff gelingt nur, wenn sie größer beziehungsweise kleiner als die Gatteradresse ist, je nachdem, ob die geschützte Zone den vorderen oder hinteren ↑Adressbereich ausmacht. Das Ergebnis des Vergleichs hat jedoch nur Auswirkung bei Ausführung des Maschinenprogramms, um nämlich Zugriffe auf die geschützte Zone zu unterbinden. Ist das Betriebssystem aktiv, das ja innerhalb der geschützten Zone liegt, würde die CPU sonst mit jedem ↑Maschinenbefehl eine ↑Ausnahmesituation herbeiführen. Daher wechselt die CPU den ↑Arbeitsmodus, je nachdem in welcher Zone sie gerade operiert: Systemmodus (↑*system mode*) für die geschützte und Benutzermodus (↑*user mode*) für die ungeschützte Zone. Normalerweise ist der Benutzermodus aktiv und das Maschinenprogramm wird ausgeführt. Jede ↑Ausnahme aktiviert den Systemmodus. Darüberhinaus ist ein ↑Systemaufruf erforderlich, um die Dienste des Betriebssystems in Anspruch zu nehmen. Diese einfache Technik ist zugleich ein sehr restriktiver Ansatz, da keine der beiden Zonen zum einen über eine bestimmte Grenze hinweg expandieren und zum anderen den nicht ausgeschöpften Bereich (↑interner Verschnitt) der jeweils anderen Zone für den eigenen Bedarf nutzen kann. Zudem ist der Ansatz nur für ↑Einprogrammbetrieb tauglich, da der Schutz durch den Adressvergleich (größer/kleiner) immer nur unidirektional greift.

Eingabetaste Taste auf einer Rechnertastatur zur Bestätigung einer Eingabeaufforderung. Bewirkt beim zeilenorientierten Betrieb einen ↑Wagenrücklauf und ↑Zeilenvorschub.

Eingrenzung ↑Speicherschutz durch ↑Grenzregister. Generiert ein ↑Prozess eine ↑Adresse, die außerhalb der durch „seine“ Grenzregister festgelegten Zone in einem ↑Adressraum liegt, wird die diesbezügliche ↑Aktion eines lesenden oder schreibenden Zugriffs abgefangen (↑*trap*). Für gewöhnlich speichert jedes der Grenzregister eine ↑reale Adresse. Bevor die von dem Prozess gebildete effektive (reale) Adresse zum ↑Adressbus geht, wird geprüft, ob diese in dem durch die beiden Grenzadressen definierten Bereich liegt: $BR_{lwb} \leq \textit{effective address} \leq BR_{upb}$, mit BR (↑*bounds register*) für eins der beiden Grenzregister (*lower* bzw. *upper bound*: *lwb*/*upb*). Liegt die effektive Adresse außerhalb der Grenzen, bricht das ↑Betriebssystem den Prozess in aller Regel ab (↑*segmentation fault*). Ist das Betriebssystem selbst eingegrenzt und generiert ein Prozess, der (als Folge von einem ↑Systemaufruf oder einer ↑Schutzverletzung) im Betriebssystem stattfindet, eine Adresse, die außerhalb der Betriebssystemgrenzen liegt, wird ↑Panik ausgelöst: in dem Fall ist von einem gravierenden Programmierfehler im Betriebssystem auszugehen, der den weiteren gesicherten ↑Programmablauf unmöglich macht.
Diese Form von Speicherschutz ist typisch für eine ↑MPU. Im ↑Prozesskontrollblock einer ↑Prozessinkarnation werden die unteren und oberen Grenzadressen von dem entsprechenden ↑Prozessadressraum als Attribute gespeichert und beim ↑Prozesswechsel in die jeweiligen Grenzregister geschrieben. Da dieser Schreibzugriff eine privilegierte Operation ist und daher im Betriebssystem erfolgt (↑*privileged mode*), müssen im Falle eines selbst eingegrenzten Betriebssystems für den System- und Benutzermodus jeweils eigene Grenzregister vorhanden sein (↑Arbeitsmodus). Beim Prozesswechsel werden die Grenzadressen in die dem Benutzermodus zugeordneten Grenzregister geschrieben, die aber erst nach Verlassen des Systemmodus zur Wirkung kommen. Grundsätzlich werden mit jedem Wechsel vom Benutzer- zum Systemmodus und umgekehrt die dem jeweiligen Modus zugeordneten Grenzregister im ↑Prozessor (bzw. in der MPU) wirksam. Ist das Betriebssystem dagegen nicht eingegrenzt, sind die Grenzregister nur einfach ausgelegt. In dem Fall ist die Überprüfung von Grenzen (*bounds check*) im Systemmodus entweder wirkungslos oder abgeschaltet, die Grenzregister kommen dann nur im Benutzermodus zur Wirkung.

Einhängen Vorgang, um ein auf einen ↑Datenträger liegendes und zur Benutzung vorgesehenes ↑Dateisystem in die Haltevorrichtung (↑*mounting point*) eines bereits in Benutzung befindlichen Dateisystems zu hängen, dadurch daran für den Betrieb zu befestigen und verfügbar zu machen. Die Befestigung ist aber nicht unwiderruflich, sie wird für gewöhnlich gelöst (*dismount*), wenn das eingehängte Dateisystem nicht mehr benötigt beziehugsweise der betreffenden Datenträger wieder ausgeworfen wird. Bei einem ↑Wechseldatenträger läuft der Einhängevorgang häufig auch automatisch ab, unterstützt durch das ↑Betriebssystem, wenn nämlich der ein (einhängbares) Dateisystem speichernde Datenträger über einen geeigneten Anschluss (z.B. ↑USB) für sein ↑Peripheriegerät verfügbar gemacht wird (↑*plug and play*).

Einlastung Vorgang der Belegung einer Verarbeitungseinheit mit einem Auftrag. Die Verarbeitungseinheit ist ein ↑wiederverwendbares Betriebsmittel bestimmter Art, das nach Gebrauch für einen weiteren Auftrag entsprechender Art weiterhin verwendet werden kann. Typisches Beispiel für einen Auftrag ist eine ↑Aufgabe, für deren Erledigung ein ↑Prozess stattfinden und dazu wiederum ein ↑Prozessor zugewiesen, das heißt, unter Last gesetzt werden muss. Nach Gebrauch des Prozessors ist dieser frei für einen weiteren Prozess (↑Prozesswechsel). Anderes Beispiel ist Ein-/Ausgabe, bei der ein ↑Peripheriegerät durch einen Prozess den Auftrag zur Lieferung von Eingabe- oder Annahme von Ausgabedaten erhält. Nach Gebrauch des Geräts ist dieses frei für einen weiteren Ein-/Ausgabeauftrag.

Einlastungslatenz (en.) ↑*dispatching latency*. Bezeichnung für die ↑Latenzzeit bis zur erfolgten ↑Einlastung von einem ↑Prozessor, und zwar relativ zum Zeitpunkt der Feststellung des diesbezüglichen Auftrags zur Ausführung einer bestimmten ↑Aufgabe. Beschreibt der Auftrag einen ↑Prozess und handelt es sich bei dem Prozessor um die ↑CPU, dann entspricht diese

Latenzzeit dem Interval, um den Prozess vom Zustand bereit in den Zustand laufend zu überführen (s. a. ↑Prozesszustand), das heißt, einen ↑Prozesswechsel durchzuführen.
Ein sowohl für ↑vorlaufende Planung als auch ↑mitlaufende Planung anfallendes Zeitintervall, das zum einen durch die ↑Ausführungszeit des ↑Umschalters bestimmt ist und zum anderen jede zu seiner Aktivierung nötige ↑Aktionsfolge einschließt. Unabwendbare ↑Gemeinkosten, sobald ein Prozessor als Folge der ↑Einplanung mehr als einen Prozess zur Verarbeitung zugeteilt bekommen hat und demzufolge im ↑Multiplexverfahren betrieben werden muss.
Für gewöhnlich schlagen diese Gemeinkosten nur zu Buche, wenn der laufende Prozess in den Zustand blockiert wechselt oder die Einplanung feststellt, dass dieser Prozess von der CPU verdrängt werden muss (↑*preemption*). Jedoch nicht jede ↑präemptive Planung hat den Prozessorentzug für einen Prozess zwingend zur Folge: entscheidend ist, dass die ↑Dringlichkeit des einzuplanenden oder eingeplanten Prozesses höher sein muss als die des laufenden Prozesses. In dem Fall gehen diese Gemeinkosten einher auch mit jedem Zustandsübergang eines Prozesses von laufend nach bereit. Dies trifft ebenfalls auf den ↑Rundlauf zu (z.B. ↑RR), durch den ein bestehender ↑Ablaufplan im ↑Zeitteilverfahren verarbeitet wird.

Einplanung Eingeplantsein, etwas in der ↑Planung berücksichtigen (Duden). Einen ↑Auftrag zur Verarbeitung durch einen ↑Prozessor einplanen. Durch dieses ↑Ereignis wird die ↑Bereitzeit der mit dem Auftrag verbundenen ↑Aufgabe festgelegt. Die wirkliche ↑Startzeit dieser Aufgabe liegt jedoch später, sie hängt ab von der Zeitspanne bis zur Entnahme des entsprechenden Auftrags aus der ↑Warteschlange für den Prozessor. Diese Zeitspanne trägt zur ↑Wartezeit des Auftrags bei und beeinflusst damit die ↑Latenz bis zum wirklichen Ausführungsbeginn der betreffenden Aufgabe.
Ein anderer wesentlicher Faktor zur Wartezeit bilden Ereignisse und daraus resultierende Maßnahmen, die zur ↑Unterbrechung der Entleerung oder, möglicherweise ursächlich eben in einer solchen Unterbrechung selbst begründet, vordringlichen Befüllung der Warteschlange führen. Sind diese Ereignisse nicht kontrollierbar, kann die Wartezeit eines Auftrags nicht begrenzt und damit für die betreffende Aufgabe auch keine Aussage zur Start- und effektiven ↑Endzeit, ganz geschweige zur ↑Ausführungszeit, getroffen werden. Dieser Aspekt ist besonders problematisch, wenn die Aufgabe mit einer ↑Frist verknüpft ist, das heißt, ihre Bearbeitung einer ↑Echtzeitbedingung unterworfen ist.

Einplanungsalgorithmus Lösungs- und Bearbeitungsschema, Handlungsvorschrift zur ↑Planung der Vergabe (↑*dispatching*) von einem ↑Auftrag an einen ↑Prozessor. Ist charakterisiert durch die Reihenfolge von Aufträgen in einer ↑Warteschlange und die Bedingungen, unter denen die Aufträge dort eingereiht werden. Die Bewertungsgrundlage (↑*scheduling criteria*) für diese Vorschrift unterscheidet dabei grob zwischen anwendungs- und systemorientierten Kriterien, das heißt, dem in einer Anwendung wahrnehmbaren Systemverhalten einerseits und der effektiven und effizienten Auslastung der ↑Betriebsmittel andererseits.
Beiden Orientierungen mit ein und demselben Verfahren gerecht zu werden, ist in aller Regel nicht möglich. Ein mehrstufiger Ansatz kann Abhilfe schaffen, wobei den einzelnen Stufen dann bestimmten Kriterien zugeordnet sind. In solch einem Szenario erfolgt dann jedoch immer eine Schwerpunktsetzung, die sich durch die Reihenfolge ergibt, in der die einzelnen Ebenen durchlaufen werden und dadurch ein Kriterium ein anderes dominiert.

Einplanungskriterium (en.) ↑*scheduling criteria*. Bewertungsgrundlage für einen ↑Einplanungsalgorithmus. Grob unterschieden wird dabei zwischen anwendungs- und systemorientierten Kriterien, das heißt, dem in einer Anwendung wahrnehmbaren Systemverhalten einerseits und der effektiven und effizienten Auslastung der ↑Betriebsmittel andererseits. Charakteristische Merkmale anwendungsorientierter Kriterien in Bezug auf das ↑Betriebssystem sind ↑Antwortzeit, ↑Durchlaufzeit, ↑Termineinhaltung und ↑Vorhersagbarkeit. Demgegenüber stehen wünschenswerte Merkmale systemorientierter Kriterien wie ↑Durchsatz, ↑Prozessorauslastung, ↑Gerechtigkeit, ↑Dringlichkeit und ↑Lastausgleich. In der Durchsetzung dieser Kriterien ist ↑Proportionalität ein zusätzlicher Aspekt, der die Verhältnismäßigkeit der mit einem Einplanungsalgorithmus möglichen Optimierungen und den sich daraus

ergebenden Konsequenzen für den Entwurf und die Implementierung eines Betriebssystems in den Vordergrund stellt.

Einplanungslatenz (en.) ↑*scheduling latency*. Bezeichnung für die ↑Latenzzeit bis zur erfolgten ↑Einplanung von einem ↑Auftrag für einen ↑Prozessor. Beschreibt der Auftrag einen ↑Prozess und handelt es sich bei dem Prozessor um die ↑CPU, dann entspricht diese Latenzzeit dem Interval, um den Prozess vom Zustand blockiert in den Zustand bereit zu überführen (s. a. ↑Prozesszustand), das heißt, auf die ↑Bereitliste zu platzieren.
Ein nur für ↑mitlaufende Planung anfallendes Zeitintervall, das zum einen durch die ↑Ausführungszeit des ↑Planers bestimmt ist und zum anderen jede zu seiner Aktivierung nötige ↑Aktionsfolge einschließt. Ersteres ist maßgeblich durch den ↑Einplanungsalgorithmus und seiner konkreten Implementierung vorgegeben, wohingegen letzteres insbesondere davon abhängt, ob die ↑Einplanung nebenläufig, das heißt, pseudo- oder echt parallel zu dem gegenwärtig auf dem betreffenden Prozessor stattfindenden Prozess erfolgen kann.
Ist für ↑Synchronisation zu sorgen, wenn nämlich nebenläufige Planerausführung nicht in Frage kommt, fallen ↑Gemeinkosten nichtfunktionaler Art an (↑*ambient noise*). Verbietet der Planer beispielsweise den ↑Wiedereintritt, weil er etwa als ↑kritischer Abschnitt implementiert ist, und läuft bereits ein Planungsvorgang, verzögert sich die erneute Aktivierung des Planers, bis der laufende Planungsvorgang zum Abschluss kommt, das heißt, der kritische Abschnitt verlassen und damit freigegeben wird. Ohne ↑Vorwissen über die maximale Anzahl von Prozessen, die gleichzeitig im Wettstreit um einen durch (in diesem Beispiel mittels) ↑blockierende Synchronisation abgesicherten Planer stehen können, bleibt diese Verzögerung unbestimmt. Dieses Vorwissen kann nur aus dem jeweiligen ↑Anwendungsfall, den das ↑Betriebssystem unterstützen soll, extrahiert werden. Kann dieses Wissen nicht zusammengestellt und für das Betriebssystem genutzt werden, ist die zu erwartende maximale Verzögerung nur noch bestimmbar, wenn der Planer auf ↑nichtblockierende Synchronisation mit ↑Wartefreiheit setzt.

Einprogrammbetrieb Bezeichnung für eine ↑Betriebsart, bei der zu einem Zeitpunkt nur ein ↑Maschinenprogramm im ↑Arbeitsspeicher zur Ausführung bereit steht. Die Inbetriebnahme des Maschinenprogramms geschieht von Hand (↑manuelle Rechnerbestückung) oder programmgesteuert (↑automatisierte Rechnerbestückung), darüber hinaus kann es als ↑sequentielles Programm oder ↑nichtsequentielles Programm ausgelegt sein und damit sequentiell oder (echt/pseudo) parallel ausgeführt werden. Zur Entlastung der ↑CPU von Ein-/Ausgabe ist ↑abgesetzter Betrieb möglich, sofern das ↑Rechensystem eine entsprechende Hardwarekonfiguration aufweist. Ganz allgemein bietet ansonsten noch ↑überlappte Ein-/Ausgabe die Option, ↑Rechenstoß und ↑Ein-/Ausgabestoß des einen in Ausführung befindlichen Maschinenprogramms parallel stattfinden zu lassen und dadurch grundsätzlich die ↑Durchlaufzeit sowie Verweildauer im Arbeitsspeicher zu verringern. Eine weitere, den ↑Durchsatz des Rechensystems steigernde — jedoch ↑Speicher kostende und ↑Hintergrundrauschen erzeugende — Maßnahme besteht in einer Verkürzung der Wechselzeiten zwischen aufeinanderfolgenden Maschinenprogrammen durch die Zwischenpufferung der jeweils als nächstes zu bearbeitenden ↑Aufgabe (↑*single-stream batch monitor*).

Einschaltrückstellung Bezeichnung für eine (schaltungstechnische) Vorrichtung oder den Vorgang, um einen elektronischen ↑Rechner in den Anfangszustand zurückzusetzen. Nach dem Anlegen der Versorgungsspannung, aber erst mit Erreichen eines bestimmten Nennwerts dieser Spannung, sorgt die Maßnahme zunächst für den definierten Zustand der elektronischen Bausteine. Anschließend wird von der Hardware eine ↑Ausnahme erhoben, um (in ↑ROM oder ↑EPROM) permanent residenter Software die weitere Initialisierung des Systems zu übertragen (*reset*: ↑*trap*). Im ↑Hauptspeicher flüchtige Software wird, soweit erforderlich, anschließend durch ↑Urladen ins System gebracht und gestartet.

einseitige Synchronisation Synonym zu ↑unilaterale Synchronisation.

Eintrittsinvarianz (en.) ↑*reentrancy*. Charakteristik eines bestimmten Abschnitts in einem ↑Programm, zur ↑Laufzeit den ↑Wiedereintritt gefahrlos zu erlauben und damit ↑Ablaufinvarianz sicherzustellen.

Einzelstromstapelmonitor Bezeichnung für einen ↑Stapelmonitor, der die gestapelten Aufträge als einzelnen Verarbeitungsstrom ausführt. Grundlage dafür bildet ein ↑Maschinenprogramm, wovon zu einem Zeitpunkt immer nur eins im ↑Hauptspeicher liegend herrscht (↑*uniprogramming*) und zur ↑Laufzeit eine bestimmte Einzelarbeit (↑*job*) leistet. Gegebenenfalls liest der Stapelmonitor im Hintergrund der Ausführung dieses Maschinenprogramms bereits das nächste Maschinenprogramm in Folge ein (↑*overlapped I/O*) und puffert es zwischen, um den Maschinenprogrammwechsel frei von ↑Durchsatz mindernder ↑Wartezeit bewerkstelligen zu können. Sobald das Maschinenprogramm mangels ↑Betriebsmittel nicht weiter ausgeführt werden kann, stoppt der Stapelmonitor seinen Betrieb und erwartet die Beendigung von einem gegebenenfalls noch in der ↑Peripherie nebenläufigen ↑Ein-/Ausgabestoß. Gibt es einen solchen Stoß und wird er fertig, können die dadurch verfügbar werdenden Betriebsmittel zur Fortsetzung der Maschinenprogrammausführung führen. Gibt es keinen solchen Stoß oder werden keine Betriebsmittel durch seine Beendigung freigestellt, kann dies ↑Panik auslösen.

Elementaroperation Primitive einer (realen/virtuellen) Maschine; grundlegende, wesentliche Operation in Bezug auf eine bestimmte Ebene der Abstraktion. Eine ↑Aktion, die auf dieser Ebene in einem Schritt (ungeteilt, atomar) stattzufinden scheint.

Elop Abkürzung für ↑Elementaroperation.

Elterverzeichnis Bezeichnung für ein ↑Verzeichnis, in dem der ↑Name von dem ↑Arbeitsverzeichnis verzeichnet ist. In ↑UNIX ist diesem Verzeichnis der Name „. .“ (↑*dot dot*) zugeordnet. Der Eintrag erlaubt die einfache Navigation zum übergeordneten Verzeichnis im ↑Namensraum, ohne den wirklichen Namen dieses Verzeichnisses kennen zu müssen.

endianness (dt.) ↑Bytereihenfolge.

Endzeit (en.) ↑*completion time*. Bezeichnung für eine ↑Prozessgröße, die den Zeitpunkt festlegt, zu dem die Durchführung einer ↑Aufgabe tatsächlich endet.

Entität Seiendes, das heißt, ein konkreter oder abstrakter Gegenstand, aber auch das Wesen dieses Gegenstands. Sammelbegriff für verschiedene, eindeutig zu bestimmende Gegenstände oder Sachverhalte, über die Informationen zur Verarbeitung durch ein ↑Rechensystem gespeichert werden sollen. Gegenstände des Rechensystems selbst sind die ↑Prozessinkarnation (inkl. ↑Faden, ↑Faser, ↑Fäserchen), der ↑Adressraum, das ↑Segment, die ↑Seite, das ↑Dateisystem, die ↑Datei, der ↑Indexknoten und so weiter, für die Sachverhalte wie etwa Anzahl, Größe, Alter, Lokalität, Geltungsbereich, Eigentümerschaften oder Zustand vermerkt werden.

EPROM Abkürzung für (en.) *erasable PROM*, (dt.) löschbarer ↑PROM.

Ereignis Auftreten von einem Geschehnis hervorgerufen durch einen ↑Prozess, beobachtbar von dem Prozess selbst oder einem anderen Prozess. Das Geschehnis ist im ↑Rechensystem direkt beobachtbar, indem ein Prozess etwa die Veränderung des Inhalts von einem ↑Speicherwort durch Abfragen (*polling*) oder Abwarten (*waiting*) wahrnimmt. Letzteres wird durch ↑unilaterale Synchronisation erreicht, das heißt, der Prozess wartet (aktiv/geschäftig oder passiv/schlafend) solange, bis ihm die Veränderung durch einen anderen Prozess explizit angezeigt wird. Das Geschehnis kann aber auch indirekt beobachtbar sein, wenn es einem Prozess nämlich möglich ist, aus wahrnehmbarem Systemverhalten auf das Auftreten gewisser Vorgänge im Rechensystem zu schließen (↑*covered channel*).

ereignisbasiertes Betriebssystem (en.) ↑*event-based operating system*. Bezeichnung für ein ↑Betriebssystem, dessen grundlegendes Konzept zur Repräsentation einer autarken ↑Aktion oder ↑Aktionsfolge das ↑Ereignis einer ↑Unterbrechung ist: jeder mögliche ↑Handlungsstrang

in dem System hat seinen Ursprung als ↑asynchrone Ausnahme. Die Besonderheit eines solchen Betriebssystems besteht darin, dass sich all diese Handlungsstränge denselben ↑Laufzeitstapel teilen. im Gegensatz dazu steht ein ↑prozessbasiertes Betriebssystem, das jedem Handlungsstrang einen eigenen Laufzeitstapel zugesteht.
Im Betriebssystem alle Handlungsstränge nur auf einen einzigen Laufzeitstapel ablaufen zu lassen reduziert den eigenen Speicherplatzbedarf erheblich, insbesondere wenn daran gedacht wird, dass ein solcher Strang die wesentliche Komponente einer ↑Prozessinkarnation bildet. Allerdings ist damit innerhalb des Betriebssystems ein ↑Prozesswechsel direkt als Folge einer ↑Unterbrechungsbehandlung dann nicht möglich. Stattdessen wird ein solcher Wechsel nur noch an ausgewählten ↑Verdrängungspunkten möglich sein. Das wiederum bedeutet die grundsätzliche Entkopplung von ↑Einplanung, die sehr wohl immer noch asynchron möglich ist, und ↑Einlastung, die nur noch synchron an den Verdrängungsstellen erfolgt.

Ersetzungsalgorithmus Lösungs- und Bearbeitungsschema, Handlungsvorschrift zur ↑Verdrängung einer ↑Seite aus dem ↑Hauptspeicher, zentraler Rechenvorgang einer ↑Ersetzungsstrategie. Ist charakterisiert durch eine bestimmte ↑Heuristik in Bezug auf die von einem ↑Prozess in naher Zukunft benutzten Seiten — mehr dazu aber in SP2.

Ersetzungsstrategie Verfahrensweise nach der ein im ↑Hauptspeicher von einem ↑Prozess belegtes ↑Umlagerungsmittel zur Freigabe bestimmt wird, um ein im ↑Umlagerungsbereich liegendes Pendant desselben oder eines anderen Prozesses einlagern zu können. Ursache dafür ist in aller Regel ein voll belegter Hauptspeicher in dem Moment, wenn ein ↑Prozess bei einem ↑Hauptspeicherfehlzugriff den Ablauf der ↑Ladestrategie ausgelöst hat. Da bei vollem Hauptspeicher offensichtlich kein passendes ↑Loch für den Ladevorgang bleibt, ist wenigstens eins der bereits geladenen Umlagerungsmittel von seinem Platz im Hauptspeicher zu verdrängen. Hintergrund ist ↑virtueller Speicher, bei dem eine solche ↑Verdrängung durch das ↑Betriebssystem auch ohne explizit im ↑Maschinenprogramm kodierte und per ↑Systemaufruf übermittelte Hinweise auskommen muss. Das besondere Problem dabei ist, den in zeitlicher Hinsicht am besten geeigneten Platz herauszufinden, nämlich ein von keinem derzeitigen Prozess mehr benutztes aber eingelagertes Umlagerungsmittel festzustellen. Allerdings ist damit gleichsam eine zu treffende Aussage über das zukünftige Verhalten eines Prozesses verbunden, nämlich dass ein als unbenutzt festgestelltes Umlagerungsmittel nicht schon mit dem nächsten ↑Maschinenbefehl von einem Prozess gleich wieder benutzt wird. Welche zukünftige ↑Aktion ein Prozess tätigt, kann aber gerade in einem dynamischen System nicht exakt bestimmt werden. Bestenfalls lässt sich eine Schätzung vornehmen, die von dem vergangenen und gegenwärtigen Verhalten eines Prozesses ausgeht und daraufhin auf sein zukünftiges Verhalten schließt. Dies ist der Knackpunkt der Verfahrensweise, nämlich die Verdrängung eines Umlagerungsmittels „minimalinvasiv“ für die Prozesse vorzunehmen. Ein erster wichtiger Schritt in diese Richtung ist die Beschränkung auf die ↑Seite als Umlagerungsmittel. Damit gestaltet sich die Suche nach dem passenden Loch/Platz im Hauptspeicher im Vergleich zum ↑Segment als sehr einfach. Folglich bildet ein ↑seitennummerierter Adressraum die Basis für virtuellen Speicher, was allerdings ↑seitennummerierte Segmentierung sehr wohl einschließt: in beiden Fällen werden ausschließlich Seiten ersetzt, womit gar die Ersetzung eines kompletten seitennummerierten Segments gemeint sein kann. Die Bestimmung der zu ersetzenden Seiten folgt dann einem ↑Ersetzungsalgorithmus, für den zu Beginn seiner Berechnung niemals alle für eine optimale Lösung benötigen Eingabedaten vorliegen (Online-Algorithmus).

event-based operating system (dt.) ↑ereignisbasiertes Betriebssystem.

event-triggered system (dt.) ereignisgesteuertes System. Ein ↑Rechensystem dessen ↑Echtzeitbetrieb jede anstehende ↑Aufgabe zu der jeweils für sie vorgegebenen (statischen/dynamischen) ↑Priorität durchführt.

exception (dt.) ↑Ausnahme.

exception dispatcher (dt.) ↑Ausnahmezuteiler.

exception handler (dt.) ↑Ausnahmehandhaber.

exception handling (dt.) ↑Ausnahmebehandlung.

execution time (dt.) ↑Ausführungszeit.

Exemplar Einzelstück aus einer Menge gleichartiger Stücke (Duden). Die Stücke sind gleichartig, weil ihnen dasselbe Modell, dieselbe Bauart zugrunde liegt: sie sind vom selben *Bautyp*. So ist eine Programmvariable ein solches Einzelstück, ebenso wie ein Objekt allgemein.

exponentielle Glättung Bezeichnung für ein Verfahren der ↑Zeitreihenanalyse, mit dem anhand eines gemessenen Werts x_t der jüngsten Vergangenheit und eines prognostizierten Werts s_t der Gegenwart ein zu erwartender Wert s_{t+1} für die nahe Zukunft abgeschätzt wird. Zur Gewichtung des Einflusses des gemessenen und prognostizierten Wertes auf die Schätzung findet ein Glättungsfaktor α Verwendung. Eine solche Glättung (1. Ordnung) wird typischerweise mit folgender rekursiven Formel berechnet:

$$s_{t+1} = \alpha \cdot x_t + (1 - \alpha) \cdot s_t, \text{ mit } 0 \leq \alpha \leq 1.$$

Mit dem Ansatz kann für die ↑mitlaufende Planung bei Bedarf und sehr einfach die Länge des nächsten zu erwartenden ↑Rechenstoßes eines ↑Prozesses abgeschätzt werden (vgl. ↑SPN).

externer Verschnitt ↑Verschnitt im ↑Hauptspeicher. Der Grad der ↑Fragmentierung ist so hoch, dass kein einziger freier Abschnitt (↑*hole*) für die ↑Speicherzuteilung nutzbar ist. Oft sind in der Situation zwar genügend viele ↑Byte im Hauptspeicher frei, sie liegen jedoch zu stark verstreut vor und bilden damit keinen zusammenhängenden Abschnitt angeforderter Größe. Der Verschnitt lässt sich durch ↑Defragmentierung auflösen.

FAA Abkürzung für (en.) *fetch and add*, (dt.) abrufen und addieren. Atomare Veränderung von einem ↑Speicherwort: alten Wert abrufen, dann addieren, die Summe als neuen Wert speichern und den altern Wert als Ergebnis liefern (Postinkrement).

Faden Bezeichnung für einen ↑Handlungsstrang, der einen eigenen ↑Laufzeitkontext besitzt und typischerweise mitlaufender (*on-line*) ↑Ablaufplanung unterliegt. Grob differenziert in ↑Anwendungsfaden und ↑Systemkernfaden.

Fäserchen Bezeichnung für einen ↑Faden, der ausschließlich im ↑Betriebssystemkern (↑Linux) residiert. Ein solcher Faden bildet die technische Grundlage, um eine ↑Systemfunktion, etwa ausgelöst durch einen asynchronen ↑Systemaufruf, unabhängig von anderen Vorgängen stattfinden zu lassen.

false sharing (dt.) ↑irrige Mitbenutzung.

FAS Abkürzung für (en.) *fetch and store*; Bezeichnung für eine ↑atomare Operation mit zwei Operanden: `word_t FAS(word_t *, word_t)`. Der erste Operand ist die ↑Adresse eines ↑Speicherworts, dessen Wert vor dem Überschreiben mit dem Wert des zweiten Operanden gelesen und zurückgeliefert wird. ↑GCC definiert dafür die atomare Standardfunktion (*atomic built-in*) `__sync_lock_test_and_set(ref, val)` und bildet diese für ↑x86 auf den unteilbaren ↑Maschinenbefehl `xchg` ab.

Faser Bezeichnung für einen ↑Faden, der strikt kooperativer ↑Ablaufplanung unterliegt und damit nicht gleichzeitig mit anderen seiner Art abläuft. Auch *leichtgewichtiger Faden*, da bei dieser Art der Ablaufplanung die Kontrolle durch das ↑Betriebssystem entfällt, der Aufwand für die ↑Zustandssicherung auf ein Minimum reduziert werden kann und ↑Synchronisation implizit sichergestellt ist. Verschiedentlich auch einer ↑Koroutine gleichgestellt.

FB Abkürzung für (en.) *feedback*. Bezeichnung für eine ↑Planung von ↑Prozessen, bei der ....

FCFS Abkürzung für (en.) *first-come, first-served*. Bezeichnung für eine Strategie zur ↑Planung der Bearbeitung von ↑Aufgaben durch einen ↑Prozessor in der Reihenfolge ihrer ↑Ankunftszeit. Sowohl ↑vorlaufende Planung als auch ↑mitlaufende Planung kann nach diesem Schema vorgehen. Charakterisiert im zuletzt genannten Fall die Aufgabenbearbeitung einen bestimmten ↑Prozess, so ist das Verfahren zudem ein Beispiel für die ↑kooperative Planung: es funktioniert dann nach dem ↑Windhundprinzip und greift immer, sobald eine der Aufgaben in den ↑Prozesszustand bereit gebracht wird. In Bezug auf die Reihung handelt es sich um eine zu ↑FIFO analoge Methode, die allerdings weder das ↑Puffern von ↑Daten noch die Abfederung gegenläufiger Wirkungskräfte bezweckt.
Die Bereitstellung von Aufgaben geschieht synchron und, je nach ↑Operationsprinzip des ↑Betriebssystems, asynchron zum jeweils auf dem betreffenden Prozessor stattfindenden Prozess. Im synchronen Fall werden die Aufgaben direkt oder indirekt durch ↑Systemaufrufe und damit von dem jeweils stattfindenden Prozess selbst eingeplant. Demgegenüber ist die Aufgabenplanung im optionalen asynchronen Fall die mögliche Folge einer ↑Unterbrechungsanforderung. Allerdings ändert letztere nichts an der Bearbeitungsreihenfolge der Aufgaben, sondern bedeutet lediglich ein zweites Moment der Einspeisung zur Bearbeitung bereiter Aufgaben: ein durch die Unterbrechungsanforderung möglicher Entzug des Prozessors findet grundsätzlich nicht statt.
Für die Prozesse allgemein ist dies eine faire Planungsstrategie, jedoch kann der dem Verfahren innewohnende ↑Konvoieffekt die Auslastung von ↑Betriebsmitteln, die insbesondere zur Verwaltung oder Durchführung von ↑Ein-/Ausgabe anfallen, erheblich beeinträchtigen. Demgegenüber ist die Implementierung des Verfahrens alles andere als kompliziert, sie ermöglicht zudem eine sehr kruze ↑Einplanungslatenz. Wenn der Prozessmix durchgehend (nahezu) homogen ist, wird das Verfahren auch zu einer guten Auslastung der Betriebsmittel führen können. Allerdings ist ein solcher Mix gerade bei ↑Simultanverarbeitung im Allgemeinen nicht gegeben, weshalb andere Einplanungsverfahren erforderlich sind, die dann auch an Komplexität zunehmen.

Exkurs Grundlage für die Implementierung des Verfahrens ist für gewöhnlich ein dynamischer ↑Datentyp, mit dem eine ↑Schlange der ein- und ausgehenden Aufgaben geformt werden kann. Damit ist gerade im Falle mitlaufender Planung die Reihung der Prozesse in konstanter ↑Laufzeit möglich, wie nachfolgend beispielhaft gezeigt wird:

```
void ready(process_t *task) {                /* schedule according to FCFS */
    state(&task->mood, READY);                 /* set process ready to run */
    enqueue(labor(), &task->line);     /* add its descriptor to ready list */
}
```

Die Einreihungsoperation (`enqueue`, vgl. S. 129) platziert die einzuplanende Aufgabe immer ans Ende der Schlange (`labor`). Wie bereits erwähnt kann die Bereitstellung (`ready`) synchron oder asynchron ausgelöst erfolgen. Ist jedoch letzteres im Betriebssystem vorgesehen, sind sämtliche Operationen des mitlaufenden ↑Planers, zusätzlich zu ihrer eigentlichen Funktion, der ↑Synchronisation zu unterziehen (↑*cross-cutting concern*).
Neben der Bereitstellung umfasst jeder mitlaufende ↑Planer noch weitere Operationen, die allgemein und in verschiedener Hinsicht die ↑Umplanung betreffen. Beispielsweise ermöglichen nachfolgende Operationen die eventuelle Unterbrechung des derzeitigen Prozesses und Freigabe des Prozessors:

```
void pause() {                              /* reschedule according to FCFS */
    ready(being(ONESELF));                   /* put oneself onto ready list */
    seize(elect(labor()));          /* switch to next ready-to-run process */
}

void check() {                               /* try to reschedule processor */
    if (ahead(labor()))                       /* any other process present? */
        pause();                                          /* yes, reschedule */
}
```

Für kooperative Planung wird der gegenwärtige Prozess von Zeit zu Zeit seinen Prozessor zur Disposition stellen und einen ↑Prozesswechsel erbeten (`pause`). Jedoch pausiert der Prozess nur, wenn festgestellt wird (`check`), dass die ↑Bereitliste nicht leer ist und eine andere Aufgabe übernommen werden kann (`ahead`). Zu diesem Punkt der Prozesspause kommt noch ein weiterer Aspekt hinzu, nämlich Unterstützung für die ↑logische Synchronisation. Diese ergibt sich zwangsläufig, wenn Prozesse sich nicht nur dem ↑Betriebssystem gegenüber kooperativ verhalten müssen, sondern auch untereinander. Eine zentrale Operation dazu ist, einen Prozess, der auf ein bestimmtes ↑Ereignis warten muss, zu blockieren:

```
void block() {                          /* release and reschedule processor */
    process_t *next, *self = being(ONESELF);
    state(&self->mood, BLOCKED);              /* mark myself blocked, also */
    next = quest(labor());               /* search for next one to execute */
    if (next != self)                              /* any other than myself? */
        seize(next);                           /* yes, switch to that process */
    else                                                    /* no, continue */
        state(&self->mood, FLUSH|RUNNING);    /* mark myself running, only */
}
```

Zu beachten ist der ↑Schwebezustand, den ein blockierender Prozess einnimmt: in dem Moment nämlich, wo dieser logisch in den ↑Prozesszustand blockiert (*blocked*) übergeht, ist er immer noch physisch laufend (*running*). Dieser Schwebezustand endet erst, wenn der betreffende Prozess den Prozessor entweder wirklich abgibt (`seize`) oder behalten darf (sonst-Zweig der Fallunterscheidung).
Die Blockierung eines Prozesses impliziert die Auswahl (`quest`) des Prozesses, der als nächster der ↑Einlastung (`seize`) zuzuführen ist. Dieser nächste Prozess kann jedoch derjenige sein, der sich gerade auf dem Wege zur Prozessorabgabe befindet: denn das Ereignis, das dieser Prozess erwartet, kann zwischenzeitlich eingetreten und beispielsweise durch eine ↑Unterbrechungsanforderung angezeigt worden sein. Im Rahmen der ↑Unterbrechungsbehandlung kann dieser Prozess sodann wieder bereitgestellt werden, so dass er sich (in `quest`) selbst auf der Bereitliste wiederfindet.
Daher wird der Prozesswechsel (`seize`) nur bedingt geschehen. Das Bestreben, einen bereitgestellten Prozess zu liefern, kann jedoch mit ↑Leerlauf einhergehen, nämlich wenn in dem Moment die Bereitliste keine Prozesseinträge enthält. Die betreffende Funktion (`quest`) muss somit die Tatsache berücksichtigen, gegebenenfalls auf die Bereitstellung eines Prozesses unbestimmt lang warten zu müssen:

```
process_t *quest(backlog_t *list) {     /* seek for a ready-to-run process */
    process_t *next;
    while ((next = elect(list)) == 0)         /* any ready-to-run process? */
        stall();                                 /* no, shutdown operation */
    return next;                            /* ready-to-run process elected */
}
```

In diesem Beispiel übernimmt immer der Prozess, der im Falle einer leeren Bereitliste blockiert, die Rolle des ↑Leerlaufprozesses (`stall`). Für gewöhnlich versetzt dieser den Prozessor in den ↑Schlafzustand, der erst durch eine Unterbrechungsanforderung beendet wird.

Kommt es als Folge einer solchen Anforderung zur Bereitstellung (`ready`) eines Prozesses, wird dieser nach Wiederaufnahme des Leerlaufprozesses gefunden (`elect`) und als Ergebnis zurückgeliefert. Dabei ist zu beachten, dass dieses Ergebnis eben auch genau jener Prozess sein kann, der nicht nur diese Schleife, sondern auch den Leerlauf kontrolliert hat.
Zu guter Letzt noch einen Hinweis zur Auswahlfunktion (`elect`), die einen bereitgestellten Prozess der Bereitliste entnehmen soll. Für das hier erklärte Verfahren entfernt diese Funktion lediglich das Kopfelement von der betreffenden Schlange (`dequeue`):

```
process_t *elect(backlog_t *list) {          /* choose ready-to-run process */
    return forge(dequeue(list));
}
```

Jedoch ergibt sich dabei ein Typkonflikt, da das Kettenglied (`chain_t`) zwar im ↑Prozesskontrollblock (`process_t`) enthalten ist, dieser selbst aber kein Kettenglied ist. Daher ist aus dem der Schlange entnommenen Kettenglied noch ein Prozesskontrollblock zu formen:

```
process_t *forge(chain_t *item) {                   /* make process pointer */
    return (process_t *)coerce(item, (int)&((process_t *)0)->line);
}
```

Diese „magische“ Anweisung sorgt für eine dynamische Typumwandlung: sie nötigt (`coerce`) einen Zeiger (`chain_t *`) auf ein Kettenglied als Zeiger (`process_t *`) auf einen Prozesskontrollblock zu erscheinen, wobei letzterer das Kettenglied (`line`) als Attribut enthalten muss. Derartige Formulierungen sind in ↑C mangels Spracheigenschaften hin und wieder leider nicht zu vermeiden. Dahinter steckt eine Zeigermanipulation folgender Art:

```
inline void *coerce(void *ptr, int off) {           /* ugly, but needed... */
    return ptr ? (void *)((unsigned)ptr - off) : 0;
}
```

Anderen Sprachen (↑C++) bieten dazu einen speziellen Operator (`dynamic_cast`), wobei dann der ↑Kompilierer eine Anweisung eben gezeigter Form generiert.

feather-weight process (dt.) ↑federgewichtiger Prozess.

federgewichtiger Prozess Bezeichnung für einen ↑Prozess, der als ↑Anwendungsfaden mit anderen Prozessen seiner Art zusammen im gemeinsamen und durch ↑Speicherschutz isolierten ↑Adressraum stattfindet.

fence (dt.) ↑Schutzgatter.

fence address (dt.) ↑Gatteradresse.

fence register (dt.) ↑Schutzgatterregister.

Fernschreiber Gerät, schreibmaschinenähnlich, das der Aufnahme und Übermittlung von Schriftzeichen dient (Duden) und mit ↑Tabellierpapier bestückt ist. Auch als ↑Dialogstation genutzt.

Fernschreibkode Schlüssel, mit dessen Hilfe chiffrierte ↑Daten übertragen werden können. Ein 5-Bit Kode, in zwei Versionen standardisiert durch das ↑CCITT: Kode Nr. 1, auch Baudot-Kode, nach Jean-Maurice-Émile Baudot (1845–1903) und Kode Nr. 2, auch Murray-Kode, nach Donald Murray (1865–1945). Letzterer findet als Umschaltkode breite Verwendung in der ↑EDV. Durch zwei Steuerzeichen wird zwischen Buchstaben- oder Ziffernmodus gewählt: `LS` (*letter shift*), 11111_2, Buchstabenumschaltung und `FS` (*figure shift*), 11011_2, Ziffernumschaltung. So werden 26 Bitkombinationen, die für die Buchstaben im lateinischen Alphabet stehen, doppelt belegt. Im Ziffernmodus sind jedoch drei Bitkombinationen keinem Zeichen zugewiesen (reserviert). Weitere (in beiden Modi gültige) Steuerzeichen sorgen für Zeilenvorschub (LF, *line feed*), 01000_2), ↑Wagenrücklauf (CR, *carriage return*), 00010_2) und Zwischenraum (*space*, 00100_2). Die Bitkombination 00000_2 wird nicht verwendet. Ausgenommen LS, FS und den vier unbelegten Bitkombinationen umfasst der Kode damit 52 Zeichen.

Festwertspeicher ↑Speicher, dessen Inhalt nach dem Beschreiben nur noch abgerufen, aber nicht mehr verändert werden kann (in Anlehnung an den Duden).

fetch policy (dt.) ↑Ladestrategie.

fetch-execute-cycle (dt.) ↑Abruf- und Ausführungszyklus.

fiber (dt.) ↑Faser.

fibril (dt.) ↑Fäserchen.

FIFO Abkürzung für (en.) *first in, first out*. Bezeichnung für ein Vorgehen bei der Organisation eines ↑Puffers und Verarbeitung der darin gespeicherten ↑Daten. In Bezug auf die Reihung handelt es sich um eine zu ↑FCFS analoge Methode, die allerdings weder die ↑Planung von ↑Aufgaben noch die Bildung einer Reihenfolge von ↑Prozessen bezweckt.

file (dt.) ↑Datei.

file system (dt.) ↑Dateisystem.

file tree (dt.) ↑Dateibaum.

first fit (dt.) erstbeste Passung: ↑Platzierungsalgorithmus.

first-class object (dt.) ↑Objekt erster Klasse.

flag bit (dt.) ↑Markierungsbit.

Flankensteuerung Auslöser für die ↑Unterbrechung ist ein Wechsel des Pegelstands auf der Signalleitung (↑*interrupt line*) zur ↑CPU, hervorgerufen durch die ↑Peripherie. Der Impuls zu diesem Wechsel ist nur lang genug, um von der CPU als ↑Unterbrechungsanforderung erkannt zu werden. Gängig ist die Zwischenspeicherung (↑*latch*) des Signals, da die CPU normalerweise nur zwischen zwei Durchläufen des ↑Abruf- und Ausführungszyklus, also nach der Ausführung von einem ↑Maschinenbefehl, den Unterbrechungszyklus fährt. Jedoch erkennt die CPU nur höchstens einen Impuls pro Durchlauf, wiederholte Impulse (*reassertion*) gehen unerkannt verloren. Eine ↑Unterbrechungssperre verlängert dieses Intervall und erhöht die Wahrscheinlichkeit verpasster Unterbrechungsanforderungen. Dieser Aspekt ist besonders virulent bei Mitbenutzung der Störleitung (*interrupt sharing*), wenn nämlich mehr als ein ↑Peripheriegerät entweder direkt oder indirekt, im letzteren Fall durch einen ↑PIC, an die CPU angeschlossen werden soll. Wiederholte Impulse auf der Störleitung sind durch die verschiedenen und voneinander unabhängigen Peripheriegeräte in solch einer Konstellation Normalität. Damit ist der flankengesteuerte Ansatz recht anfällig für den Verlust von Unterbrechungsanforderungen, im Gegensatz zur ↑Pegelsteuerung.

FLIH Abkürzung für (en.) *first-level interrupt handler*. Bezeichnung für den ↑Unterbrechungshandhaber erster Stufe. Dieser ↑Handhaber wird direkt durch eine ↑Unterbrechungsanforderung gestartet und beginnt seine Ausführung auf der durch die Hardware (↑CPU, ↑PIC oder ↑Peripherie) bestimmten ↑Unterbrechungsprioritätsebene. Die durch ihn erfolgende ↑Unterbrechungsbehandlung findet grundsätzlich asynchron zum auf ↑Benutzerebene oder ↑Systemebene unterbrochenen ↑Prozess statt.
Der Wirkungskreis des Handhabers ist stark eingeschränkt und erstreckt sich in erster Linie nur auf die Bedienung des Geräts, das die ↑Unterbrechung hervorgerufen hat. Seine ↑Aufgabe ist die explizite Bestätigung der Unterbrechungsanforderung an dem betreffenden Gerät (was bei ↑Pegelsteuerung zwingend erforderlich ist), das Lesen oder Schreiben von ↑Daten über ↑Ein-/Ausgaberegister und, sofern ↑E/A-Aufträge vorliegen, den nächsten ↑Ein-/Ausgabestoß abzusetzen. All diese ↑Aktionen bedeuten ↑programmierte Ein-/Ausgabe (auch wenn für den eventuellen Datentransfer ↑DMA zum Einsatz kam), sie müssen schnell, zügig und vor allem in endlicher Zeit erfolgen, damit der unterbrochene Prozess nur auf das

Kürzeste und darüber hinaus begrenzt verzögert wird.
Insbesondere wegen der Asynchronität zu Prozessen der Systemebene ist beim Aufruf einer ↑Systemfunktion durch den Handhaber größte Vorsicht geboten. Nur wenn alle wettlaufgefährdeten ↑Aktionsfolgen im Betriebssystem (a) jeweils als ↑kritischer Abschnitt ausgelegt und (b) mittels ↑Unterbrechungssperren abgesichert sind, darf ein solcher Aufruf überhaupt in Erwägung gezogen werden. Finden solche Sperren im Betriebssystem nämlich Verwendung, konnte kein kritischer Abschnitt aktiv gewesen sein, wenn der Handhaber zur Ausführung gelangt ist — womit allerdings nicht zum Ausdruck gebracht werden soll, Unterbrechungssperren seien die zu bevorzugenden Mechanismen zur ↑Synchronisation.
Für eine ↑Betriebsart, die auf einen ↑Uniprozessor ausgerichtet ist, wäre mit solchen Sperren ein gangbarer Weg eröffnet — allerdings nur, wenn ein ↑IRQ die Unterbrechungsursache war. In dem Fall ist aber vor Aufruf einer Systemfunktion die Unterbrechungspriorität wieder zu senken, um die ↑Unterbrechungslatenz für Handhaber derselben Prioritätsebene nicht unnötig zu verlängern. Dazu ist auf die zum Zeitpunkt der Handhaberaktivierung gültigen Prioritätsebene zurückzugehen, mit der möglichen Konsequenz, dass derselbe Handhaber, während er noch aktiv ist, durch eine nachfolgende Unterbrechungsanforderung erneut zur Ausführung gelangt (indirekte ↑Rekursion). All diese Reinkarnationen des einen Handhabers verwenden denselben Laufzeitstapel, der niemals vor Erreichen der Abbruchbedingung der Rekursion überlaufen darf. Je tiefer der Handhaber jedoch durch direkte/indirekte Aufrufe von Systemfunktionen ins Betriebssystem eindringt, umso stärker dehnt sich sein Laufzeitstapel aus. Gegebenenfalls bieten CPU oder Betriebssystem sogar nur einen einzigen Laufzeitstapel für alle Handhaber im System, womit die Stapelkapazität sehr schnell an ihre Grenze kommt. Wird diese Grenze überschritten, verursacht dies für gewöhnlich ↑Panik.
Der ↑NMI, der grundsätzlich nicht an der CPU maskiert werden kann, erhöht noch das Risiko eines möglichen Stapelüberlaufs. Ein entsprechender Handhaber wird immer mit der höchsten Priorität ausgeführt, wobei es es hier eben keine Prioritätsobergrenze gibt: gemeinhin kann jeder Handhaber dieses Unterbrechungstyps in eine indirekte Rekursion verfallen. Sollte ein solcher Handhaber eine Systemfunktion aufrufen, hilft für gewöhnlich nur noch ↑nichtblockierende Synchronisation mit wartefreier ↑Fortschrittsgarantie.
Findet ein ↑Multiprozessor Verwendung, gelten alle vorher genannten Aspekte weiterhin. Darüberhinaus muss dann jedoch die Unterbrechungssperre an einer bestimmten CPU insbesondere auch durch Prozesse der Systemebene von jeder anderen CPU aus verhängt werden können. Für gewöhnlich ist dies nicht ohne weiteres aus der Ferne möglich und benötigt entweder Spezialhardware oder einen ↑PIC, dessen IRQ-Ausgang einer Maskierung unterzogen werden kann. Letzteres bedeutet, dass die in dem PIC typischerweise integrierte ↑Leitweglenkung von Unterbrechungsanforderungen (↑*interrupt* ↑*routing*) komplett ab- und wieder anstellbar von jedem ↑Rechenkern aus ist. Aber auch in dem Fall ist jede Unterbrechungssperre immer noch mit dem grundsätzlichen Problem behaftet, Unterbrechungsanforderungen zu unterbinden, deren jeweilige Behandlung in überhaupt keiner Abhängigkeit zum gegenwärtigen Prozess steht und daher auch nebenläufig stattfinden könnte.
Aus all den genannten Gründen ruft ein Unterbrechungshandhaber erster Stufe eine Systemfunktion daher für gewöhnlich niemals direkt auf, sondern indirekt im Rahmen einer nachgeschalteten Unterbrechungsbehandlung (↑SLIH). Diese wird durch den Handhaber nur ausgelöst, das heißt, zur späteren Ausführung hinterlassen, und durch einen ↑Ausnahmezuteiler zu gegebener Zeit gestartet. Der richtige Zeitpunkt zum Starten solcher zurückgestellten Aufrufe (↑DPC) ist dann gegeben, wenn zum Zeitpunkt der Beendigung einer Unterbrechungsbehandlung kein Handhaber der ersten Stufe mehr aktiv und demzufolge der Laufzeitstapel komplett abgebaut ist.

flüchtiges Register (en.) ↑*volatile register*. Konzept der ↑Aufrufkonvention: der Inhalt eines solchen Prozessorregisters gilt als unbeständig. Vorkehrungen zur Sicherung und Wiederherstellung des Registerinhalts werden im ↑Unterprogramm nicht getroffen, sondern gegebenenfalls im aufrufenden Programm (*caller-saved*). So definiert beispielsweise ↑cdecl für ↑x86 die Register `EAX`, `ECX` und `EDX` als flüchtig, wobei `EAX` den Funktionsrückgabewert speichert.

FMS Abkürzung für „*FORTRAN Monitor System*“. Gilt als erstes wirkliches ↑Betriebssystem, basierend auf Magnetbandgeräte, das automatisch mehr als ein ↑Programm nacheinander im Stapel (*batch*) verarbeiten konnte. Die Programme konnten zudem in ↑FORTRAN formuliert sein und wurden dann vor Ausführung automatisch der ↑Kompilation unterzogen. Erste Installation im Jahr 1955 (IBM 704).

formal parameter (dt.) ↑formaler Parameter.

formaler Parameter (en.) ↑*formal parameter*. Bestandteil der formalen Schnittstelle (Signatur) von einem ↑Unterprogramm. Bezeichner eines Platzhalters zur Übernahme eines zuweisungskompatiblen Werts (↑tatsächlicher Parameter) als Argument eines Unterprogrammaufrufs.

FORTRAN Abkürzung für (en.) „*formula translation*“; Programmiersprache: prozedural, imperativ (1957).

Fortschrittsgarantie (en.) ↑*progress guarantee*. Zusicherung über das Vorankommen eines ↑Prozesses oder Prozesssystems, wobei ein gemeinsames ↑nichtsequentielles Programm die Grundlage bildet und ↑nichtblockierende Synchronisation für die Koordinierung der Prozesse sorgt. Typischerweise werden drei Striktheitsgrade unterschieden:

- **behinderungsfrei** (en. *obstruction-free*) Ein einzelner, in Isolation stattfindender Prozess wird seine ↑Aktion oder ↑Aktionsfolge in einer begrenzten Anzahl von Schritten durchführen und beenden. Der Prozess findet isoliert statt, sofern alle anderen Prozesse, die ihn behindern könnten, zeitweilig (z.B. durch die ↑Ablaufplanung) zurückgestellt sind.
- **sperrfrei** (en. *lock-free*) Jeder Schritt eines Prozesses trägt dazu bei, dass die Ausführung des nichtsequentiellen Programms insgesamt voranschreitet. Damit ist systemweiter Fortschritt garantiert, jedoch können einzelne Prozesse dem ↑Verhungern unterliegen. Umfasst Behinderungsfreiheit.
- **wartefrei** (en. *wait-free*) Die Anzahl der zur Durchführung und Beendigung einer Aktion oder Aktionsfolge auszuführenden Schritte durch einen Prozess ist konstant oder zumindest nach oben begrenzt. Damit ist der Fortschritt jedes einzelnen Prozesses garantiert. Umfasst Sperrfreiheit.

Entsprechend des Striktheitsgrads nimmt nicht selten die Schwierigkeit in Entwurf und Implementierung eines nichtblockierend auf Prozesse wirkenden Koordinierungsverfahrens zu. Für gewöhnlich ist Sperrfreiheit in vielen Fällen für ein ↑Betriebssystem noch vergleichsweise einfach umzusetzen, wohingegen wartefreie Lösungen zuweilen sehr herausfordernd sind und auf Konstruktionshilfen in der Software angewiesen sind.

Fortsetzung Mechanismus zur Übergabe der Kontrolle über den ↑Rechenkern an einen anderen ↑Handlungsstrang. Letzterer ist in dem Fall nicht als ↑Faden implementiert (auch in keiner Variante davon) und besitzt daher keinen eigenen ↑Laufzeitstapel, um *implizit* darauf beim ↑Prozesswechsel den Zustand zur Wiederaufnahme des Programmablaufs sichern zu können. Stattdessen teilen sich alle Handlungsstränge ein und denselben Stapel und sichern diesen Zustand *explizit* in ein ↑Objekt erster Klasse. Dieses Objekt liegt in einer Form vor, die es erlaubt, es als Funktion aufrufen zu können. Beim Aufruf sichert der Handlungsstrang seinen aktuellen Zustand und gibt sich einen neuen Zustand, den er einem zweiten Objekt entnimmt. Soweit es den ↑Prozessorstatus betrifft, ist der Zustand nur definiert durch „beständige“ ↑Prozessorregister (↑nichtflüchtiges Register) und insbesondere den ↑Befehlszähler. Damit wird der Handlungsstrang nicht an die Stelle des Funktionsaufrufes zurückkehren, sondern immer an eine Stelle, die durch den Befehlszähler des wiederhergestellten Zustands bestimmt ist.

Fortsetzungssperre Vorkehrung zur Abwehr des anschließenden Teils einer asynchron gestarteten ↑Aktion oder ↑Aktionsfolge, stellt ↑Synchronisation her. Typisches Beispiel dafür ist die

zeitweilige Aussetzung der weiteren Abarbeitung zurückgestellter ↑Unterbrechungsbehandlungen der zweiten Ebene (↑SLIH). Eine bereits begonnende Behandlung dieser Ebene wird nach Setzen einer solchen Sperre jedoch bis zum Abschluss durchgeführt (↑*run to completion*), die Sperre wirkt erst beim Übergang zum jeweils nächsten anstehenden Behandlungsauftrag.

FPU Abküzung für (en.) *floating point unit*, (dt.) Gleitkommaprozessor.

Fragmentierung Zerstückelung von ↑Hauptspeicher in kleine freie Abschnitte mit zwischengelagerten belegten Abschnitten. Je höher der Zerstückelungsgrad, umso mehr freie Abschnitte liegen vor, umso kleiner sind diese Abschnitte und umso höher ist die Wahrscheinlichkeit, dass die ↑Speicherzuteilung an einen ↑Prozess scheitert.

FreeBSD Variante von ↑BSD, erste Installation 1993 (Intel 80386). *Freie Software*, bei deren Empfang gleichsam die vollen Nutzungsrechte uneingeschränkt entgegengenommen werden.

Freigabekonsistenz Modell der ↑Speicherkonsistenz, für das ein ↑kritischer Abschnitt den Ausgangspunkt bildet. Die Auswirkung jeder innerhalb dieses Abschnitts von einem einzelnen ↑Prozessor auf den ↑Arbeitsspeicher durchgeführte Schreiboperation ist für andere Prozessoren erst nach Verlassen eben dieses Abschnitts sichtbar.

Freispeicherliste Zusammenstellung freier Abschnitte im ↑Hauptspeicher, wobei jeder Abschnitt einen von einem ↑Prozess derzeit nicht genutzten ↑Adressbereich (↑*hole*) entspricht. Die Liste ist in Abhängigkeit vom jeweils gewählten ↑Platzierungsalgorithmus unterschiedlich technisch ausgeprägt, üblich ist jedoch eine *dynamische Datenstruktur*. Darüberhinaus sind zwei Varianten der Abspeicherung dieser Datenstruktur gebräuchlich: getrennt von oder vereint in den durch sie gelisteten freien Abschnitten. Die erste Variante ist zweckmäßig vor dem Hintergrund von ↑Speicherschutz, wenn nämlich der die freien Abschnitte verwaltende Prozess in einem ↑Adressraum residiert, der von dem des einen freien Abschnitt in seinem Adressraum erzeugenden Prozesses getrennt ist (↑Adressraumisolation). Dies ist meist der Fall für das ↑Betriebssystem, das normalerweise die ↑Speicherzuteilung global für jedes ↑Programm im ↑Rechensystem durchführt. Ausnahme davon bildet ein Betriebssystem, dessen Adressraum zugleich ↑realer Adressraum ist (↑*identity mapping*): in diesem Fall können die Knoten der Datenstruktur zur Erfassung aller freien Abschnitte des Hauptspeichers in diesen Abschnitten selbst liegen. Diese Repräsentation der Liste ist darüberhinaus typisch für die Verwaltung der freien Abschnitte in einem ↑Haldenspeicher, da hier freie und belegte Abschnitte grundsätzlich demselben Adressraum zugehören. Sie bedeutet aber auch, dass die Größe eines bei der ↑Speicherzuteilung einem Prozess zugewiesenen Abschnitts mindestens die Größe eines Knotens der Datenstruktur zur Implementierung der Liste freier Hauptspeicherabschnitte haben muss. Für eine einfach verkettete Liste enthält ein solcher Knoten zwei Attribute: die Größe (`unsigned int`: 2 – 8 Byte) des durch ihn repräsentierten freien Abschnitts und die ↑Adresse (`void*`: 4 – 8 Byte) des Folgeknotens, zusammen also 6 bis16 Byte (je nach ↑CPU).

Frist (en.) ↑*deadline*. Bezeichnung für eine ↑Prozessgröße, die den Zeitpunkt festlegt, zu dem die Durchführung einer ↑Aufgabe spätestens beendet sein muss. Ein ↑Termin, der nicht überschritten werden sollte oder darf.

funktionale Hierarchie Gesamtheit von in einer Rangfolge stehenden Ebenen, wobei jede Ebene durch eine bestimmte Menge von Funktionen definiert ist. Der Idee nach stellt eine Ebene in einer solchen Anordnung eine (reale/virtuelle) Maschine für die nächst höhere Ebene zur Verfügung. Im zugehörigen Entwurf wird eine Ebene oberhalb einer anderen Ebene platziert, um zum Ausdruck zu bringen, dass das korrekte Funktionieren ersterer von der Existenz einer korrekten Implementierung der Funktionen letzterer abhängt: dass die obere Ebene die untere „benutzt“.

GAS Abkürzung für (en.) *GNU assembler*, (dt.) ↑GNU ↑Assemblierer.

Gastbetriebssystem ↑Betriebssystem, das einen ↑VMM oder ein ↑Wirtsbetriebssystem benutzt (↑Benutzthierarchie). In reiner Ausführung nimmt das Gastsystem diese Benutzung nicht wahr: jede in diesem System verursachte ↑Ausnahme wird vom jeweiligen Wirt abgefangen (↑*trap*) und in funktionaler Hinsicht „transparent“ behandelt. Demgegenüber kann das Gastsystem in der Realisierung einer eigenen ↑Systemfunktion jedoch auch vom Wirtssystem profitieren, allerdings impliziert dies Änderungen im Gastsystem (unreine Ausführung).

Gatteradresse ↑Adresse, die in einem bestimmten ↑Adressbereich eine geschützte Zone im ↑Arbeitsspeicher von einer ungeschützten Zone trennt. Ursprünglich bildet ein ↑realer Adressraum die Obermenge für diesen Adressbereich — aber auch ein ↑logischer Adressraum oder ↑virtueller Adressraum kann so in zwei Zonen unterteilt sein, in Abhängigkeit von dem durch das ↑Betriebssystem im Zusammenspiel mit einer ↑MMU jeweils implementierte Modell von dem ↑Adressraum für sich selbst und einem ↑Maschinenprogramm (↑*userland*).

GCC Abkürzung für (en.) *GNU Compiler Collection*, (dt.) ↑GNU ↑C ↑Kompilierer.

Gemeinkosten (en.) ↑*overhead*. Unkosten, indirekte Kosten, einer ↑Systemfunktion — aber auch allgemein jeder von einem ↑Rechensystem zu erfüllenden ↑Aufgabe. Zuschlag bezüglich Rechenzeit, Energieverbrauch oder Speicherplatz, der bei Durchführung dieser Aufgabe in Kauf zu nehmen ist, um von der dadurch dann bereitgestellten Funktion zu profitieren.

gemeinsamer Speicher ↑Speicher, der von mehr als einen ↑Prozess zugleich genutzt werden kann. Im Falle von ↑Arbeitsspeicher ist typischerweise zu differenzieren, ob die gemeinsame Nutzung ein ↑Textsegment betrifft (↑*shared code*, ↑*shared library*) oder ein ↑Datensegment, bis hin zu einem einzelnen ↑Speicherwort darin (↑*shared data*), zur Grundlage hat. ↑Text ist zur ↑Laufzeit in der Regel nur lesbar, kann also von den Prozessen nicht verändert werden. Ganz im Gegensatz dazu stehen ↑Daten, die zwischen den Prozessen ausgetauscht werden und dazu eine ↑Aktion oder ↑Aktionsfolge zum Lesen und Schreiben von verschiedenen Prozessen ausgehend zugleich auf ein oder einer Folge von Speicherwörtern stattfinden muss. In dem Fall ist ↑Synchronisation zu erzielen, um Datenkonsistenz sicherzustellen.

Gemeinschaftsbibliothek Bezeichnung für eine ↑Bibliothek, die zu einem Zeitpunkt von mehr als einen ↑Prozess zugleich benutzt wird (↑*shared memory*).

general-purpose operating system (dt.) ↑Universalbetriebssystem.

Gerätedatei Beispiel für eine ↑Pseudodatei, über die einem ↑Prozess im ↑Maschinenprogramm die direkte Interaktion mit dem ↑Gerätetreiber für ein ausgewähltes ↑Peripheriegerät möglich ist. Typisches Merkmal für ein ↑UNIX-artiges ↑Betriebssystem.

Gerätetreiber ↑Unterprogramm in einem ↑Betriebssystem, durch das ein bestimmtes ↑Peripheriegerät verwaltet und bedient wird. Nach außen stellt solch ein Unterprogramm typischerweise eine für eine Klasse von Peripheriegeräten einheitliche Schnittstelle zur Verfügung (*major device*, ↑UNIX) nach innen ist es speziell zugeschnitten auf das jeweilige ↑Exemplar eines Geräts dieser Klasse (*minor device*, ↑UNIX).

Gerechtigkeit Prinzip des Verhaltens von einem ↑Betriebssystem, das jedem ↑Prozess gleichermaßen Fortschritt zusichert. Fairness in der ↑Synchronisation von Prozessen oder der Vergabe von einem oder mehrerer ↑Betriebsmittel an einen Prozess.

geschäftiges Warten Situation, in der ein ↑Prozess durch anhaltendes Abfragen (*polling*) von einem ↑Speicherwort ein bestimmtes ↑Ereignis erwartet. Gegebenenfalls unterbricht sich der Prozess nach jedem Abfrageschritt (re.)

```
void probe(event) {
    while(*event == 0);
}
```

```
void probe(event) {
    while (*event == 0)
        favor(event);
}
```

und signalisiert damit dem ↑Planer, den ↑Prozessor einem anderen Prozess zuteilen zu können. Der wartende Prozess wechselt jedoch nur dann von laufend nach bereit (↑Prozesszustand), wenn es in dem Moment einen bereitgestellten (anderen) Prozess gibt und dieser gemäß ↑Prozesseinplanung keine niedrigere Priorität besitzt. In logischer Hinsicht behält der wartende Prozess den Prozessor, obwohl er dabei selbst keine produktive Arbeit verrichten kann. Behält der Prozess den Prozessor auch in physischer Hinsicht, trägt er selbst zur Verlängerung seiner Wartezeit bei, da kein anderer Prozess den Prozessor zugeteilt bekommt, um so das erwartete Ereignis herbeiführen zu können. Letzteres trifft auch dann zu, wenn die Prozesseinplanung nach einem ↑Zeitteilverfahren arbeitet: dann ist die Länge des dem Prozess zugebilligten Zeitintervalls maßgeblich dafür, wann ihn ein anderer Prozess verdrängt, der dann eventuell das erwartete Ereignis bewirkt.

GID Abkürzung für (en.) *group identifier*, (dt.) ↑Gruppenkennung.

globale Ersetzungsstrategie Art einer ↑Ersetzungsstrategie, bei der ein ↑Umlagerungsmittel für die Ersetzung ausgewählt werden kann, das einem beliebigen ↑Prozessadressraum zugeordnet ist. Anders als die ↑lokale Ersetzungsstrategie, wird die globale Ausführung somit eine ↑Ausnahmesituation in einem „fremden“ ↑Prozess herbeiführen können, wenn dieser nämlich auf ↑Text oder ↑Daten zugreift, die plötzlich nicht mehr im ↑Hauptspeicher liegen — mehr dazu aber in SP2.

GNU Bezeichnung für ein Mehrplatz-/Mehrbenutzerbetriebssystem, ursprünglich, allgemein jedoch bekannt als Sammlung von (freier) Software, Anwendungs- und Dienstprogrammen. Abkürzung für (en.) *GNU is Not Unix*: gleichwohl als ↑UNIX-ähnliches ↑Betriebssystem konzipiert, aber auf ↑Mach basierend (auch als „*GNU Hurd*“ bezeichnet). In Entwicklung seit 1990 (Intel 80386). Programmiert in ↑Assembliersprache (↑GAS) und ↑C.

GPU Abkürzung für (en.) *graphics processing unit*, ↑Grafikprozessor.

Grafikprozessor (en.) ↑*GPU*. Bezeichnung für einen ↑Prozessor, der ursprünglich nur speziell für die Berechnung von Grafiken ausgelegt war. Allerdings hat sich mittlerweile (2016) auch eine Klasse solcher Prozessoren herausgebildet, die allgemein zur ↑Parallelverarbeitung bestimmter rechenintensiver Probleme jenseits von Grafiken nutzbar sind: beispielsweise der Nvidia Tesla mit 5 760 (K80) oder der Intel Xeon Phi mit 72 realen/288 logischen (Knights Landing) ↑Rechenkernen.

Granularität Anzahl von Untergliederungen eines Elements (Duden). Beispielsweise die Byteanzahl pro ↑Speicherwort, Speicherwortanzahl pro ↑Zwischenspeicherzeile, Zwischenspeicherzeilenanzahl pro ↑Seite, Seitenanzahl pro ↑Segment oder Segmentanzahl pro ↑Adressraum, wenn insbesondere verschiedene Ebenen in der ↑Speicherhierarchie zusammenhängend betrachtet werden.

Grenzregister Schutzvorkehrung mit der die ↑Eingrenzung von dem für einen ↑Prozess gültigen ↑Adressbereich bewerkstelligt wird. Spezielle ↑Prozessorregister, die als Paar die untere und obere (d.h., die erste und letzte gültige) ↑Adresse in diesem Bereich festlegen. Der Inhalt des jeweiligen Registers gleicht einer ↑Gatteradresse, das heißt, der Bezugspunkt ist für gewöhnlich ein ↑realer Adressraum. Im Unterschied zum ↑Schutzgatterregister wird der Adressraum jedoch nicht in nur zwei Gebiete (↑Betriebssystem einerseits, ↑Maschinenprogramm andererseits) unterteilt, in denen abwechselnd zu einem Zeitpunkt immer nur ein Prozess stattfinden kann (↑*uniprogramming*). Stattdessen sind in dem Adressraum mehrere eingegrenzte Gebiete möglich, wovon ein Gebiet für das Betriebssystem und die anderen Gebiete jeweils für ein Maschinenprogramm vorgesehen sind (↑*multiprogramming*). Gleichwohl „erbt“ jedes der beiden Register die typischen Merkmale eines Schutzgatterregisters: der Zugriff darauf ist eine privilegierte Operation (↑*privileged mode*), für einen stattfindenden Prozess sind die gespeicherten Gatteradressen fest und ein ↑verschiebender Lader bringt das jeweilige Maschinenprogramm in den ↑Hauptspeicher.

Gruppenkennung Zeichen zur eindeutigen Identifizierung der Gruppe, der eine durch ↑Benutzerkennung autorisierte ↑Entität angehört; für gewöhnlich eine Nummer. Die Kennung wird bei der ↑Anmeldung initial zugeordnet und kann gegebenenfalls im weiteren Verlauf von einem ↑Prozess für sich selbst geändert werden (`setgid(2)`). Der mögliche ↑Kennungswechsel ist hochgradig abhängig vom ↑Betriebssystem.

Halde Aufschüttung von zurzeit nicht erwerbbaren Vorräten an ↑Speicher (in Anlehnung an den Duden). Eine dynamische Datenstruktur, die der Speicherung von Datenelementen unterschiedlicher Anzahl und Größe dient. Jedes dieser Elemente ist ↑Exemplar eines Datentyps.

Haldenspeicher Bezeichnung für einen ↑Speicher, der als Haufen (↑*heap*) organisiert ist; eine Ansammlung von Speicherbereichen im ↑Adressraum von einem ↑Maschinenprogramm, die durch einen ↑Prozess in diesem ↑Programm belegt oder belegbar (frei) sind. Ein ↑dynamischer Speicher, der in erster Linie der Verwaltung dynamischer Datenstrukturen dient. Seine Kapazität ist begrenzt, aber bis zu dieser Grenze variabel.

Haltebefehl Bezeichnung für einen ↑Maschinenbefehl, bei dessen Ausführung die ↑CPU bis zur nächsten ↑Unterbrechungsanforderung angehalten wird. Der Befehl kommt für gewöhnlich zur Ausführung, wenn sonst keine direkte Arbeit zur Erledigung mehr ansteht, das heißt, wenn das System sich im ↑Leerlauf befindet. Typischerweise ist dieser Befehl nur im privilegierten ↑Arbeitsmodus ausführbar. Der Versuch, diesen Befehl unprivilegiert auszuführen, führt zur ↑Ausnahmesituation.

Handhabe (en.) ↑*handle*. Bezeichnung für etwas, was ein auf ein bestimmtes Ziel gerichtetes Vorgehen ermöglicht, erlaubt (Duden). Eine opake ↑Referenz zu einer ↑Entität.

Handhaber (en.) ↑*handler*. Bezeichnung für ein ↑Unterprogramm zur Bedienung oder Steuerung eines Geräts (↑Peripherie) oder Instruments (Hardware, Software), aber auch zur Behandlung eines bestimmten ↑Ereignisses, etwa der Umgang mit einer ↑Ausnahme (↑*trap*, ↑*interrupt*).

handle (dt.) ↑Handhabe.

handler (dt.) ↑Handhaber (dem dt. Patentwesen entnommene fachbegriffliche Übersetzung).

Handlungsstrang ↑Aktionsfolge, die der Bearbeitung einer komplexen und zusammenhängen Aufgabe entspricht. Mehrere Handlungsstränge können ein und dieselbe Aktionsfolge gemeinsam haben (Exemplare desselben Typs), nur in Teilen der Aktionsfolge überschneiden (Exemplare verschiedener Typen, aber mit wenigstens einem gemeinsamen ↑Unterprogramm) oder komplett verschiedene Aktionsfolgen ausmachen (Exemplare verschiedener Typen).

hard link (dt.) ↑Verknüpfung.

Hauptplatine Bezeichnung für eine ↑Trägerleiterplatte, auf der die zentralen Bauteile von einem ↑Rechner platziert sind. Die einzelnen (oftmals in Sockeln steckenden) Bauteile sind Halbleiterbausteine (*chip*) für ↑CPU, ↑RAM und ↑ROM (jew. in den verschiedensten Ausprägungen), zum Anschluss von ↑Peripherie, sowie Steckplätze für Erweiterungskarten unterschiedlicher Art. Eine mögliche Implementierung der ↑Zentraleinheit eines Rechners..

Hauptprogramm Bezeichnung für ein ↑Programm, das in logischer Hinsicht von keinem anderen Programm aufgerufen wird: es beginnt seine Ausführung, indem es vom ↑Betriebssystem gestartet wird. Technisch geschieht dies in bestimmten Fällen sehr wohl durch einen Aufruf, beispielsweise einen in der Form `main(argc, argv, envp)` bei ↑C. Hier erfolgt der Aufruf durch eine spezielle ↑Aktionsfolge zur Inbetriebnahme (*startup*) des Programms überhaupt, und zwar im Namen von der vom Betriebssystem vorher dazu eingerichteten ↑Prozessinkarnation. Dieser Aufruf bewirkt auch, dass `main()` sehr wohl zurückkehren kann. So wird `main()` letztlich von einer speziellen ↑Mantelprozedur aufgerufen, die zudem für die gegebenenfalls benötigte Parameterversorgung (`argc`, `argv`, `envp`) sorgt. Nach Rückkehr aus

`main()` zu dieser Mantelprozedur verlässt der ↑Prozess per ↑Systemaufruf (`_exit(2)`) das ↑Maschinenprogramm, betritt das Betriebssystem und terminiert dort.

Hauptrechner Rechenanlage, die von einer anderen (für gewöhnlich kleineren) Rechenanlage vorbereitete Aufgaben entgegennimmt, durchführt und nach Erledigung an diese (ggf. sogar eine andere, kleinere Rechenanlage) zur Nachbearbeitung abgibt. Historisch auch als Inbegriff für die ↑Zentraleinheit, die in ↑Stapelverarbeitung die über ↑Satellitenrechner bereitgestellten Aufträge ausführt (↑*remote operation*). An einer solchen Anlage ist, neben der ↑Systemkonsole, für gewöhnlich nur ein schnelles ↑Peripheriegerät angeschlossen, über das die ↑automatisierte Rechnerbestückung und Entsorgung (d.h., Ausgabe der Berechnungsergebnisse) geschieht. Dieses Gerät war früher (ab Ende 1950) ein Bandlaufwerk, zwischenzeitlich (ab Mitte 1960) ein Wechselplattenlaufwerk und ist heute (2016) zumeist auch ein Steuergerät zur Hochgeschwindigkeitskommunikation.

Hauptspeicher Bezeichnung für den ↑Speicher in dem ↑Programm liegen muss, damit es von der ↑CPU ausgeführt werden kann. Die für die Ausführung erforderlichen Bestände von ↑Text und ↑Daten des Programms liegen flüchtig (temporär: ↑DRAM, ↑SRAM) oder nichtflüchtig (permanent: ↑ROM, ↑PROM; semi-permanent: ↑EPROM, ↑EEPROM, ↑NVRAM) vor. Als ↑Direktzugriffsspeicher (↑RAM) organisiert, im Allgemeinen als flüchtig ausgelegt verstanden in Form von Schreib-lese-Speicher.

Hauptspeicherfehlzugriff Fehlzugriff auf eine im ↑Hauptspeicher gewähnte Informationseinheit (↑Maschinenbefehl, ↑Daten). Die für den Zugriff von der ↑CPU im ↑Abruf- und Ausführungszyklus applizierte ↑virtuelle Adresse ist der ↑Teilinterpretation durch das ↑Betriebssystem zu unterziehen. Der Zyklus wird unterbrochen, abgefangen (↑*trap*) und nach erfolgter Behandlung des Fehlzugriffs im Betriebssystem später von der CPU wieder aufgenommen (↑*rerun*). Typisch für einen ↑Seitenfehler — jedoch gibt ein ↑virtueller Adressraum, der Voraussetzung für die Erkennung eines solchen Fehlzugriffs ist, dem Betriebssystem damit einen allgemeinen Mechanismus in die Hand, um gezielt die Kontrolle zurückzugewinnen, wenn ein ↑Prozess einen bestimmten ↑Adressbereich durchstreift. Dazu programmiert das Betriebssystem die ↑MMU, beim Zugriffsversuch auf eine bestimmte ↑Seite oder ein bestimmtes ↑Segment den Prozess zu unterbrechen. Eine übliche Maßnahme ist es, die Seite/das Segment nicht vorhanden erscheinen zu lassen (↑*present bit*). Unterstützt die MMU solch eine Maßnahme nicht direkt, lässt sich das gewünschte Verhalten gegebenenfalls durch eine Behelfslösung herbeiführen, indem im ↑Seitendeskriptor/↑Segmentdeskriptor gespeicherte Attribute zweckentfremdet werden (z.B. alle Zugriffsrechte löschen).

Hauptsteuerprogramm ↑Programm im ↑Betriebssystemkern, das alle wesentlichen Funktionen zur Mehrfachnutzung der zugrunde liegenden (realen/virtuellen) Maschine entsprechend der gewünschten ↑Betriebsart umfasst. Mindestens enthalten im Funktionsumfang sind ↑Prozesseinplanung und ↑Ausnahmebehandlung. Eine typische weitere Funktion ist der ↑Speicherschutz, sofern die bereitzustellende Betriebsart dies erfordert.

heap (dt.) ↑Halde.

heavy-weight process (dt.) ↑schwergewichtiger Prozess.

Heimatverzeichnis Bezeichnung für ein ↑Verzeichnis, das den ↑Namenskontext für einen ↑Prozess direkt nach erfolgter Systemanmeldung (↑*login*) bildet. Es ist gleichsam das ↑Arbeitsverzeichnis, in dem der Prozess seine Arbeit aufnimmt. Dieses Verzeichnis ist (unter ↑UNIX) auch häufig die Stelle, an der ein ↑Dienstprogramm unterstützende ↑Daten platziert. Diese in Form einer ↑Datei indirekt in einem weiteren ↑Verzeichnis gespeicherten Daten werden namentlich durch einen Punkt angeführt.

Hertz Maßeinheit der Frequenz, Zeichen: Hz (Duden). Anzahl sich regelmäßig wiederholender Vorgänge pro Sekunde.

Heterogenität Verschiedenartigkeit, Ungleichartigkeit, Uneinheitlichkeit im Aufbau, in der Zusammensetzung (Duden); ein ↑Architekturmerkmal. Gegenteil von ↑Homogenität.

Heuristik Methode einer Anleitung für ein analytisches Verfahren, das auf Grundlage unvollständiger Informationen und in endlicher Zeit zu brauchbaren Ergebnissen kommt.

hexspeak Bezeichnung für eine auf Hexadezimalziffern beruhende Schreibweise, um vor allem einprägsame „magische Zahlen“ zu konstruieren.

hierarchische Struktur Anordnung der Teile eines Ganzen, die durch eine Methode der Aufteilung eines Systems in seine Einzelbestandteile und die Art der Relation zwischen Teilepaaren bestimmt ist. Strukturen sind hierarchisch, wenn eine solche Relation $R(\alpha, \beta)$ Ebenen entstehen lässt. Dabei ist Ebene_0 eine Menge von Teilen α, so dass es kein β gibt mit $R(\alpha, \beta)$. Ebene_i, für $i > 0$, ist Menge von Teilen α, so dass gilt: es existiert ein β auf Ebene_{i-1} mit $R(\alpha, \beta)$ und falls $R(\alpha, \gamma)$, dann liegt γ auf Ebene_{i-1}. Dabei sind folgende Arten von R geläufig: „benutzt“, wenn das korrekte Funktionieren von α die Existenz wenigstens einer korrekt funktionierenden Implementierung von β voraussetzt (↑Benutzthierarchie, ↑funktionale Hierarchie); „ruft“, wenn α einen Aufruf an ↑Unterprogramm β enthält (Aufrufhierarchie); „beauftragt“, wenn α einen Auftrag an β abgibt und beide Vorgänge dabei unabhängig von der relativen Geschwindigkeit des jeweils anderen Vorgangs operieren (Prozesshierarchie).

hierarchischer Namensraum ↑Namensraum, der eine ↑hierarchische Struktur aufweist. Die einer solchen Struktur zugrundeliegende, Ebenen entstehen lassende, Relation $R(\alpha, \beta)$ zwischen Teilepaaren α und β bedeutet hier „definiert“: der ↑Namenskontext α bestimmt den Bezugsrahmen, in dem ↑Name β eindeutig ist. Um eine aus vielen Ebenen bestehende Struktur zu bilden, kann der in einem Namenskontext verzeichnete Name seinerseits einen Namenskontext auf tiefer gelegener Ebene bezeichnen. Die sich dadurch ergebende Struktur ist baumartig (↑*file tree*).

Hintergrundrauschen (en.) ↑*background noise*, in anderen Fällen auch als ↑Umgebungsrauschen bezeichnet. Entspricht typischerweise der Gesamtheit aller messbaren Störgrößen beziehungsweise ↑Gemeinkosten in einem ↑Betriebssystem, die im Hintergrund von einem normalen ↑Programmablauf anfallen. Beispiel dafür ist ein ↑Dämon, der in regelmäßigen Abständen bestimmte Verwaltungsaufgaben durchführt (↑*spooler*, ↑*pager*). Ebenso ein im Betriebssystem mitlaufender ↑Planer, der im ↑Zeitteilverfahren arbeitet und gegebenenfalls die ↑Verdrängung von einem ↑Prozess bewirkt. Ferner die durch einen ↑Zeitgeber regelmäßig ausgelöste ↑Systemfunktion zur Bestimmung der ↑Arbeitsmenge eines Prozesses. Jeder weitere Fall einer ↑Unterbrechungsbehandlung macht eine solche Störgröße aus. Die plötzliche Ersetzung einer ↑Seite oder ↑Zwischenspeicherzeile kann einen Fehlzugriff durch einen Prozess auslösen, der die referenzierte ↑Entität eben noch direkt zugreifbar wähnte: die Behandlung des Fehlzugriffs verzögert den Prozess ungewollt. Je nach der ↑Betriebsart fallen Anzahl, Häufigkeit und Höhe dieser Störgrößen und Unkosten verschieden aus. Jedoch lassen sich die verschiedenen Betriebsarten nicht so einfach in Bezug auf ihr „Rauschen“ klassifizieren. Hier müssen im Einzelfall Messungen und Analysen der ↑Laufzeit vorgenommen werden, wenn die Parameter relevant für den Ablauf von einem ↑Maschinenprogramm sind.

Hintergrundspeicher ↑Speicher zur Auslagerung von Programmen und Daten; auch Sekundärspeicher. Peripherer Speicher, der indirekt über Ein-/Ausgabeoperationen durch das ↑Betriebssystem bedient wird.

hole (dt.) ↑Loch, Lücke, Leerstelle.

hole list (dt.) ↑Löcherliste.

home directory (dt.) ↑Heimatverzeichnis.

Homogenität Gleichartigkeit, Gleichmäßigkeit im Aufbau, in der Zusammensetzung (in Anlehnung an den Duden); ein ↑Architekturmerkmal. Gegenteil von ↑Heterogenität.

HRRN Abkürzung für (en.) *highest response ratio next*. Bezeichnung für eine Strategie zur ↑Planung der Bearbeitung von ↑Aufgaben durch einen ↑Prozessor in einem *Vorrangverfahren*, das Aufgaben mit kürzerer ↑Bedienzeit bevorzugt, dabei aber die ↑Wartezeiten der die Aufgaben jeweils bearbeitenden ↑Prozesse berücksichtigt (↑*ageing*).
Grundlage für die Auswahl der als nächstes zu bearbeitenden Aufgabe ist ein Verhältniswert R (*ratio*), $1 \leq R < \infty$, der wie folgt definiert ist:

$$R = \frac{t_{wait} + t_{burst}}{t_{burst}} = 1 + \frac{t_{wait}}{t_{burst}}.$$

Dabei ist t_{burst} die gemäß ↑SPN abgeschätzte Bedienzeit als Dauer des nächsten ↑Rechenstoßes des betreffenden Prozesses. Die Wartezeit t_{wait} ist, ausgehend von einem bestimmten *Beobachtungszeitpunkt*, der Zeitunterschied zur ↑Ankunftszeit einer Aufgabe, nämlich wenn der dieser Aufgabe entsprechende Prozess auf die ↑Bereitliste kommt:

$$t_{wait} = |p_{observe} - p_{arrive}|.$$

Zur Ankunftszeit gilt für die betreffende Aufgabe $t_{wait} = 0$. Damit wird der dieser Aufgabe entsprechende Prozess mit $R = 1$ auf die Bereitliste gesetzt. Je größer dieser Wert ist, umso stärker ist der Vorrang des betreffenden Prozesses gegenüber anderen Prozessen: die Bereitliste ist nach absteigenden Werten von R sortiert. Ein bereitzustellender Prozess wird daher immer ans Ende der Bereitliste gelangen, da jeder andere auf dieser Liste bereits platzierte Prozess eine Wartezeit $t_{wait} > 0$ besitzt und damit $R > 1$ gilt. Das wiederum bedeutet eine ↑Aktionsfolge zum Einfügen, die bei geeigneter Auslegung der Bereitliste (↑queue) mit konstantem Zeitaufwand geschehen kann.
Mit voranschreitender Verweildauer auf dieser Liste nimmt auch die Wartezeit der Aufgabe eines bereitgestellten Prozesses zu, die Bedienzeit dieser Aufgabe bleibt jedoch unverändert. Als Folge vergrößert sich R für jeden dieser Prozesse. Werden fortlaufend weitere Prozesse bereitgestellt, gelangen diese zunächst ans Listenende. In Abhängigkeit von ihrer Bedienzeit können diese Prozesse zu einem späteren Zeitpunkt, während sie warten müssen, nach und nach jedoch vordere Plätze in der Liste erreichen und damit von entsprechend stärkerem Vorrang profitieren. Dies ist typisch für Prozesse mit eher kurzer Bedien- und immer länger werdender Wartezeit.
Für den Beobachtungszeitpunkt, um R für jeden Prozess auf der Bereitliste zu berechnen, gibt es zwei Optionen: dieser Zeitpunkt ist entweder das Moment der ↑Einlastung eines Prozesses oder ein in regelmäßigen Zeitabständen auftretendes ↑Ereignis. In beiden Fällen ist die gesamte Bereitliste zu durchlaufen und gegebenenfalls umzusortieren. Folglich hat die erste Option eine vergleichsweise hohe ↑Einlastungslatenz zur Konsequenz — deren Höhe zudem von der jeweiligen Listenlänge abhängt, damit nicht nur variiert, sondern für gewöhnlich auch nicht vorhergesagt werden kann. Im anderen Fall hängt die Vorgehensweise von der zugrunde liegenden ↑Rechnerarchitektur und deren Nutzung durch das ↑Betriebssystem ab:

Multiplexbetrieb Ist ein ↑Uniprozessor gegeben, wird die Berechnung typischerweise durch einen ↑Zeitgeber ausgelöst und im Rahmen der entsprechenden ↑Unterbrechungsbehandlung durchgeführt. Da auch in dem Fall sowohl der Durchlauf als auch die Neuordnung der Bereitliste zeitlich variiert, muss die Berechnung durch einen ↑Unterbrechungshandhaber zweiter Stufe (↑SLIH) erfolgen, um den jeweils unterbrochenen Prozess nicht zwingend unbestimmt lang zu verzögern. Die dann erforderlichen regelmäßigen ↑Unterbrechungsanforderungen sorgen jedoch für zusätzliches ↑Hintergrundrauschen.

Parallelbetrieb Liegt ein ↑Multiprozessor vor, bietet sich als zusätzliche Variante die Auslagerung dieser Berechnung auf einen dedizierten und nicht für die ein- oder umzuplanenden Prozesse bestimmten ↑Rechenkern an. Die benötigten zyklischen Unterbrechungsanforderungen gehen alle an diesen Rechenkern und verzögern daher die zu planenden Prozesse nicht.

Der Vorteil dieser jeweils unterbrechungsgesteuerten und im Hintergrund stattfindenden Berechnung besteht darin, dass die (im Vordergrund von jedem Prozess selbst zu tätigende) Auswahl des Prozesses, der als nächstes der Einlastung zuzuführen ist, zeitlich begrenzt und sogar in (nahezu) konstantem Zeitaufwand geschieht: die Einlastungslatenz wäre minimal und begrenzt. Dieses Argument ist vielfach entscheidend, weshalb das Verfahren in der Umsetzung typischerweise auch dieser zweiten Option (Hintergrundausführung) folgt.
Mit SPN gemeinsame Merkmale sind (a) die ↑probabilistische Planung, da die jeweiligen Bearbeitungszeiten für gewöhnlich nur abgeschätzt werden können (↑*time series analysis*) und (b) die nicht ↑präemptive Planung: eintreffende Prozesse bewirken den Prozessorentzug ebenso wenig wie ein Prozess, der während seiner Wartezeit den Listenanfang erreicht haben sollte. Im Unterschied zu SPN sind die Prozesse nicht von ↑Verhungern betroffen, weil jeder neu bereitgestellte Prozess zunächst hinten auf die Bereitliste kommt und zum „Überholen" früher eingetroffener Prozesse immer erst eine bestimmte Wartezeit akkumulieren muss.

Exkurs Als Erweiterung zu SPN fallen im Wesentlichen in zweierlei Hinsicht Änderungen an. Zum einen ist das Sortierkriterium nicht mehr bloß die Länge t_{burst} des zu erwartenden nächsten Rechenstoßes eines Prozesses, sondern der Verhältniswert R zur Prozesswartezeit t_{wait}. Die Bereitstellung eines Prozesses wird daraufhin gemäß ↑FCFS geschehen können, nur dass die Ankunftszeit des Prozesses festzuhalten ist, um damit zum Beobachtungszeitpunkt seine jeweilige Wartezeit bestimmen zu können. Zum anderen ist ein Unterbrechungshandhaber zweiter Stufe erforderlich, der in regelmäßigen Zeitabständen die Bereitliste durchläuft und gegebenenfalls umstellt. Für den ersten Aspekt ergibt sich folgende Bereitstellungsoperation (vgl. SPN, S. 149):

```
void ready(process_t *task) {                /* schedule according to HRRN */
    if (task != being(ONESELF))                              /* not myself? */
        task->time.guess = guess(task);      /* yes, estimate burst length */

    state(&task->mood, READY);                 /* set process ready to run */

    ticket(&task->line, 1);                        /* define initial ratio */
    task->time.ready = time();                      /* remember ready time */
    enqueue(labor(), &task->line.next);           /* append to ready list */
}
```

Der auf zweiter Stufe nachgeschaltete Unterbrechungshandhaber bestimmt für jeden Prozess auf der Bereitliste den Verhältniswert R und ordnet den betreffenden Prozess entsprechend neu ein. Nachfolgende Prozedur (aufgerufen im bereits für SPN benötigten ↑FLIH zur Messung der Prozessverzögerungen, vgl. S. 146) skizziert den Einstieg in die dafür benötigten Aktionsfolgen:

```
void operate(train_t *this) {
    static dpc_t slih = { 0, hrrn, 0 };                 /* SLIH descriptor */
    defer(annex(), &slih);                                 /* release SLIH */
}
```

Diese Operation des Unterbrechungshandhabers erster Stufe löst lediglich den entsprechenden Handhaber zweiter Stufe aus, der dann die Bereitliste durchläuft und für jeden darauf verzeichneten Prozess den für ihn geltenden Wert für R berechnet. Dieser Durchlauf sei wie folgt skizziert:

```
void hrrn(data_t none) {           /* update ready list according to HRRN */
    process_t *task = forge(ahead(labor()));           /* read first task */
    if (task) {                                    /* any task ready to run? */
        queue_t list = {0};                  /* newly assembled task queue */
        chain_t *next;                      /* successor task on ready list */
        time_t time = time();      /* reference point for observation time */
        do {                                      /* walk through ready list */
            next = chain(&task->line.next);          /* remember successor */
            ticket(&task->line, judge(task, time));        /* assign ratio */
            enlist(&list.head, &task->line);       /* add task to new list */
            if (chain(&task->line.next) == 0)    /* trailing list element? */
                list.tail = &task->line.next;  /* yes, adjust tail pointer */
            task = forge(next);          /* make it a process/task pointer */
        } while (task);              /* update next process, if applicable */
        *labor() = list;                                /* update ready list */
    }
}
```

Bevor der Listendurchlauf startet, wird der für alle Listeneinträge (`task`) identische Beobachtungszeitpunkt (`time`) festgehalten. Die für einen gelisteten Prozess verstrichene Wartezeit ergibt sich relativ zu diesem Zeitpunkt. Verrechnet mit der jeweils erwarteten Rechenstoßlänge wird sodann jeder dieser Prozesse neu bewertet (`judge`) und entsprechend seiner Bewertung in die neue Bereitliste (`list`) einsortiert (`enlist`). Wenn ein so behandelter Prozesseintrag ans Ende dieser Liste gelangt ist, wird der Schlangenendezeiger (`list.tail`) entsprechend aktualisiert. Nach erfolgtem Listendurchlauf geschieht die Aktualisierung der globalen Bereitliste (`labor`), womit schließlich die ↑Umplanung stattgefunden hat.
Die Neubewertung eines jeden Prozesses folgt der oben vorgestellten Berechnung des Verhältniswerts R. Die entsprechende Funktion dazu ergibt sich wie folgt:

```
rank_t judge(process_t *task, time_t time) {    /* value according to HRRN */
    time_t wait = llabs(time - task->time.ready);  /* settle waiting time */
    return (wait + task->time.guess) / task->time.guess; /* compute ratio */
}
```

Der so berechnete Verhältniswert bestimmt letztlich den Rank, den der Prozess auf der Bereitliste erhält. Diesem Rank entsprechend wird der Prozess einsortiert (`enlist`), wozu das Verkettungsglied im betreffenden Prozesskontrollblock zuvor noch zu definieren ist:

```
rank_t ticket(entry_t *item, rank_t rank) {      /* fetch and set sort key */
    rank_t aux = item->rank;                          /* pluck old rank */
    item->rank = rank;                               /* define new rank */
    return aux;                                     /* deliver old rank */
}
```

Abschließend sei nochmals darauf hingewiesen, dass die dieses Verfahren mit sich bringende Umplanung von Prozessen für gewöhnlich im Hintergrund laufend stattfindet, nämlich im Rahmen einer zyklischen Unterbrechungsbehandlung. Auch wenn, wie hier gezeigt, dafür ein Unterbrechungshandhaber zweiter Stufe zum Einsatz kommt, geschieht die Umplanung immer noch im Namen des jeweils unterbrochenen Prozesses. Von der damit einhergehenden Verzögerung kann jeder beliebige Prozess betroffen sein. Daher sollten zur Implementierung der Bereitliste für das hier erläuterte Verfahren Datenstrukturen verwendet werden, die in Algorithmen mit geringem und begrenztem Laufzeitaufwand für den Listendurchlauf resultieren. Die einfachheitshalber hier gewählte Schlange ist mit Bedacht zu verwenden und bietet sich nur bei einer eher kleinen Anzahl von Prozessen an.

hybrider Adressraum Bezeichnung für einen ↑Adressraum, der sowohl ↑realer Adressraum, ↑logischer Adressraum als auch ↑virtueller Adressraum ist. Er vereint alle Eigenschaften eines logischen/virtuellen Adressraums mit der besonderen (eher ungewöhnlichen) Eigenschaft, einem ↑Prozess den kompletten ↑Hauptspeicher direkt zugänglich zu machen. Der Prozess hat damit direkten Zugriff auf alle freien, aber auch auf alle von allen anderen Prozessen belegten Abschnitte. Ein solcher Adressraum ist (im Falle von ↑Mehrprogrammbetrieb) nur einem ↑Betriebssystem zuzubilligen — jedoch auch dann bedenklich, da jede ↑Aktion im Betriebssystem direkten Zugriff auf jedes ↑Speicherwort und damit jeden beliebigen Abschnitt von ↑Text und ↑Daten im Hauptspeicher hat.

Hypervisor ↑Programm, das eine ↑virtuelle Maschine steuert und überwacht; entspricht einem ↑VMM, Typ I oder II. Üblichweise stellt dieses Programm mehrere Exemplare desselben Bautyps einer virtuellen Machine zur Verfügung und verwaltet diese entsprechend (Mehrfachnutzung). Legendarisch als Steuerprogramm für das „Leitsystem“ (↑*supervisor*) von ↑VM/370 bestimmt, woraus sich die Namensgebung ableitet.

I/O (dt.) ↑E/A.

I/O bound (dt.) ↑Ein-/Ausgabe gebunden. Bezeichnung für eine ein-/ausgabeintensive ↑Aufgabe, einen interaktiven ↑Prozess. Gegenteil von ↑*CPU bound*.

I/O burst (dt.) ↑Ein-/Ausgabestoß.

i286 Intel 80286, 16-Bit Mikroprozessor (1982). Führte den Schutzmodus (*protected mode*) für ↑x86-Prozessoren ein, indem ein ↑segmentierter Adressraum den ↑Speicherschutz ermöglichte.

i386 Intel 80386, 32-Bit Mikroprozessor (1985). Nachfolger des ↑i286, wobei der Schutzmodus wahlweise als ↑segmentierter Adressraum oder ↑seitennummerierter Adressraum ausgeführt ist. Darüberhinaus kann die ↑Segmentadressierungseinheit mit der ↑Seitenadressierungseinheit kombiniert werden, um damit ↑segmentierte Seitenadressierung zu ermöglichen.

IBM 709 Großrechner (1958): Röhrentechnik, 36-Bit ↑Maschinenwort, ↑überlappte Ein-/Ausgabe, Ein-/Ausgabekanäle (↑Kanalprogramm).

IBM System/360 Großrechner (1964): 32-Bit ↑Maschinenwort, 24-Bit ↑Adressbreite, byteorientierter ↑Hauptspeicher (8-Bit ↑Byte), mikroprogrammierte ↑CPU, ↑EBCDIC Zeichensatz.

IBM z Systems Großrechner (2008): 64-Bit System, bis zu 141 Haupt- und 640 Hilfsprozessoren, „*zero downtime*“. Volle Rückwärtskompatibilität bis ↑IBM System/360.

identity mapping (dt.) identische Abbildung: ↑hybrider Adressraum.

idle (dt.) ↑Leerlauf.

idle process (dt.) ↑Leerlaufprozess.

IDT Abkürzung für (en.) ↑*interrupt descriptor table*.

index node (dt.) ↑Indexknoten.

Indexknoten ↑Informationsstruktur in einem ↑Dateisystem; ↑Deskriptor einer auf einem ↑Datenträger liegender ↑Datei. Typische Attribute dieser Struktur sind (↑UNIX): Eigentümer (↑UID), Gruppenzugehörigkeit (↑GID), Rechte (lesen, schreiben, ausführen: für Eigentümer, Gruppe und die Welt), Zeitstempel (letzter Zugriff, letzte Änderung), Anzahl der Verweise (↑*hard link*) und Typ. Letzteres Attribut klassifiziert eine Datei als: ↑Verzeichnis, ↑symbolische Verknüpfung, Kommunkationskanal (*pipe*, *named pipe*), Sockel (*socket*) zur Interprozesskommunikation, Gerätedatei, ↑Pseudodatei oder reguläre Datei. Im Falle einer regulären Datei führt der Deskriptor Buch über die Dateigröße (Vielfaches von ↑Byte) und jeden ↑Block (in Form der jeweiligen ↑Blocknummer), der zur Speicherung der ↑Nutzdaten verwendet wird.

Indexknotennummer Zahl, die einen ↑Indexknoten im ↑Dateisystem kennzeichnet. Mit Hilfe dieser Zahl (↑numerische Adresse) wird der zu verwendende Eintrag in der ↑Indexknotentabelle selektiert, sie ist technisch ein Tabellenindex und als ↑Adresse nur gültig in Bezug auf den ↑Namensraum dieses einen Dateisystems.

Indexknotentabelle Zusammenstellung von ↑Indexknoten, listenförmig, pro ↑Dateisystem. Ihre Größe wird bei Formatierung des Dateisystems festgelegt. Jeder Eintrag ist der ↑Deskriptor einer ↑Datei auf dem ↑Datenträger. Die Größe der Tabelle bestimmt damit die maximale Anzahl von Dateien pro Dateisystem.

Informationsstruktur Tripel von Typen von Informationsträgern, ihrer Repräsentation und der Menge der auf sie anwendbaren Operationen. Auch zu verstehen als Menge von abstrakten Datentypen einer (realen/virtuellen) Maschine, ein Aspekt in ihrem ↑Operationsprinzip.

inode Abkürzung für (en.) *index node*, (dt.) ↑Indexknoten.

inode number (dt.) ↑Indexknotennummer.

inode table (dt.) ↑Indexknotentabelle.

instance (dt.) Beispiel, ↑Exemplar.

instruction cycle (dt.) ↑Befehlszyklus.

instruction set (dt.) ↑Befehlssatz.

integrierter Befehl Bezeichnung für einen ↑Auftrag, den der in einem ↑Kommandointerpreter jeweils stattfindende ↑Prozess von selbst und direkt durchführt. Technisch ausgelegt zumeist als ↑Unterprogramm.

Integrität Unverletzlichkeit von Programmtext und -daten, die korrekte Funktionsweise eines Systems gemäß Spezifikation.

Intel 64 Prozessorarchitektur, 64-Bit, ursprüngliche Spezifikation als ↑AMD 64. Zwischenzeitlich zunächst bezeichnet als IA32e, dann als EM64T (*extended memory 64 technology*). Erste Produktfreigabe in 2004 (Intel Xeon „Nocona“).

Interferenz Überlagerung beim Zusammentreffen zweier oder mehrerer Vorgänge, die sich dadurch möglicherweise gegenseitig beeinflussen. Jeder Vorgang entspricht dabei einem ↑Prozess, der wenigstens ein ↑Betriebsmittel mit einen anderen Prozess gemeinsam hat. Teilen sich zwei Prozesse beispielsweise die ↑CPU im Zeitmultiplexverfahren (↑partielle Virtualisierung), kann der eine Prozess durch den anderen verdrängt werden. Der verdrängende Prozess überlagert den verdrängten Prozess und verzögert diesen dadurch. Darf das Betriebsmittel nur exklusiv genutzt werden, ist im Falle mehrerer Prozesse, die gleichzeitig darauf zugreifen, ↑Synchronisation explizit sicherzustellen. Ein Prozess, der ein solches Betriebsmittel hält, beeinflusst andere Prozesse, die dieses Betriebsmittel gleichzeitig beanspruchen und sich synchronisieren müssen, wodurch letztere sich gegenseitig selbst beeinflussen. Spezielle Ausprägung davon ist ein ↑kritischer Abschnitt. Grundsätzlich kann ↑Synchronisierung die ↑Ablaufplanung überlagern, was Störungen im ↑Ablaufplan zur Folge haben kann; genauer: ↑blockierende Synchronisation, die eine Prozessreihenfolge festlegt und dadurch der Entscheidung der Ablaufplanung möglicherweise entgegenwirkt.

interner Verschnitt ↑Verschnitt innerhalb von einem ↑Speicherstück. Normalerweise bezogen auf eine ↑Seite, betrifft aber jeden Typ ↑Speicherbereich, dessen Größe echtes Vielfaches der Größe eines ↑Byte ist und am Ende ein ↑Loch aufweist. Typisch ist ein solcher Verschnitt wenn ein ↑seitennummerierter Adressraum die Grundlage für die Verwaltung von ↑Arbeitsspeicher bildet. In dem Fall wird die ↑Speicherzuteilung einem ↑Prozess immer einen Abschnitt zuteilen, dessen Größe Vielfaches der Größe einer Seite ist. Damit kann am Ende der

letzten/einzigen Seite des zugeteilten Abschnitts einer linearen Folge von Seiten ein Loch der Größe $hole = size \mod sizeof(page)$ entstehen, wobei *size* die Größe des angeforderten Speicherbereichs ist. Alle Seiten dieses Abschnitts werden in Folge in den ↑Adressraum des Prozesses platziert, der Prozess erhält damit auch mehr als angefordert zugeteilt. In logischer Hinsicht dürfte dieser Prozess nicht auf den dem Loch entsprechenden Seitenabschnitt zugreifen. Physisch kann er an einen solchen Zugriff allerdings nicht gehindert werden — obwohl er sich in dem Fall fehlerhaft verhalten würde, da er das Loch nämlich nicht kennen dürfte: mit der ↑MMU kann dieser Zugriffsfehler nicht festgestellt werden.

Interpretation Vorgang der Deutung und bedingten Ausführung der Anweisungen in einem ↑Programm durch einen ↑Interpreter. Ausgeführt werden nur gültige Anweisungen. Ob und unter welchen Bedingungen eine Anweisung gültig ist und was eine ungültige Anweisung zur Folge hat, legt das ↑Operationsprinzip der (realen/virtuellen) Maschine fest, die der Interpreter implementiert. Der ↑Befehlssatz dieser Maschine definiert die Menge der allgemein gültigen Anweisungen (d.h., Befehle), aus denen die Programme für diese Maschine formuliert werden. Ungültige Anweisungen werden von der Maschine ignoriert oder bewirken eine ↑Ausnahme, die von der Maschine erhoben wird. Eine solche Ausnahme kann die ↑partielle Interpretation der betroffenen Anweisung bedeuten.

Interpreter Soft-, Firm- oder Hardwareprozessor, der die Anweisungen von einem ↑Programm deutet und direkt ausführt. Gegebenenfalls erfolgt zuvor eine ↑Vorübersetzung durch einen ↑Kompilierer, um die ↑semantische Lücke zwischen der Quellsprache, in der das Programm formuliert ist, und der ↑Interpretersprache, in der das zu deutende Programm vorliegen muss, zu verringern und dadurch die Interpretation zu beschleunigen.

Interpretersprache Bezeichnung der Programmiersprache für ein ↑Programm, dessen Ausführung durch ↑Interpretation in Software (↑CSIM) bestimmt ist. Beispiele dafür sind BASIC, Forth, Perl, Python oder PHP. Verwandte Form ist die ↑Skriptsprache.

Interpretersystem Prinzip nach dem die ↑Interpretation von einem ↑Programm geschieht. Normalerweise erfolgt diese *total*, also vollständig durch einen ↑Interpreter, der die (reale/virtuelle) Maschine, für die das Programm bestimmt ist, implementiert. Im Falle einer ↑Ausnahme kann jedoch ein anderer Interpreter helfend eingreifen und teilweise die Programmausführung übernehmen (↑partielle Interpretation).

interrupt (dt.) ↑Unterbrechung, unterbrechen.

interrupt descriptor table (dt.) ↑Unterbrechungsdeskriptortabelle.

interrupt handler (dt.) ↑Unterbrechungshandhaber.

interrupt handler stub (dt.) ↑Unterbrechungshandhaberstumpf.

interrupt line (dt.) ↑Störleitung.

interrupt mask (dt.) ↑Unterbrechungsmaske.

interrupt vector (dt.) ↑Unterbrechungsvektor.

interrupt vector table (dt.) ↑Unterbrechungsvektortabelle.

IP Abkürzung für (en.) *Internet Protocol*, seit 1974. Ermöglicht die Kommunikation von Nachrichtenpaketen fester Länge in einem ↑Netzwerk. Grundfunktionen sind (1) Adressierung der Netzknoten, (2) Kapselung, Formatierung und Verpackung der ↑Daten, (3) Stückung und Wiederzusammenfügung von Datagrammen und (4) Leitweglenkung (*routing*).

IPL Abkürzung für (en.) *interrupt priority level*, (dt.) ↑Unterbrechungsprioritätsebene.

IRQ Abkürzung für (en.) *interrupt request*, (dt.) ↑Unterbrechungsanforderung.

irrige Mitbenutzung Zugriffsmuster auf einzelne Strukturelemente (↑Speicherwort) der kleinsten Verwaltungseinheit einer Speicherressource (↑Zwischenspeicherzeile, ↑Seite), wobei nur letztere der Mitbenutzung (*sharing*) unterliegt. Auch wenn simultane Schreib-lese-Vorgänge verschiedene Strukturelemente betreffen, also logisch nicht zueinander in Konflikt stehen, besteht in solch einem Fall dennoch ein physischer Schreib-lese-Konflikt, nämlich in Bezug auf die umfassende Verwaltungseinheit. Zur Konsistenzwahrung zwischengespeicherter globaler Daten bewirkt jeder Schreibvorgang entweder die Ausspülung (*write-invalidate*) oder die Aktualisierung (*write-update*) der betreffenden Speicherressource. Dies hat leistungsmindernde Blocktransfers zur Folge, obwohl in logischer Hinsicht keine Zugriffskonflikte bestehen.

ISA Abkürzung für (en.) *instruction set architecture*.

isolierte Ein-/Ausgabe Art von Ein-/Ausgabe, bei der die ↑Ein-/Ausgaberegister von einem ↑Peripheriegerät über einen speziellen Anschluss (*port*) zugänglich sind. Der Anschluss wird über eine Nummer identifiziert, die einer ↑Adresse gleicht, jedoch keine ↑Speicherstelle im eigentlichen Sinn adressiert. Zur Durchführung eines Ein-/Ausgabevorgangs ist ein spezieller ↑Maschinenbefehl (`in`/`out` bei x86) erforderlich, der ↑Daten zwischen einem ↑Prozessorregister und dem ausgewählten Anschluss transferiert.

ITS Mehrplatz-/Mehrbenutzerbetriebssystem. Abkürzung für (en.) *Incompatible Time-Sharing System*, als Seitenhieb auf ↑CTSS. Implementiert in ↑Assembliersprache, erste Installation um 1969 (PDP-6, später PDP-10). Leistete ↑Mehrprogrammbetrieb ohne ↑Speicherschutz, machte virtuelle Geräte (zur geräteunabhängigen Ein-/Ausgabe) verfügbar, erlaubte netzwerktransparenten Dateizugriff über ↑ARPANET, ermöglichte die Suspendierung von dem jeweils im Vordergrund stattfindenden ↑Prozess (`^Z`) und bot einen Mechanismus, um einen ↑Systemaufruf sicher unterbrechen zu können (↑PCLSRing). Alle diese Konzepte, bis auf den fehlenden Speicherschutz, sind für ↑UNIX übernommen worden. Das System hatte wesentlichen Einfluss auf die Hackerkultur.

IVT Abkürzung für (en.) ↑*interrupt vector table*.

Java Programmiersprache: imperativ, objektorientiert (1995), eigentlich ↑Interpretersprache.

JCL Abkürzung für (en.) ↑*job control language*.

job (dt.) Tätigkeit, hier: ↑Teilaufgabe.

job control language (dt.) ↑Auftragssteuersprache.

jump label (dt.) ↑Sprungmarke.

JVM Abkürzung für (en.) *Java Virtual Machine*, Teil der Laufzeitumgebung für ein in ↑Java formuliertes ↑Programm. Typischerweise als ↑CSIM realisierte ↑virtuelle Maschine, die ↑Zwischenkode (konkret: ↑Java Bytekode) direkt ausführt. Jedes Programm läuft auf seiner eigenen virtuellen Maschine.

Kachel ↑Seitenrahmen.

Kanal Steuereinheit, die die Operationen von einem ↑Peripheriegerät kontrolliert, wobei sich die Betriebsüberwachung (*sphere of control*) auf mehr als ein Gerät derselben Art oder verschiedener Arten erstrecken kann. Wesentliche Funktion dieser Einheit ist es, den bei der Ein-/Ausgabe anfallenden Transfer von ↑Daten zwischen ↑Hauptspeicher und ↑Peripherie unabhängig von der ↑CPU durchzuführen. Dazu nutzt die Einheit ↑Speicherdirektzugriff und sendet zu gegebener Zeit eine ↑Unterbrechungsanforderung an die CPU, um die Beendigung der asynchron geleisteten Ein-/Ausgabearbeit anzuzeigen. Diese als ↑Prozessor fungierende Einheit ist durch ein ↑Kanalprogramm, das die jeweils zu leistende Arbeit genau beschreibt,

zwar sehr flexibel einsetzbar, sie dient jedoch vornehmlich dem Sonderzweck (*special purpose*) der Entlastung der CPU von jeder zeitaufwendigen ↑Aufgabe, die mit Ein-/Ausgabe in Verbindung steht. Seit Einführung mit ↑IBM 709, nach wie vor die in einem Großrechner von IBM vorherrschende Ein-/Ausgabetechnik.

Kanalprogramm Folge von Anweisungen für eine Steuereinheit zur eigenständigen, von der ↑CPU unabhängigen Ein-/Ausgabe von ↑Daten. Die Steuereinheit ist ein dedizierter ↑Prozessor, der einen Ein-/Ausgabekanal (↑*channel*) implementiert. Über diesen ↑Kanal wird der Datentransfer zwischen ↑Hauptspeicher und ↑Peripherie direkt abgewickelt, gesteuert durch ein vom ↑Betriebssystem dazu jeweils dem Prozessor zur Ausführung übergebenes ↑Programm. Eingeführt mit ↑IBM 709, weiterentwickelt für ↑IBM System/360 und nach wie vor zentrale Ein-/Ausgabetechnik der ↑IBM z Systems.

Kartenleser ↑Peripheriegerät, das auf Knopfdruck automatisch eine ↑Lochkarte nach der anderen einliest, die darauf kodierten ↑Daten (mechanisch oder optisch) abfragt und mittels ↑Ein-/Ausgaberegister dem zugehörigen ↑Gerätetreiber zur Eingabe bereitstellt. Der Gerätetreiber ist für gewöhnlich ein ↑Unterprogramm von einem Steuerprogramm zur Organisation und Überwachung des Rechnerbetriebs (↑*resident monitor*). Falls ↑abgesetzter Betrieb geführt wird, ist das Gerät selbst am ↑Satellitenrechner angeschlossen, ansonsten am ↑Hauptrechner.

Kartenlocher Gerät zur Erfassung von ↑Daten und deren direkte Speicherung auf einer ↑Lochkarte in kodierter Form. Über eine Zuführvorrichtung werden nach und nach die leeren Lochkarten zur Erfassung automatisch bereitgestellt. Eine Ablagevorrichtung nimmt abschließend die jeweils bearbeitete Lochkarte auf. Die Dateneingabe erfolgt durch eine Tastatur, die der von einem ↑Fernschreiber oder einer Schreibmaschine gleicht. Bei Betätigung der Taste eines alphanumerischen Zeichens (d.h., Buchstabe, Ziffer oder Interpunktionszeichen), erzeugt eine elektromechanisch arbeitende Eingabeeinheit eine Lochung in der aktuellen Spalte einer eingelegten Lochkarte und transportiert die Karte automatisch um eine Spalte weiter nach links. Die Lochung entspricht dem für das jeweilige Zeichen verwendeten Kode (↑EBCDIC). Der Wechsel zu nächsten Karte erfolgt automatisch, wenn eine komplette Zeile (d.h., 80 Spalten) abgelocht wurde, oder durch Betätigung der Taste zum ↑Wagenrücklauf. Bei Geräten mir Kopiervorrichtung, können bestimmte oder alle Spaltenlochungen von einer vorhergehenden Karte mittels Kopiertaste automatisch auf die nachfolgende Karte übertragen werden. Gegebenenfalls wird jedes eingegebene Zeichen an der Oberkante der Karte, über seiner Lochung in der Spalte, in für den Menschen lesbarer Form zusätzlich dargestellt.

Kartenstanzer ↑Peripheriegerät, das durch einen zugehörigen ↑Gerätetreiber die zur Ausgabe nacheinander mittels ↑Ein-/Ausgaberegister bereitgestellten ↑Daten automatisch auf eine ↑Lochkarte nach der anderen in entsprechend kodierter Form überträgt. Der Gerätetreiber ist für gewöhnlich ein ↑Unterprogramm von einem Steuerprogramm zur Organisation und Überwachung des Rechnerbetriebs (↑*resident monitor*). Falls ↑abgesetzter Betrieb geführt wird, ist das Gerät selbst am ↑Satellitenrechner angeschlossen, ansonsten am ↑Hauptrechner.

kaskadierte Unterbrechung (en.) ↑*cascaded interrupt*. Bezeichnung für eine stufenförmig angeordnete ↑Unterbrechung, wobei jede Stufe einer bestimmten ↑Unterbrechungsprioritätsebene entspricht. Die Kaskade bildet sich, sobald eine ↑Unterbrechungsbehandlung niedriger Priorität durch eine ↑Unterbrechungsanforderung höherer Priorität unterbrochen wird. In dem Fall wird die Behandlung auf niedrigerer Prioritätsebene aus- und erst wieder fortgesetzt, wenn die durch die Anforderung ausgelöste Behandlung auf höherer Prioritätsebene zum Abschluss gekommen ist. Diese Kaskadierung kann sich wiederholen, in Abhängigkeit von den Eigenschaften der Hardware (↑CPU, ↑PIC): sie bildet einen Stapel von laufenden Unterbrechungsbehandlungen, wobei aber immer nur die oben auf dem Stapel liegende Behandlung voranschreitet und alle im Stapel liegenden Behandlungen sich verzögern.

Typisch ist eine solche Kaskade unterschiedlicher Prioritätsebenen für den Unterbrechungstyp ↑IRQ. Sie bildet sich aber auch im Falle eines ↑NMI, wobei dann jedoch nicht mit jeder Unterbrechungsanforderung dieser Art eine weitere Prioritätsebene entsteht, wohl aber eine weitere Inkarnation eines ↑Unterbrechungshandhabers. Ein NMI ist nicht nur nicht maskierbar, sondern aktiviert auch immer die höchste Prioritätsebene: jede Inkarnation des entsprechenden Handhabers wird mit derselben Priorität zur Ausführung kommen. So erweitert der NMI die durch den IRQ möglichen Prioritätsebenen effektiv nur um eine weitere Ebene, jedoch bleibt dadurch die Anzahl der Handhaberinkarnationen unbestimmt:

$$N_{level} = \sum_{l=0}^{n} IRQ_l + NMI \qquad \text{versus} \qquad N_{incarnation} = \sum_{l=0}^{n} IRQ_l + \sum_{l=0}^{\infty} NMI_l$$

Diese Unbestimmtheit der Anzahl möglicher Handhaberinkarnationen, wenn der NMI mit in die Betrachtung einbezogen wird, stellt ein grundsätzliches Problem der ↑Systemprogrammierung für die „nackte Hardware“ dar. Sollte etwa die Hardware einen Defekt (*fault*) in Bezug auf den NMI zeigen, der eine hochfrequente „unechte Unterbrechung“ (↑*spurious interrupt*) als Fehler (*error*) hervorbringt, dann besteht die große Gefahr für das totale Versagen (*failure*) des ↑Rechensystems. Daher wird der NMI auch nur verwendet, um ein schwerwiegendes Problem als ↑Ereignis an das ↑Betriebssystem zu signalisieren. Darüberhinaus wird für diese Signalisierung für gewöhnlich die ↑Flankensteuerung genutzt, um nämlich die Wahrscheinlichkeit zu senken, dass der ↑Laufzeitstapel im Betriebssystem überläuft und dadurch ↑Panik entsteht — was bei ↑Pegelsteuerung leicht der Fall wäre, da der NMI-Handhaber niemals die Unterbrechungsanforderung quittieren sowie abstellen könnte, da er nämlich sofort nach seinem Start wieder unterbrochen wird und im weiteren Verlauf in eine (logisch) unendliche indirekte ↑Rekursion verfällt. Gleiches könnte übrigens geschehen, sollte ein IRQ-Handhaber während seines Verlaufs von sich aus auf eine niedrigere Prioritätsebene wechseln und er dadurch Gefahr läuft, mit einer der nächsten Unterbrechungsanforderungen sich selbst in der Ausführung rekursiv zu überlappen.
Aus dieser Gründen findet eine Unterbrechungsbehandlung für gewöhnlich in zwei Phasen statt. Zum einen ist jede der Unterbrechungsbehandlungen durch einen Unterbrechungshandhaber „erster Stufe“ (↑FLIH) repräsentiert, der mit der durch die Hardware definierten Unterbrechungspriorität ausgeführt wird. Zum anderen kommen Unterbrechungshandhaber „zweiter Stufe“ (↑SLIH) zur Ausführung, um ↑Aufgaben von geringerer und durch Software definierter Priorität zu bewerkstelligen. Letztere starten aber erst dann, wenn eine möglicherweise vorhandene Kaskade ersterer komplett wieder abgebaut wurde. Die Kaskadierung ist und bleibt damit in erster Linie ein Merkmal der Hardware.

Kausalordnung Bezeichnung für eine die *Kausalität* (Beziehung zwischen Ursache und Wirkung) bestimmende Reihenfolge innerhalb einer Menge von ↑Ereignissen. Ein Ereignis B, das erst stattfinden kann (Wirkung), nachdem ein anderes Ereignis A bereits stattgefunden hat (Ursache), ist kausal abhängig ist von diesem anderen Ereignis ($A < B$ bzw. $B > A$): Ereignis B wird durch Ereignis A beeinflusst.

Kennungswechsel Veränderung der ↑Benutzerkennung oder ↑Gruppenkennung von einem ↑Prozess. Der Wechsel geschieht explizit durch einen ↑Systemaufruf (z.B. `setuid(2)`) oder implizit bei Gebrauch von einem ↑Objekt, das dazu mit einem speziellen Attribut versehen sein muss. Beispiel für letzteres ist das *setuid*-Recht einer ausführbaren ↑Datei, etwa eine Datei, die ein ↑Maschinenprogramm enthält. Wird dieses Maschinenprogramm zur Ausführung gebracht, findet der zugehörige Prozess, zusätzlich zu seinen ursprünglichen Rechten, mit den Rechten derjenigen ↑Entität statt, der die Datei gehört.
Typischerweise wird — in einem ↑UNIX-artigen ↑Betriebssystem — zwischen echter (*real*), effektiver (*effective*) und geretteter (*saved*) Kennung eines Prozesses unterschieden. Die echte Kennung wird bei der ↑Anmeldung zugewiesen und definiert die ursprüngliche (*primary*) Kennung eines Prozesses. Die effektive Kennung nutzt ein Prozess, um zeitweise mit höheren Privilegien oder Rechten stattfinden zu können. Mit ihr erfolgt in den überwiegenden

Fällen die Rechteprüfung im Betriebssystem. Die gerettete Kennung erlaubt einem zeitweise mit höheren Privilegien/Rechten stattfindenden Prozess, temporär mit niedrigeren Privilegien/Rechten voranzuschreiten, ohne seinen erhöhten Rechtestatus zu verlieren. Initial haben alle drei Kennungen denselben Wert, nämlich den der echten Kennung.

kernel mode ↑*privileged mode*.

kernel thread (dt.) ↑Systemkernfaden.

key punch (dt.) ↑Kartenlocher.

Kommando Befehl, der einem ↑Prozess einen ↑Auftrag erteilt beziehungsweise zur Übernahme einer bestimmten ↑Aufgabe verpflichtet. Die einzelne Anweisung in einer ↑Kommandosprache.

Kommandointerpreter Bezeichnung für ein ↑Programm, bei dessen Ablauf die in ↑Kommandosprache formulierten Anweisungen an ein ↑Rechensystem interpretiert werden. Ein ↑Interpreter von Kommandos, die ihm als Befehlsstrom in einem ↑Stapel (↑*batch processing*) oder interaktiv über eine ↑Dialogstation (↑*time sharing*) zugestellt werden: letzteres, um eine Sitzung mit dem ↑Betriebssystem zu bestreiten (↑*shell*). Ursprünglich (um 1955) jedoch Inbegriff für das zentrale Steuerprogramm zur ↑Stapelverarbeitung und wesentlicher Bestandteil von einem „embryonalen Betriebssystem“ (↑*resident monitor*, ↑FMS).

Kommandosprache Sprache, in der ein ↑Auftrag an ein in ↑Stapelbetrieb laufendes ↑Rechensystem formuliert wird. Ursprünglich eine reine Steuersprache (↑*job control language*) zur ↑Stapelverarbeitung, um die ↑automatisierte Rechnerbestückung durch einen ↑Kommandointerpreter zu erreichen. Als eine Art ↑Skriptsprache heutzutage (2016) jedoch auch ein Ausdrucksmittel, um wiederkehrende oder interaktionslose Arbeiten am ↑Rechner zu verrichten.
Nachfolgend zwei Beispiele, links ein (fiktiver) Auftrag für ↑FMS zur Ausführung von einem ↑Programm geschrieben in ↑FORTRAN und rechts ein Auftrag für eine(n) ↑*shell* (`bash(1)`) zur Ausführung eines semantisch äquivalenten Programms in ↑C:

```
*JOB, 42, ARTHUR DENT
*XEQ
*FORTRAN
 PRINT *, "Hello World!"
 END
*END
```

```
echo '
int main() {
    printf("Hello World!\n");
}
' | gcc -x c -
./a.out
```

Beiden Beispielen gemeinsam ist die vollständige Beschreibung (in einer domänenspezifischen ↑JCL) des von einen Kommandointerpreter abzuwickelnden Auftrags, um diesen dann ohne weiteren Eingriff ausführen zu können (↑*batch processing*). Im Beispiel links ist ein für den Kommandointerpreter bestimmtes ↑Kommando durch das Fluchtsymbol `*` (*escape character*) markiert. Bei Verwendung einer ↑Lochkarte sind solche Fluchtsymbole gelegentlich einer speziellen (z.B. der ersten) Spalte vorbehalten, um dadurch die Auswertung aller nachfolgenden Zeichen dieser Zeile dem Kommandointerpreter zu übertragen. Im Beispiel rechts dagegen erfolgt durch das Fluchtsymbol `'` eine Auswertungsunterdrückung (*quoting*), um die bis zum folgenden Zeichen `'` stehenden Anweisungen dem Kommandointerpreter vorzuenthalten. Der für FMS bestimmte Auftrag beschreibt eine durch `JOB` im ↑Betriebssystem mit der Kennung `42` zur Abrechnung für Benutzer `ARTHUR DENT` identifizierte ↑Teilaufgabe zur Ausführung (`XEQ`) des Übersetzers (`FORTRAN`), der die nachfolgenden Zeilen in ein anschließend sofort auszuführendes ↑Maschinenprogramm transformieren soll. Dabei markiert das erste `END` einerseits das Ende der Übersetzungseinheit und andererseits den Start der Programmausführung, wohingegen das zweite `END` das Auftragsende für den Kommandointerpreter anzeigt. Der *shell*-Auftrag leitet mittels `|` (*pipe*-Kommando) die in C formulierten, durch `'` zunächst unterdrückten und dann aber wiedergegebenen (`echo`), Anweisungen in

den Übersetzer (`gcc`), um das ↑Lademodul (`a.out`) erzeugt zu bekommen und daraufhin das generierte Maschinenprogramm im ↑Arbeitsverzeichnis zu starten (`./a.out`).

Kommandozeile Reihe von nebeneinanderstehenden Wörtern, wobei das erste Wort für gewöhnlich ein ↑Kommando bezeichnet und die nachfolgenden Wörter Optionen oder Parameter dazu benennen. Die korrekte Verknüpfung der sprachlichen Einheiten in einem Satz (d.h., die Syntax) gibt die verwendete ↑Kommandosprache vor. Gegebenenfalls erlaubt die Sprache mehrere solcher Kommandos pro Zeile, jeweils getrennt durch ein spezielles Satzzeichen. Mit abgeschlossener Eingabe der Reihe (↑Wagenrücklauf) beginnt die Ausführung der Kommandos durch ein spezielles Steuerprogramm (↑CLI).

Kompilation Vorgang der ↑Übersetzung von einem ↑Programm durch einen ↑Kompilierer.

Kompilierer Softwareprozessor, der ein ↑Programm, das in einer Quellsprache formuliert vorliegt, in ein semantisch äquivalentes Programm einer Zielsprache übersetzt.

komplexe Koroutine Bezeichnung für eine ↑Koroutine, die getrennt von anderen ihrer Art einen eigenen ↑Laufzeitstapel besitzt. Die ↑Handhabe für einen ↑Handlungsstrang innerhalb einer solchen Koroutine ist ein ↑Stapelzeigerwert. Dieser Wert ist (logisch nicht auflösbar, zusammenfassend) der Zeiger auf einen Zeiger auf den ↑Maschinenbefehl, der nach erfolgtem ↑Koroutinenwechsel als nächster ausgeführt wird. Anders als eine ↑primitive Koroutine.

Konkurrenzsituation Umstand, in dem sich ein ↑Prozess bei dem Versuch befindet, ein ↑unteilbares Betriebsmittel zu beanspruchen, das bereits von einem anderen Prozess belegt ist. In dieser Lage muss der Prozess warten, bis das Betriebsmittel wieder freigegeben wird und er an die Reihe kommt, das freigegebene Betriebsmittel zu belegen. Für die Reihenfolgebildung unterliegen die konkurrierenden Prozesse der ↑Synchronisierung. Typisches Beispiel dafür ist ein ↑kritischer Abschnitt.

Konsolenprotokoll Aufzeichnung der auf einem ↑Rechner ablaufenden Vorgänge (Duden) zur Darstellung auf der ↑Systemkonsole und Aufzeichnung in einer ↑Datei oder auf Druckpapier.

Konstantenfaltung (en.) ↑*constant folding*. Bezeichnung für eine Optimierungstechnik bei der ↑Kompilation. Dabei werden konstante Ausdrücke in einem ↑Programm vom ↑Kompilierer bereits zur Übersetzungszeit erkannt und ausgewertet, anstatt sie erst zur ↑Laufzeit des Programms durch den ↑Prozessor berechnen zu lassen.

Konstantenpropagation (en.) ↑*constant propagation*. Bezeichnung für eine Optimierungstechnik bei der ↑Kompilation. Dabei werden Werte von Konstanten in Ausdrücken in einem ↑Programm vom ↑Kompilierer zur Übersetzungszeit ersetzt. Diese Konstanten sind entweder im Programm daselbst kodiert oder wurden durch ↑Konstantenfaltung bestimmt.

konsumierbares Betriebsmittel Bezeichnung für ein ↑Betriebsmittel, das logisch in unbegrenzter Anzahl vorhanden und von dem jede Einheit verfügbar ist. Wird eine Einheit davon von einem als *Konsument* (Verbraucher) auftretender ↑Prozess erworben (*acquire*), hört sie in dem Moment auf zu bestehen, sie erlischt, verliert Gültigkeit, wird implizit zerstört. Nur ein als *Produzent* (Erzeuger) dieses Betriebsmittels agierender Prozess kann Einheiten davon in beliebiger Anzahl freisetzen (*release*). Dies kann zu beliebigen Zeitpunkten geschehen, sofern der Prozess seinerseits nicht den Erwerb einer Betriebsmitteleinheit erwartet und demzufolge blockiert ist. Jede freigesetzte Einheit des Betriebsmittels wird sofort verfügbar. Beispiele sind Signale der Hardware (↑IRQ, ↑NMI) oder Software (↑Semaphor, `signal(2)`), Nachrichten oder jede Form von Daten, die im weitesten Sinne Ein- oder Ausgabe eines Prozesses darstellen.

Kontrollstruktur Spezifikation der für die ↑Interpretation benötigten Algorithmen und der Transformation der Informationsträger einer (realen/virtuellen) Maschine, ein Aspekt in ihrem ↑Operationsprinzip.

Konvoieffekt Bild für die Auswirkung einer nach ↑FCFS vorgehenden ↑Ablaufplanung, wenn ein Mix von kurzen und langen ↑Prozessen gegeben ist. Dabei entspricht die Prozesslänge der Dauer von einem ↑Rechenstoß, das heißt, der jeweils (zu erwartenden) ↑Bedienzeit des Prozesses durch den ↑Prozessor. Dem Effekt liegt die Annahme zugrunde, kurze Rechenstöße seien ein-/ausgabeintensive (d.h., interaktive, ↑*I/O bound*) und lange Rechenstöße seien rechenintensive (d.h., nicht interaktive, ↑*CPU bound*) Prozesse.
Wenn nun Prozesse kurzer Rechenstöße vergleichsweise lange auf der ↑Bereitliste stehen, weil sie erst die Freigabe des Prozessors durch einen Prozess mit vergleichsweise langem Rechenstoß abwarten müssen, kann in der jeweiligen ↑Wartezeit kein ↑Ein-/Ausgabestoß von ihnen gestartet werden: die von ihnen beanspruchte ↑Peripherie liegt in der Zeit brach, wenn sie nicht noch mit einer vorangegangenen ↑Aufgabe beschäftigt ist. Gibt der lange Prozess den Prozessor ab, werden die kurzen Prozesse nach und nach ihre Zeit nutzen, vornehmlich Ein-/Ausgabestöße zu starten, um daraufhin ihrerseits den Prozessor wieder abzugeben. In der Zwischenzeit sei ein langer Prozess, gegebenfalls derselbe wie zuvor, wieder an der Reihe. Dieser belegt den Prozesor, nachdem der letzte der kurzen Prozesse die Kontrolle abgegeben hat. Während der lange Prozess rechnet werden die kurzen Prozesse wieder bereitgestellt, müssen abermals auf die Zuteilung des Prozessors warten, können daher ihre Ein-/Ausgabestöße nicht absetzen und lassen die Peripherie damit erneut unter- oder unbeschäftigt. Dieser Zyklus wiederholt sich, solange der bestehende Prozessmix unverändert bleibt. Die Folge ist, dass der Prozessor zwar durchgehend beschäftigt ist, die Peripherie jedoch phasenweise immer nichts zu tun bekommt — dies obwohl Prozesse bereitstehen, die der Peripherie Ein-/Ausgabeaufträge zustellen würden, jedoch nicht können, weil sie eben noch nicht an der Reihe sind, den Prozessor zu erhalten.
Die kurzen Prozesse bilden einen Verband von zusammenhängenden Aufgaben für die Peripherie, der allerdings immer erst dann vorankommt, wenn der lange Prozess den Prozessor verlässt. Der Prozessor gleicht einer einspurigen Straße (↑wiederverwendbares Betriebsmittel), auf der die schnelleren Autos (kurze Prozesse) darauf warten müssen, dass der langsamere Laster (langer Prozess) vor ihnen sie vorbeilässt (eine Ausweichstelle benutzt). Dieses Verkehrsszenario ist der Ursprung für die Namensgebung des beschriebenen Effekts.

kooperative Planung (en.) ↑*cooperative scheduling*. Modell der ↑Ablaufplanung, die (im Gegensatz zu ↑präemptive Planung) davon ausgeht, dass ↑Prozesse stets bereitwillig und in Zusammenarbeit mit dem ↑Betriebssystem die Kontrolle über den ↑Prozessor ab- oder weitergeben. Die Prozesse sichern zu, von Zeit zu Zeit einen ↑Systemaufruf zu tätigen, dadurch dem Betriebssystem die Möglichkeit zur ↑Umplanung zu geben und somit anderen Prozessen den Prozessor zukommen zu lassen. Dieser Systemaufruf ist entweder direkt der Ablaufplanung gewidmet oder betrifft die Durchführung einer beliebigen anderen ↑Systemfunktion. Im letzteren Fall wird (insb. vom ↑Systemaufrufzuteiler) jedoch immer auch die Ablaufplanung gestreift, die dann indirekt durch den Systemaufruf ihre Funktion ausüben kann.
Diese strikt kooperative Vorgehensweise schließt allerdings durch ↑Unterbrechungen ausgelöste asynchrone Abläufe im Betriebssystem nicht aus. Auch kann die Ablaufplanung als Folge solcher Unterbrechungen sehr wohl tätig werden, jedoch wird sie niemals sofort die ↑Einlastung der ↑CPU mit einer ↑Prozessinkarnation, das heißt, einen ↑Prozesswechsel nach sich ziehen: ↑Einplanung und Einlastung sind zwar räumlich gekoppelt, aber zeitlich entkoppelt. Sollte die Einplanung einen anderen als den derzeit als laufend (↑Prozesszustand) bekannten Prozess bevorzugen, wird die Einlastung und damit der Prozesswechsel nicht vor dem nächsten Systemaufruf vollzogen. Bleibt ein Systemaufruf aus, erhält kein anderer Prozess die CPU zugeteilt: der gegenwärtig stattfindende Prozess monopolisiert damit die CPU.

Koordination Aufeinanderabstimmen von Vorgängen, diese nebenordnen. Die dazu in einem ↑Rechensystem vorhandenen Maßnahmen greifen unterschiedlich. Zum einen die ↑Ablaufplanung, die einen ↑Prozess nebenordnet bevor dieser seine Tätigkeit (erneut) aufnimmt. Sind für die Prozesse alle Daten-, Kontrollfluss- und Zeitabhängigkeiten vorab bekannt, ist ein (statischer) ↑Ablaufplan möglich, durch den die Prozesse implizit koordiniert stattfinden werden. Fehlt jedoch dieses Wissen im Moment der ↑Prozesseinplanung oder verbietet eine zu

hohe Berechnungskomplexität die mitlaufende (*on-line*) Erstellung und Fortschreibung eines solchen Plans, ist zum anderen explizite ↑Nebenläufigkeitssteuerung der dann stattfindenden Prozesse erforderlich. Letztere unternehmen die Prozesse selbst, indem sie Einvernehmen über den weiteren Ablauf in Abhängigkeit der von ihnen gerade beabsichtigen ↑Aktion oder ↑Aktionsfolge erzielen und sich so nebenordnen.

Koroutine (en.) ↑*coroutine*. Bezeichnung für eine ↑Routine, die zusammen mit (lat. *con*) anderen Routinen auf derselben Stufe steht. Eine Art „Nebenprogramm“, das zwar die Rolle von einem ↑Hauptprogramm einnimmt, aber in Wirklichkeit kein solches Hauptprogramm darstellt: um abzulaufen, wird keine dieser Routinen aufgerufen, jedoch kann, wechselseitig und in kooperativer Art und Weise, jede die andere fortsetzen (↑*resume*). Zur Fortsetzung einer so gleichgestellten Routine unterbricht die laufende Routine (↑Prozess) ihre Ausführung, um diese auf Veranlassung einer anderen Routine dieser Art später wieder aufzunehmen. Dabei definiert der Pfad bis zum jeweils nächsten Unterbrechungspunkt innerhalb einer solchen Routine einen autonomen ↑Handlungsstrang. Je nach der inneren Struktur sind mehrere dieser Stränge innerhalb einer solchen Routine möglich, die in Abhängigkeit von dem jeweils definierenden Pfad verschieden sein können.
Ein einfaches Beispiel zeigen die folgenden beiden ↑Programme in ↑C. Hier ist der Rumpf einer solchen wechselseitig betreibbaren Routine programmiersprachlich als ↑Unterprogramm (`niam`) ausformuliert, wobei die ↑Handhabe für den entsprechenden Handlungsstrang innerhalb dieser Routine als untypisierter Zeiger (`void*`) ausgelegt ist:

```
typedef void* coroutine_t;              /* handle type */

coroutine_t __attribute__((noreturn)) niam (coroutine_t root) {
    static volatile coroutine_t peer;  /* own variable */
    peer = root;                        /* remember origin */
    printf("niam(%p) at %p\n", root, niam);
    for (;;) {
        printf("niam: resume %p\n", peer);
        peer = resume(peer);            /* switch coroutine */
    }
}
```

Diese Routine beschreibt zwei Handlungsstränge. Der erste Strang reicht vom Routinenanfang bis zum ersten Aufruf der Funktion `resume` (↑Koroutinenwechsel) innerhalb der Endlosschleife und der zweite Strang reicht von der Rückkehr aus dieser Funktion bis zum erneuten Aufruf (einzelner Schleifendurchlauf). Zwischen diesen Handlungssträngen ist die betreffende Routine (`niam`) inaktiv.
Das Attribut `noreturn` hat sowohl einen dokumentarischen als auch einen technischen Zweck. Einerseits weist es darauf hin, dass gleichgestellte Routinen (wie `niam`) einen Aufruf nicht durch eine Rückkehr beantworten dürfen. Andererseits bedingt dieses Attribut eine Warnmeldung durch den Kompilierer (`gcc`), sollte dennoch fälschlicherweise ein Rückkehrpfad aus der Routine existieren. Eingangsparameter (`root`) ist eine Handhabe, die die erzeugende und initial aufrufende Routine identifiziert. Mit dieser kann der wechselseitige Betrieb aufgenommen werden. Der lokale Zustand des Handlungsstrangs ist in einer ↑Eigenvariablen (`peer`) gespeichert und bleibt damit während Phasen seiner Inaktivität erhalten. Im Rahmen einer Endlosschleife wird diese Variable hier lediglich als Handhabe genutzt, um die jeweils den Handlungsstrang (in `niam`) aktivierende ↑Entität sicher reaktivieren zu können.
Nachfolgende Programmskizze zeigt das entsprechende Pendant, sie beschreibt das Hauptprogramm (`main`) im eigentlichen Sinne beziehungsweise die ursprüngliche (`niam`) gleichgestelle Routine:

```
int main (int argc, char *argv[]) {
    static volatile coroutine_t next;  /* own variable */
    printf("main(%d) at %p\n", argc, main);
    next = assume(niam);               /* make coroutine from routine */
    while (argc--) {
        printf("main: resume %p\n", next);
        next = resume(next);           /* switch coroutine */
    }
}
```

Diese Routine beschreibt vier Handlungsstränge. Der erste Strang reicht vom Routinenanfang bis zum Aufruf der Funktion `assume` (↑Koroutinenaufbau). Mit Rückkehr aus dieser Funktion startet der zweite Handlungsstrang, der bis zum ersten Aufruf der Funktion `resume` innerhalb der Kopfschleife reicht. Der dritte Strang reicht von der Rückkehr aus und bis zum wiederholten Aufruf von `resume`, sofern die Kopfschleife mehr als einmal durchlaufen wird ($argc > 0$). Schließlich der vierte Handlungsstrang, der mit der Rückkehr aus der Funktion `resume` beginnt, die Kopfschleife verlässt ($argc = 0$) und die Ausführung des Programms beendet. Zwischen diesen Handlungssträngen ist die Routine (`main`) inaktiv.
Auch hier wird zur Speicherung des lokalen Datenzustands eine Eigenvariable (`next`) verwendet: wie im anderen Fall auch, enthält diese Variable eine Handhabe für den jeweils als nächstes zu aktivierenden Handlungsstrang. Damit in einer Routine autonome Handlungsstränge überhaupt möglich sind, muss diese die Eigenschaft zum wechselseitigen Betrieb übernehmen: ein Aufruf der Funktion `assume` bewirkt, dass die als ↑tatsächlicher Parameter (`niam`) spezifizierte Routine eben diese Eigenschaft übertragen wird (↑Koroutinenaufbau). Im konkreten Beispiel erhält die Routine `niam` diese Eigenschaft, die jedem ihrer Handlungsstränge dadurch ermöglicht, mit einem Handlungsstrang in der Routine `main` wechselseitig, durch Verwendung der Funktion `resume`, ablaufen zu können.
Da keine dieser gleichgestellten Routinen gewöhnlich aufgerufen wird, können sie auch nirgends hin zurückkehren: der Rückkehrversuch stellt eine ↑Ausnahmesituation dar, die den weiteren Verlauf des entsprechenden Handlungsstrangs undefiniert lässt. Beabsichtigt eine solche Routine dennoch die Beendigung, macht sie dies durch einen entsprechenden globalen Zustand deutlich, unterbricht die Ausführung und gibt dadurch Möglichkeit zur Entsorgung von anderer Stelle (durch eine andere Routine).

Koroutinenaufbau Errichtung einer ↑Koroutine auf Grundlage einer gewöhnlichen ↑Routine, der dazu die Fähigkeit zur gleichgestellten Ausführung im Zusammenspiel mit anderen Koroutinen derselben Art übertragen wird. Wenn möglich sind Vorkehrungen zur ↑Abfangung der errichteten Koroutine zu treffen, sollte diese unerwarteterweise einen Rückkehrpfad aus der sie definierenden Routine nehmen. Nach erfolgter Übernahme der für diese Form der Ausführung nötigen Eigenschaften schreitet die Ko-/Routine immer wechselseitig und in kooperativer Weise mit ihresgleichen voran (↑*resume*): sie unterbricht ihre Ausführung von Zeit zu Zeit, beendet sich jedoch nicht von selbst. Damit diese wechselseitigen Abläufe möglich sind, ist der den Ausgangspunkt bildenden *Basisroutine*:

1. die Fortsetzungsadresse einer bereits bestehenden Koroutine zu vermitteln, um zu wissen, wohin sie im Falle ihrer Ausführungsunterbrechung wechseln kann und
2. ein Zustand zu geben, der die Fortsetzung ihrer unterbrochenen Ausführung ermöglicht.

Wünschenswertes Kriterium der Maßnahmen ist es, den ↑Koroutinenwechsel unabhängig von seiner jeweiligen Abfolge stattfinden zu lassen, nämlich nicht zwischen den ersten (initialen) und allen weiteren (normalen) Wechseln unterscheiden zu müssen. Dazu muss der Routine der ↑Koroutinenstatus einer scheinbar zuvor bereits gelaufenen und vorübergehend unterbrochenen Koroutine gegeben werden. Folglich unterscheidet sich der durchzuführende beziehungsweise zu hinterlassene Aufbau je nach Art der zu errichtenden Koroutine (↑primitive Koroutine, ↑komplexe Koroutine).

Primitive Koroutine Wesentliches Merkmal dieser Art von Koroutine ist der allen ↑Exemplaren eben dieses elementaren Typs gemeinsame ↑Laufzeitstapel. Dadurch eröffnen sich grundsätzlich zwei Wege für die Koroutineneinrichtung, die dann nämlich einerseits fremd- und anderseits selbstgesteuert geschehen kann. Im fremdgesteuerten Fall bringt eine beliebige andere Ko-/Routine die Basisroutine dazu, zukünftig als Koroutine zu funktionieren. Dazu wird die Basisroutine derart durch eine ↑Mantelprozedur aufgerufen, dass sie ihre Ausführung unterbrechen und zu einer anderen Koroutine umschalten kann. Die ↑Handhabe für den ↑Handlungsstrang dieser Koroutine hat die Basisroutine beim initialen Aufruf als Parameter übergeben bekommen.
Im selbstgesteuerten Fall sorgt die Basisroutine von sich aus für die Metamorphose. Dazu wird diese Routine bereits durch einen regulären Koroutinenwechsel aktiviert und muss dann allerdings die Handhabe des Handlungsstrangs einer anderen Koroutine selbst in Erfahrung bringen, um zu letzterer irgendwann hinschalten zu können: ein Handlungsstrang innerhalb einer Koroutine hat für gewöhnlich keinen Parameter, weshalb im Fall der initialen Aktivierung die benötigte Handhabe, anders als bei der fremdgesteuerten Variante, auch nicht als Aufrufparameter übergeben werden kann. Stattdessen ist diese Handhabe durch Ausführung einer speziellen *Übernahmefunktion* in Erfahrung zu bringen. Der Einfachheit wegen bietet es sich dann an, die Handhabe desjenigen Handlungsstrangs derjenigen Koroutine zu liefern, aus dem heraus der initiale Koroutinenwechsel geschah.
Die Beschreibung macht deutlich, dass die Einrichtung einer Koroutine keine einfache Operation ist. Typischerweise gestaltet sich diese Operation schwieriger als der Koroutinenwechsel selbst. Ein Beispiel für eine solche Operation liefert nachfolgende Skizze einer entsprechenden Funktion, die für gewöhnlich ein in ↑Assembliersprache (↑GAS) formuliertes ↑Unterprogramm erfordert, da die durchzuführenden ↑Aktionen vom jeweiligen ↑Programmiermodell des zugrunde liegenden Prozessors (↑x86) abhängen:

```
assume:                 # use subroutine call: peer = assume(0 or <name>)
    cmpl $0, 4(%esp)    # check for role of calling co-/routine
    je 1f               # bypass in case of self-controlled setup
    pushl %ebx          # save non-volatile processor state
    pushl %ebp          # "
    pushl %esi          # "
    pushl %edi          # "
    movl %esp, %ebp     # save top of stack
    movl $2f, %eax      # pass home address for self-controlled setup
    pushl %eax          # do similar for pilot-controlled setup
    call *24(%esp)      # perform initial activation for both cases
    xorl %eax, %eax     # should never return to here: error code
2:                      # come here in the case of resumption
    movl %ebp, %esp     # restore top of stack
    popl %edi           # restore non-volatile processor state
    popl %esi           # "
    popl %ebp           # "
    popl %ebx           # "
1:                      # come here in the case of bypass
    ret                 # continue as coroutine or fail
```

Die gezeigte Funktion (`assume`) ist sowohl für die fremd- als auch für die selbstgesteuerte Umwandlung einer Routine in eine Koroutine geeignet. Ein Beispiel für den fremdgesteuerten Ansatz ist bereits an anderer Stelle (S. 71) behandelt worden. Wird dieser Weg gegangen, bildet `assume` die Mantelprozedur für den Aufruf der Basisroutine (`call *24(%esp)`).
Im selbstgesteuerten Ansatz wird davon ausgegangen, dass die Basisroutine durch eine `resume`-Aktion gestartet wurde. In diesem Fall entspricht die von dieser Routine auzurufende Funktion `assume` praktisch einem ↑Leerbefehl, sie umgeht die ↑Aktionsfolge zur fremdgesteuerten Einrichtung einer Koroutine und kehrt sofort wieder zurück (↑Sprungmarke 1). Hier ist

zu beachten, dass in diesem Fall die `assume`-Funktion dann genau den Wert zurück liefert, den eine von diesem Handlungsstrang sonst aufgerufene `resume`-Funktion zurückgeliefert hätte — letztere mangels Eingabeparameter für `resume` in der Initialisierungsphase der neuen Koroutine jedoch noch nicht aufgerufen werden kann. Dieser Rückgabewert ist die Handhabe desjenigen Koroutinenhandlungsstrangs, der `resume` aufgerufen hat, um den nunmehr laufenden Handlungsstrang der gerade neu entstehenden Koroutine zu aktivieren.
Um `assume` im fremdgesteuerten Ansatz auch unabhängig davon benutzen zu können, welchem Model die Basisroutine nun folgt, wird sowohl der Rückgabewert für die selbstgesteuerte Übernahme (`assume(0)`) aufgesetzt (`movl $2f, %eax`) als auch der Eingabeparameter für die fremdgesteuerte Übernahme (`assume(<name>)`) übergeben (`pushl %eax`). Darüber hinaus liefert die Funktion (durch `xorl %eax, %eax`) den Rückgabewert Null, wenn die initial aufgerufene Routine, die eine Koroutine werden sollte, aus dem Aufruf zurückkehrt.

Komplexe Koroutine Anders als im Fall der zuvor behandelten Koroutinenart verfügen die Exemplare einer komplexen Koroutine über einen eigenen ↑Laufzeitstapel. Dies hat zur Folge, dass nur noch die fremdgesteuerte Einrichtung praktiziert werden kann, da eine komplexe Koroutine ohne eigenen Laufzeitstapel nicht ablauffähg ist und daher auch der Aufbau aus eigener Kraft, wie für den selbstgesteuerten Fall notwendig, ausscheidet.
Der koroutineneigene Laufzeitstapel legt die Grundlage für die physische Entkopplung der Handlungsstränge verschiedener Koroutinen. Die Entkopplung zeigt sich insbesondere in der Eigenschaft, die Koroutine initial durch einen gewöhnlichen Prozeduraufruf betreten zu können, der ihr damit einen Rückkehrweg eröffnet. Letzterer bedeutet jedoch lediglich, den betreffenden Handlungsstrang einer solchen Koroutine „einzufangen“ und, wenn möglich, kontrolliert anzuhalten oder eine ↑Ausnahme zu erheben. Auf diesem Stapel werden zudem alle für die Koroutine relevanten ↑Eigenvariablen abgelegt.
Wie nachfolgendes Beispiel (GAS, x86) zeigt, hat die Operation zur Einrichtung einer komplexen Koroutine gewisse Gemeinsamkeiten mit dem Aufbau einer primitiven Koroutine. Dies betrifft jedoch nur die Struktur in drei Blöcken von Maschinenbefehlen, wobei die beiden äußeren zum Sichern und Wiederherstellen des relevanten ↑Prozessorstatus' identisch sind mit der zuvor behandelten Funktionsvariante. Der mittlere Befehlsblock ist spezifisch für den Aufbau einer komplexen Koroutine, er sorgt für (1) den mit eigenem Laufzeitstapel versehenen Aufruf der Basisroutine, (2) die Vermittlung der Fortsetzungsadresse der laufenden Ko-/Routine, wohin die aufgebaute Koroutine durch einen Koroutinenwechsel umschalten kann und (3) die Abfangung, sollte die Basisroutine aus ihrem Aufruf zurückkehren:

```
assign:                 # use subroutine call: peer = assign(name, hook)
    pushl %ebx          # save non-volatile processor state
    pushl %ebp          # "
    pushl %esi          # "
    pushl %edi          # "
    pushl $2f           # setup own resumption address
    movl %esp, %ebx     # setup and backup own coroutine handle
    movl 28(%esp), %esp # switch to stack of new (virgin) coroutine
    pushl %ebx          # pass handle of parent coroutine
3:                      # come here after return from halt: re-enter
    call *24(%ebx)      # perform initial coroutine activation
    hlt                 # should never return to here: halt
    jmp 3b              # should never return from halt: loop
2:                      # come here in the case of resumption
    popl %edi           # restore non-volatile processor state
    popl %esi           # "
    popl %ebp           # "
    popl %ebx           # "
    ret                 # continue as coroutine
```

Zunächst generiert die die `assume-Funktion` aufrufende Ko-/Routine ihre eigene Fortsetzungsadresse und hinterlässt diese auf ihrem Laufzeitstapel (`pushl $2f`). Daraufhin definiert sie die Handhabe (`movl %esp, %ebx`), die später ↑tatsächlicher Parameter der Basisroutine wird. Anschließend erfolgt die Umschaltung hin zum Laufzeitstapel der aufzubauende Koroutine (`movl 28(%esp), %esp`), die Parameterübergabe (`pushl %ebx`) an die Basisroutine sowie ihr Aufruf (`call *24(%ebx)`). Mit dem Basisroutinenaufruf startet der erste Handlungsstrang in der neu eingerichteten Koroutine.
Sollte die Basis-/Koroutine unerwarteterweise aus dem Aufruf zurückkehren, wird der betreffende Handlungsstrang abgefangen. Die radikale Lösung hierfür ist es, die CPU einen ↑Haltebefehl (`hlt`) ausführen zu lassen. Auf der hier betrachteten Abstraktionsebene gibt es keine andere Möglichkeit zu diesem Schritt, da Wissen über die im System vorhandenen Koroutinen fehlt. Dieses Wissen ist erst auf höherer Ebene vorhanden, wenn nämlich Koroutinen die Basis für ↑Prozessinkarnationen bilden, die dann von einem ↑Planer verwaltet werden und dazu, unter anderem, auf der ↑Bereitliste geführt werden. Wenn die CPU ihren Haltezustand verlässt ist es daher auch sinnvoll, die Basis-/Koroutine erneut zu betreten (`jmp 3b`), um auf höherer Ebene Kenntnis von nunmehr gegebenenfalls vorhandenen anderen Koroutinen zu erhalten und zu einer auf dieser Liste wechseln zu können.

Koroutinenstatus Gesamtheit von ↑Daten, die den statischen Zustand einer ↑Koroutine manifestieren. Dieser Zustand umfasst alle Daten, die zur Wiederaufnahme und zum Fortsetzen einer unterbrochenen Koroutinenausführung erforderlich sind.
Unerlässliches Merkmal dieses Datenzustands ist die Fortsetzungsadresse, das heißt, der Wert des ↑Befehlszählers zum Zeitpunkt der Kontrollabgabe der Koroutine über den ↑Prozessor. Dieser Wert (↑Adresse) markiert die Stelle im ↑Programm, an der die Koroutine ihre Ausführung zuletzt unterbrochen hatte und gegebenenfalls wiederaufnehmen wird. Zusätzlich kann dieser Zustand durch die Inhalte weiterer ↑Prozessorregister wie auch lokaler Variablen definiert sein, je nachdem, welcher ↑Aufgabe die Koroutine seit ihrer letzten Ausführungswiederaufnahme nachgekommen ist. All diese Zustandsdaten müssen in Phasen der Inaktivität einer Koroutine invariant sein: jede Variable, die diesen Zustand mit definiert, ist eine ↑Eigenvariable.
Zur Aufbewahrung von Eigenvariablen kann ↑statischer Speicher oder ↑dynamischer Speicher genutzt werden. Letzteres bietet sich vor allem an, wenn das Koroutinenkonzept zur Implementierung von ↑Prozessen zum Einsatz kommen soll (↑komplexe Koroutine). In diesem Fall ist insbesondere der ↑Stapelspeicher ein bedeutsames Hilfsmittel: jede Koroutine verfügt dann über einen eigenen ↑Laufzeitstapel, in dem die Eigenvariablen liegen. Bei einem ↑Koroutinenwechsel wird sodann der ↑Stapelzeiger umgesetzt, wodurch der Datenzustand für die jeweils aktivierte Koroutine automatisch wieder zur Verfügung steht.

Exkurs Einen wesentlichen Anteil an den Eigenvariablen einer Koroutine hat der ↑Prozessorstatus, genauer die Prozessorregister, die von der Koroutine im Moment des ↑Koroutinenwechsels gerade belegt sind (↑aktives Register). Dies bedeutet eine mehr oder weniger umfangreiche Umspeicherung von ↑Daten, je nachdem, wie viele diese Register im Moment des Wechsels für die Koroutine gerade aktiv sind.
Das Wissen über die gerade aktiven Register hat der ↑Kompilierer. Ist in der Programmiersprache ein Koroutinenkonzept verankert, so wird der Kompilierer bei der ↑Übersetzung nur die zur Sicherung und Wiederherstellung wirklich erforderlichen ↑Maschinenbefehle absetzen, nämlich nur um die Inhalte der im Moment des Koroutinenwechsels aktiven Register bewahren. Allerdings kennt ↑C kein solches Konzept, daher ist die Sicherung und Wiederherstellung dieser Register selbst zu programmieren. Wie viele Register dabei zu berücksichtigen sind, hängt davon ab, welcher Abstraktionsebene eine Koroutine jeweils zugeordnet ist.
Typischerweise sind hier zwei Ebenen relevant, nämlich die der höheren Programmiersprache und die des ↑Maschinenprogramms. Erfolgt der Wechsel zwischen Koroutinen derselben höheren Programmiersprache, gibt die ↑Aufrufkonvention dieser Sprache den Hinweis über die unbedingt zu sichernden und wiederherzustellenden Register. Anderenfalls muss das

↑Programmiermodell des Prozessors herangezogen werden. Daher bietet sich die parametergesteuerte Registerbehandlung an, wie folgendes Beispiel einer in C für ↑x86 formulierten Sicherungsoperation zeigt:

```
inline void unload(trim_t trim, void *tank) {
    if ((trim & ~CDECL) == INNER) {                /* use run-time stack? */
        if (!(trim & CDECL)) {       /* yes, consider volatile registers? */
            asm volatile (               /* yes, backup to run-time stack */
                "pushl %eax\n\t"
                "pushl %ecx\n\t"
                "pushl %edx");
        }
        asm volatile ( /* backup non-volatile registers to run-time stack */
            "pushl %ebx\n\t"
            "pushl %ebp\n\t"
            "pushl %esi\n\t"
            "pushl %edi");
    } else if ((trim & ~CDECL) == PLAIN) {           /* use buffer store? */
        asm volatile (              /* yes, backup non-volatile registers */
            "movl %%edi, 0(%0)\n\t"
            "movl %%esi, 4(%0)\n\t"
            "movl %%ebp, 8(%0)\n\t"
            "movl %%ebx, 12(%0)" : : "r" (tank));
        if (!(trim & CDECL)) {            /* consider volatile registers? */
            asm volatile (                 /* yes, backup to buffer store */
                "movl %%edx, 16(%0)\n\t"
                "movl %%ecx, 20(%0)\n\t"
                "movl %%eax, 24(%0)" : : "r" (tank));
        }
    }
}
```

Dabei ist der ↑Datentyp `trim_t` wie folgt definiert:

```
typedef enum trim {
    NAKED, INNER, PLAIN, CDECL=(1<<31)
} trim_t;
```

Die Umspeicherung von Registerinhalten entlastet (`unload`) den Prozessor. Dazu sind letztlich drei Varianten beschrieben: `INNER`, sichert in den Laufzeitstapel; `PLAIN`, sichert in einen Pufferspeicher (`tank`); weder noch beziehungsweise `NAKED`, sichert nirgendwo hin. Darüber hinaus wird spezifiziert, ob entsprechend der Aufrufkonvention von C (↑cdecl) vorgegangen werden soll (`CDECL`). Ist dies der Fall, betrifft die Entlastung des Prozessors nur einen Teil seiner Register (↑*non-volatile register*). Anderenfalls werden zusätzlich noch die Inhalte der flüchtigen Register (↑*volatile register*) gesichert.
Die Anweisungen sind prozessorabhängig. Durch Konstantenfaltung (*constant folding*) wertet der Kompilierer die Fallunterscheidungen bereits bei der Übersetzung des ↑Unterprogramms (`unload`) aus und generiert eine ↑Aktionsfolge, die frei von Programmverzweigungen ist. Darüber hinaus hinterlässt inzeiliges assemblieren (*inline assembly*) keine weiteren Spuren im Maschinenprogramm.
Die inverse Operation folgt demselben Muster, hier sei jedoch nur ihre Signatur angegeben:

```
inline void reload(trim_t trim, void *tank) { ... }
```

Beide Operationen rahmen typischerweise die zentrale, für gewöhnlich von allen Koroutinen verwendete Anweisung zur Ausführungsunterbrechung und damit zur Prozessorabgabe ein. Ein Beispiel, bei dem dann der Zustand der nichtflüchtigen Register im Laufzeitstapel gesi-

chert vorliegt, zeigt folgender Programmausschnitt:

```
unload(INNER|CDECL, 0);       /* backup non-volatile registers, use stack */
last = resume(next);                                    /* switch coroutines */
reload(INNER|CDECL, 0);       /* define non-volatile registers, use stack */
```

Zu beachten ist hierbei, dass eine Koroutine (`last`) die Funktion `resume` aufruft und eine andere Koroutine (`next`) aus dem Aufruf, den sie irgendwann vorher selbst getätigt hat, zurückkehrt. Der Funktionswert identifiziert dann die Koroutine, die `resume` gerade eben aufgerufen hatte. Jede dieser Koroutinen sichert ihre aktiven Register nicht nur selbst, sondern definiert sie auch für sich selbst. Keine Koroutine muss damit die aktiven Register der jeweils anderen Koroutine kennen.

Koroutinenwechsel Veränderung in dem ↑Programmablauf, Wechsel von einer ↑Koroutine zu einer anderen Koroutine: ↑*resume*. Je nach Art der Koroutine erfolgt dieser Wechsel unterschiedlich, und zwar abhängig von dem zur Adressierung eines ↑Handlungsstrangs verwendeten ↑Prozessorregister im Steuerwerk einer ↑CPU. Für eine ↑primitive Koroutine wird der Handlungsstrang durch den ↑PC adressiert, der Wechsel verläuft direkt. Im Gegensatz dazu ist der Schalthebel für eine ↑komplexe Koroutine (um letztlich ebenfalls den PC umzusetzen) der ↑SP, der Wechsel verläuft indirekt. Passend zur Wechseltechnik ist für einen initialen ↑Koroutinenaufbau zu sorgen, der es beispielsweise einer gewöhnlichen ↑Routine gestattet, als Koroutine gleichgestellt mit anderen Routinen ausgeführt werden zu können. In beiden Fällen ist die Signatur einer entsprechenden Operation jedoch gleich, beispielsweise:

```
coroutine_t __attribute__((fastcall)) resume(coroutine_t);
```

Das angegebene Attribut (`fastcall`) spezifiziert die Parameterübergabe durch Verwendung eines ↑Prozessorregisters (hier: `ecx`, ↑x86).
Um den Handlungsstrang einer zeitweilig unterbrochenen Koroutinenausführung wieder aufnehmen zu können, muss die Wechseloperation jede Koroutine dazu veranlassen, eine ↑Handhabe zur Wiederaufnahme der Ausführung eigenständig zu sichern und wiederherzustellen. In Bezug auf den bei Bedarf zusätzlich noch zu berücksichtigenden ↑Prozessorstatus ist jedoch auch dessen fremdgesteuerte Wiederherstellung möglich: die den Wechsel auslösende Koroutine setzt den Prozessorstatus der nachfolgenden Koroutine auf, bevor letztere ihren Handlungsstrang wieder aufnimmt. Das bedeutet aber auch, dass dann in dieser einen Wechseloperation Wissen über den Prozessorstatusumfang der jeweils nachfolgenden Koroutine verankert sein muss. Dieses Wissen liegt für gewöhnlich nicht vor, weshalb in dem Fall und unter Kontrolle der jeweils vorangehenden Koroutine immer der komplette Prozessorstatus gesichert und wiederhergestellt werden müsste.
Da Koroutinen von den Prozessorregistern unterschiedlichen Gebrauch machen können, ist auch nur die Sicherung und Wiederherstellung der Inhalte jener Prozessorregister nötig, die die jeweilige Koroutine im Moment ihrer Ausführungsunterbrechung belegt. Der nachfolgend gezeigte `resume`-Mechanism trägt diesem Aspekt Rechnung.

Primitive Koroutine Ohne entsprechende Eigenschaften der CPU (↑orthogonaler Befehlssatz) bildet die Operation eine ↑Aktionsfolge, was zudem für gewöhnlich die Formulierung in ↑Assembliersprache erfordert. Ein Beispiel für ↑GAS und x86 ist nachfolgend skizziertes ↑Unterprogramm, das eine Funktion implementiert, die (a) zu einer Koroutine hinschaltet, deren Adresse in einem Prozessorregister enthalten ist und (b) als Wert die Fortsetzungsadresse des Handlungsstrangs der wegschaltenden Koroutine liefert:

```
resume:                 # use subroutine call: last = resume(next)
    popl %eax           # return address becomes function return value
    jmp *(%ecx)         # continue other coroutine
```

Insgesamt sind damit jedoch wenigstens vier Maschinenbefehle für den Wechsel von solch einem Handlungsstrang notwendig, nicht nur zwei. Die beiden zusätzlichen Befehle bewirken den Unterprogrammaufruf:

```
    movl <next>, %ecx     # pass continuation address as parameter
    call resume           # and perform the coroutine switch
```

Dabei steht `<next>` für einen Operanden mit der Fortsetzungsadresse des Handlungsstrangs einer Koroutine, zu der hingeschaltet werden soll. Die ↑Gemeinkosten dieser Nachbildung einer ↑Elementaroperation zum Ablaufwechsel sind vergleichsweise hoch. Alternativ kann diese Operation als ↑Makrobefehl ausgelegt werden, um den Aufwand auf zwei Maschinenbefehle zu begrenzen:

```
.macro resume next        # input next (continuation address), spoil %eax
    movl $1f, %eax        # generate output value: continuation address
    jmp *(\next)          # actually switch to (i.e., continue) coroutine
    .p2align 3            # enforce aligned continuation address
1:                        # point of return, i.e., resumption
.endm                     # %eax holds the continuation address
```

Der erste (`movl`) Befehl liefert die Fortsetzungsadresse (`$1f`, d.h., Marke 1 vorwärts) der laufenden Handlungsstrangs als Ausgabeparameter (`%eax`). Diese Adresse ist der vom ↑Binder berechnete Wert einer ↑Sprungmarke (`1:`), nämlich die Stelle, an der die wegschaltende Koroutine später die Wiederaufnahme ihrer Ausführung erwartet. Der zweite (`jmp`) Befehl überträgt die Kontrolle an die Koroutine, deren Fortsetzungsadresse als Parameter (`next`) übergeben wurde. Die in diesem Makro definierte Befehlsfolge lässt sich nahezu direkt in ↑C übernehmen, und zwar durch Anweisungen zum inzeiligen Assemblieren (*inline assembly*):

```
inline coroutine_t resume(coroutine_t next) {
    register coroutine_t last asm("eax");     /* enforce register variable */
    asm volatile(                                 /* start inline assembly */
        "movl $1f, %%eax\n\t"                       /* define return value */
        "jmp *%0\n\t"                            /* switch to next coroutine */
        ".p2align 3\n"                /* enforce alignment of following text */
        "1:"                                           /* come back to here */
        : : "g" (next) : "%eax");                /* specify input parameter */
    return last;                        /* deliver pointer to last coroutine */
}
```

Vorteil dieser Lösung ist die nahtlose Einbindung von Maschinenbefehlen (GAS/x86) in eine in Hochsprache (C) formulierte ↑Routine. Dabei setzt der Kompilierer (`gcc`) die in der Funktion (`resume`) verwendeten Variablen (`next`, `last`) mit passender ↑Adressierungsart automatisch als Operanden ein.
Die Implementierungen greifen das Konzept des ↑Verbindungsregisters auf, um die Fortsetzungsadresse einer Koroutine zu speichern. Mehr Statusinformationen sind auch für eine gewöhnliche Routine nicht notwendig, um nämlich die Stelle zu vermerken, an der sie die Kontrolle an eine andere Routine durch einen Maschinenbefehl zum Unterprogrammaufruf (`call`) abgegeben hat und zurückerhalten kann. Wie auch im Falle herkömmlicher Unterprogramme wird der wirkliche Datenzustand davon abhängig sein, welcher Funktion die betreffende Koroutine dient. Ist diese Funktion der ↑Prozesswechsel, muss typischerweise ein erweiterter ↑Koroutinenstatus berücksichtigt werden. Darüber hinaus ist, jeweils passend zu den hier gezeigten Funktionen zum Koroutinenwechsel, für die Einrichtung einer Koroutine derart zu sorgen, dass zu dieser auch umgeschaltet werden kann, obwohl sie noch keine Möglichkeit hatte, selbst einen Koroutinenwechsel durchzuführen (↑Koroutinenaufbau).

Komplexe Koroutine Im Gegensatz zum vorangegangem Modell, erfolgt hier die Umschaltung des ↑Stapelzeigers, um zwischen Handlungssträngen verschiedener Koroutinen hin und her zu wechseln. Beispiel einer entsprechenden Operation (x86) ist nachfolgend skizziertes Unterprogramm. Die darin beschriebene Funktion (a) schaltet hin zu einer Koroutine, deren Stapelzeiger in einem Prozessorregister enthalten ist und (b) liefert als Wert den Stapelzeiger dieser zuvor eben noch gelaufenen Koroutine:

```
coroutine_t __attribute__((fastcall)) resume(coroutine_t next) {
    coroutine_t last;                          /* return value placeholder */
    asm volatile(                                 /* start inline assembly */
        "movl %%esp, %0\n\t"                        /* define return value */
        "movl %1, %%esp"                           /* switch run-time stack */
        : "=r" (last) : "r" (next));  /* specifiy input/output constraints */
    return last;                      /* deliver pointer to last coroutine */
}
```

Wird dieses Unterprogramm aufgerufen, gelangt automatisch die Fortsetzungsadresse des Handlungsstrangs der wegschaltenden Koroutine auf den Stapel: diese ist nämlich die beim Aufruf hinterlassene Rücksprungadresse. Die ↑Speicherstelle, an der diese Adresse vom Prozessor (x86) abgelegt wurde, identifiziert der Stapelzeiger (`esp`). Der für die zu aktivierende Koroutine gültige Stapelzeiger wird vor dem Aufruf noch als ↑tatsächlicher Parameter in einem ↑Prozessorregister (`fastcall`: `ecx`) übergeben.
Zu beachten dabei ist die Bedeutung des Rückgabewertes der Funktion `resume`. Dieser Wert entspricht dem Stapelzeiger der soeben weggeschalteten Koroutine und identifiziert damit die Stelle, an der für gewöhnlich auch der ↑Laufzeitkontext dieser Koroutine gesichert vorliegt. Mit diesem Rückgabewert als Parameter für einen späteren `resume`-Aufruf wird die betreffende Koroutine wieder in den Zustand gebracht, den sie zuvor inne hatte. Im Gegensatz zur primitiven Koroutine, deren Handhabe ein Befehlszählerwert ist, ist für den hier gezeigten Koroutinenwechsel allerdings auch eine dazu passende Einrichtungsprozedur zu druchlaufen (↑Koroutinenaufbau).

kritischer Abschnitt (en.) ↑*critical section*. Bezeichnung für eine bestimmte Region in einem ↑Programm, die eine ↑Wettlaufsituation gleichzeitiger ↑Prozesse möglich macht und dadurch eine in sich inkonsistente ↑Aktionsfolge hervorbringen kann. Letztere resultiert in falsche Berechungsergebnisse, widersprüchliche ↑Daten oder gefährliche Kontrollflüsse (↑*lost wake-up*). Der ursprünglichen Bedeutung nach darf diese Region zu jedem Augenblick nur von höchstens einen Prozess besetzt sein (↑*mutual exclusion*). Gemeinhin ist jedoch jede Form von ↑Nebenläufigkeitssteuerung zulässig, solange damit zu jeder Zeit ein in sich konsistenter ↑Programmablauf in dieser Region sichergestellt ist.

Kurvenschreiber ↑Peripheriegerät zur Ausgabe von Funktionsgraphen, technische Zeichnungen oder Grafiken; X-Y-Schreiber. Messgerät mit einer Schreibeinrichtung, um den zeitlichen Verlauf eingetragener Messgrößen festzuhalten. Die durch einen zugehörigen ↑Gerätetreiber zur Ausgabe nacheinander mittels ↑Ein-/Ausgaberegister bereitgestellten ↑Daten definieren die jeweilige Lage der Schreibeinrichtung in einer Ebene (Koordinate). Je nach Geräteart ist die Schreibeinrichtung ein Stift (Papier), Messer (Folie), Laserstrahl (Folie) oder Lichtkopf (Film). Der Gerätetreiber ist für gewöhnlich ein ↑Unterprogramm von einem Steuerprogramm zur Organisation und Überwachung des Rechnerbetriebs (↑*resident monitor*). Falls ↑abgesetzter Betrieb geführt wird, ist das Gerät selbst am ↑Satellitenrechner angeschlossen, ansonsten am ↑Hauptrechner.

kurzfristige Einplanung (en.) ↑*short-term scheduling*. ↑Ablaufplanung auf kurze Sicht, typischerweise im Mikro- oder Millisekundenbereich. Eine ↑Systemfunktion der Prozessverwaltung (*on-line* ↑*scheduling*). Festlegung der Reihenfolge, nach der die ↑Einlastung der CPU geschehen soll. Obligatorische Maßnahme zur ↑Simultanverarbeitung — mehr dazu aber erst in VL 9.2.

Ladeadresse Bezeichnung für eine ↑Adresse, an der das zugehörige Objekt (↑Modul, ↑Unterprogramm, ↑Exemplar eine Datentyps) im für ein ↑Maschinenprogramm einzurichtenden (realen, logischen, virtuellen) ↑Adressraum zu platzieren ist. Ein solche Adresse wird zur ↑Bindezeit des Maschinenprogramms für jedes einzelne darin enthaltende Objekt (↑Text, ↑Daten, ↑BSS) bestimmt.

Lademodul ↑Datei mit der Eingabe für einen ↑Lader, Ausgabedatei von einem ↑Binder.

Laden Vorgang, um ↑Text oder ↑Daten von einem ↑Datenträger in den ↑Arbeitsspeicher zu übertragen. Dabei ist ein ↑verschiebender Lader weitestgehend frei in der Wahl der ↑Adresse im Arbeitsspeicher, an der das zu ladende ↑Maschinenprogramm platziert werden kann. Ein ↑bindender Lader hat zunächst die gleiche Wahl und sorgt aber zusätzlich noch dafür, jede eventuell noch bestehende ↑unaufgelöste Referenz zu beseitigen. Dazu wird die ↑Bindung mit der ↑Ladeadresse der referenzierten ↑Entität komplettiert, wozu letztere gegebenenfalls noch selbst in den Arbeitsspeicher zu bringen ist. Ohne solch eine Wahl ist der ↑Lader an die durch das ↑Lademodul sodann vorgegebene Ladeadresse gebunden. Passend zur ↑Betriebsart wird der ↑Adressraum für das Maschinenprogramm etabliert, eine ↑Prozessinkarnation eingerichtet, für die Zuteilung weiterer ↑Betriebsmittel gesorgt und der zugehörige ↑Prozess bereitgestellt (↑*scheduling*).

Lader ↑Systemfunktion, die das durch ein ↑Lademodul beschriebene ↑Maschinenprogramm in den ↑Arbeitsspeicher platziert und abschließend einen ↑Prozess damit verknüpft. Dieser Prozess existiert bereits und hat den ↑Systemaufruf zum Laden selbst abgesetzt (UNIX `exec`) oder er wird beim Ladevorgang erzeugt (VMS `run`, Windows `spawn`).

Ladestrategie Verfahrensweise nach der das Moment des Zugriffs auf ein im ↑Umlagerungsbereich von einem ↑Prozess belegtes ↑Umlagerungsmittel bestimmt wird. Ein solcher Zugriff impliziert die Einlagerung des Umlagerungsmittels in den ↑Hauptspeicher. Diese Einlagerung geschieht bei Bedarf (*on demand*) oder im Voraus (*anticipatory*), das heißt, entweder bei oder vor dem Zugriff auf das ausgelagerte Umlagerungsmittel. Im ersten Fall kommt zur Ausführung von einem ↑Maschinenbefehl durch die ↑CPU eine ↑virtuelle Adresse zur Geltung, die zwar gültig ist, jedoch nicht auf ↑Text oder ↑Daten im Hauptspeicher abgebildet werden kann. Es kommt zu einen ↑Hauptspeicherfehlzugriff. Bildet ↑virtueller Speicher den Bezug, betrifft der Fehlzugriff in aller Regel eine ↑Seite und es kommt zur ↑Seitenumlagerung durch das ↑Betriebssystem (↑*pager*). Der Fehlzugriff kann sich jedoch auch auf ein ↑Text- oder ↑Datensegment beziehen, das noch nicht in den ↑Prozessadressraum eingebunden wurde (↑*dynamic binding*). In dem Fall wird ein ↑dynamischer Binder im Betriebssystem das referenzierte ↑Segment komplett (↑*segmentation*) oder teilweise (↑*segmented paging*) in den Hauptspeicher bringen und entsprechend im Umlagerungsbereich verbuchen. Erstere Variante ist gleichsam ein Beispiel für die Einlagerung im Voraus, im Falle seitennummerierter Segmentierung kann dies insbesondere den ↑Seitenvorabruf für alle Seiten des Segments auslösen. Ist die ↑Arbeitsmenge des den Fehlzugriff auslösenden Prozesses bekannt, werden wenigstens alle zu dieser Menge zählenden Seiten eingelagert. Damit die Einlagerung vollzogen werden kann, bestimmt die ↑Platzierungsstrategie den dafür benötigten ↑Speicherbereich im Hauptspeicher. Ist nicht genügend freier Hauptspeicher verfügbar, kommt entweder die ↑Ersetzungsstrategie ins Spiel oder der Prozess wird vorläufig durch den ↑Planer suspendiert (↑Prozesswechsel), bis ausreichend Hauptspeicherplatz zur Verfügung steht.

Ladezeit Zeitpunkt zu dem ein ↑Programm zur Ausführung in den ↑Arbeitsspeicher geladen wird oder Zeitspanne eben dieses Ladevorgangs.

langfristige Einplanung (en.) ↑*long-term scheduling*. ↑Ablaufplanung auf lange Sicht, typischerweise im Sekunden- oder Minutenbereich. Eine ↑Systemfunktion der Lastkontrolle im ↑Rechensystem, steuert den Grad an ↑Mehrprogrammbetrieb und bestimmt dazu den Zeitpunkt der Zulassung von einem ↑Prozess und damit den Moment seiner Teilnahme am Rechenbetrieb.

Für gewöhnlich greift diese Form von ↑Einplanung bei der ↑Anmeldung, zur ↑Umlagerung kompletter ↑Programme (↑*swapping*) oder zum Zwecke der ↑Verklemmungsvermeidung.

Lastausgleich Berechnungen beziehungsweise Aufträge auf mehr als einen ↑Rechenkern von einer ↑CPU oder einem ↑Multiprozessor verteilen. Setzt ↑Parallelverarbeitung voraus.

latch (dt.) Auffangregister; zustandsgesteuertes FlipFlop (1-Bit ↑Speicher).

Latenz Vorhandensein einer Sache, die noch nicht in Erscheinung getreten ist (Duden). Im Kontext von einem ↑Betriebssystem beispielsweise die Sache, dass eine ↑Unterbrechungsbehandlung erfolgen wird, da die ↑CPU eine ↑Unterbrechungsanforderung erkannt hat, diese Behandlung wegen einer (implizit durch die CPU oder explizit durch das Betriebssystem) gesetzten ↑Unterbrechungssperre allerdings noch nicht möglich ist. Analog dazu die Sache, dass ein ↑Prozesswechsel geschehen wird, da der ↑Planer den Vorrang für einen ausgelösten ↑Prozess festgestellt hat, die dafür notwendige ↑Einlastung der CPU wegen einer (bedingt durch das ↑Operationsprinzip des Betriebssystems, implizit oder explizit) gesetzten ↑Verdrängungssperre jedoch unterbunden ist. Allgemein jedes ↑Ereignis, das (gemäß ↑Programm) als logische Folge eines vorausgegangenen Ereignisses absehbar ist, aber auf Grund eines bestimmten Systemzustands noch nicht sofort herbeigeführt werden darf. Die Zeit zwischen dem ursächlichen und dem wirkenden Ereignis ist die ↑Latenzzeit.

Latenzzeit Zeitspanne einer technisch bedingten Verzögerung.

Laufzeit Zeitpunkt zu dem ein ↑Prozess stattfindet beziehungsweise Zeitspanne einer ↑Aktion, ↑Teilaufgabe, ↑Aufgabe oder ↑Programmablaufs.

Laufzeitkontext Gesamtheit an Zustandsdaten der (realen/virtuellen) Maschine, die einen ↑Prozess definieren. Neben dem ↑Laufzeitstapel ist ein weiteres wesentliches Bestandteil dieser Daten der ↑Prozessorstatus. Kommt es zur Unterbrechung eines Prozesses, spiegelt der in dem Moment gültige Prozessorstatus den Kontext zur späteren Fortsetzung eben dieses Prozesses wider. Um den Prozess fortsetzen zu können, als wenn seine Unterbrechung nie geschehen wäre, ist der Prozessorstatus invariant zu halten. Dies wird erreicht durch eine zeitweilige ↑Zustandssicherung in eine dem Prozess eigene Datenstruktur im ↑Hauptspeicher: Der Prozessorstatus wird bei der Unterbrechung in diese Datenstruktur gesichert und zur Fortsetzung wieder daraus geladen.

Laufzeitstapel ↑Stapelspeicher von einem ↑Handlungsstrang; dient vornehmlich der zeitweiligen Lagerung lokaler Daten.

Laufzeitsystem Menge von Funktionen, die in Abhängigkeit von der jeweils verwendeten Programmiersprache eine Ablaufunterstützung für ein ↑Maschinenprogramm bildet. Typische Beispiele solcher Funktionen für ↑C sind formatierte Ein-/Ausgabe (`scanf(3)`, `printf(3)`), kopieren von Speicherbereichen (`memcpy(3)`, `strcpy(3)`), Verarbeitung von Zeichenketten (`string(3)`), Ein-/Ausgabe von Binärdatenströmen (`fread(3)`), Handhabung von Signalen (`signal(3)`), Kontrollfluss- und Kontextverwaltung (`setjmp(3)`), Verwaltung von ↑Haldenspeicher (`malloc(3)`) und mehr. Technisch sind die Funktionen jeweils als ↑Unterprogramm aus- und in einer ↑Programmbibliothek abgelegt (u.a. ↑libc). Nicht wenige dieser Funktionen dienen insbesondere der Interaktion mit dem ↑Betriebssystem und versuchen gerade in dem Zusammenhang, die durch einen ↑Systemaufruf bedingte ↑Latenzzeit zu kaschieren.

LCFS Abkürzung für (en.) *last come, first served*, (dt.) wer zuletzt kommt, mahlt zuerst; Reihungsverfahren, gleichbedeutend mit ↑LIFO.

Leerbefehl (en.) ↑*no-op instruction*. Bezeichnung für einen wirkungslosen ↑Maschinenbefehl, der keinen Effekt hat, außer Platz im ↑Hauptspeicher zu belegen und ↑Ausführungszeit zu benötigen.

Leerlauf (en.) ↑*idle*. Zustand der Untätigkeit, nämlich wenn auf einem ↑Prozessor kein ↑Prozess stattfindet. Entweder wird der Prozessor gerade eben durch einen auf einem anderen Prozessor stattfindenden Prozess hochgefahren, durchläuft seine Initialisierungsprozedur, und hat vom ↑Planer noch keinen Prozess zugewiesen bekommen (↑Multiprozessor) oder im Moment der Blockierung des auf ihm stattfindenden Prozesses steht kein anderer Prozess zur ↑Einlastung mehr bereit.

Der Prozessor kann nur noch durch einen externen Prozess aus diesen Zustand befreit werden, indem ihm „von außen“ ein Prozess zur Einlastung bereitgestellt wird. Dies kann durch eine ↑Unterbrechungsanforderung geschehen, die die Deblockierung eines blockierten Prozesses bewirkt oder, vorausgesetzt der Prozessor ist aktiv untätig, indem er auf der ↑Bereitliste plötzlich einen Prozess vorfindet, den der Planer von einem anderen Prozessor aus dort abgelegt hat. Im letzteren Fall ist ein ↑Leerlaufprozess aktiv wartend darauf, dass die Bereitliste wieder gefüllt wird (↑*busy waiting*). Normalerweise jedoch wird ein Prozessor in solch einer Situation in den ↑Schlafzustand versetzt, um nicht unnötig das ↑Rechensystem zu strapazieren, Energie zu verbrauchen, Abwärme zu produzieren — damit Unkosten zu verursachen und die Umwelt zu belasten.

Exkurs Auf den ersten Blick erscheint der Vorgang, einen Prozessor in den Schlafzustand zu versetzen, keine besonders komplizierte ↑Aktion zu sein. Der Übergang in diesen Zustand ist eine *bedingte Anweisung*, die den Schlafzustand nur aktiviert, wenn die Bedingung dafür noch gilt. Gegebenenfalls kann diese *Schlafbedingung* im Moment des Schlafengehens aber bereits entkräftet worden sein, ohne dass der diesen Übergang steuernde Prozess etwas davon mitbekommen haben muss (↑*lost wake-up*). Gleichzeitige Prozesse, die Auswirkungen auf die Aktivierung des Schlafzustands des Prozessors haben können, sind daher unbedingt zu koordinieren:

```
void relax(void *sign) {     /* retire processor, prepare for sleep state */
    arena(ENTER);                                        /* join danger zone */
    if (*(long *)sign == 0)                             /* still out of work? */
        drift();                                     /* yes, let CPU drift off */
    arena(LEAVE);                                        /* quit danger zone */
}
```

Die hier gezeigte Klammerung der dem kritischen Pfad entsprechenden „Kampfbahn“ (`arena`) beugt der Gefahr vor, die ↑CPU möglicherweise in den ewigen Schlaf zu treiben (`drift`). Den Schlafzustand bewirkt typischerweise ein spezieller ↑Maschinenbefehl (z.B. `hlt`, ↑x86), dessen Ausführung die CPU nämlich erst mit der nächsten Unterbrechungsanforderung beendet. Mit Ausführungsbeginn dieses Befehls hebt die CPU zwar implizit eine eventuell bestehende ↑Unterbrechungssperre auf, allerdings ist damit noch längst nicht garantiert, dass die ↑Peripherie auch eine weitere Unterbrechungsanforderung zur CPU sendet. Sollte der CPU also eine solche Anforderung entgangen sein, schläft sie möglicherweise für immer — es sei denn, das ↑Betriebssystem nutzt einen periodischen ↑Zeitgeber, der jedoch nicht für alle ↑Betriebsarten nötig und somit alles andere als Standard ist.
Gängige Lösung zur Absicherung dieses Pfads ist es, beim Betreten (`ENTER`) eine *totale Unterbrechungssperre* zu setzen und diese beim Verlassen (`LEAVE`) wieder aufzuheben. Die entsprechenden Anweisungen dazu zeigt nachfolgendes Beispiel:

```
inline void arena(plan_t plan) {                  /* bracketing construct */
    if (plan == ENTER)                                  /* start of path? */
        asm volatile ("cli": : : "cc");                /* yes, disable IRQ */
    else if (plan == LEAVE)                               /* end of path? */
        asm volatile ("sti": : : "cc");                 /* yes, enable IRQ */
}
inline void drift() {                    /* suspend CPU, force sleep state */
    asm volatile ("hlt");
}
```

Da der Haltebefehl (in `drift`: `hlt`) trotz Sperre eine Unterbrechungsanforderung zulässt und bei korrekter Klammerung (s.o., `relax`) bis dahin der Prozess nicht unterbrochen werden kann, ist softwareseitig dafür Sorge getragen, den erwarteten Weckruf nicht zu verpassen. Dies setzt allerdings voraus, dass bei ↑Flankensteuerung die Hardware wenigstens eine solche zwischenzeitlich eingehende Anforderung zwischenspeichert.

Alternativ ist ein *sperrfreies Verfahren* möglich, das beim Betreten des Pfads einen *Rücksetzpunkt* definiert und diesen beim Verlassen des Pfads wieder ungültig macht. Ist ein solcher Rücksetzpunkt im Moment einer dann jederzeit möglichen Unterbrechungsanforderung gültig und kommt es im Verlauf der ↑Unterbrechungsbehandlung zur Bereitstellung eines Prozesses, setzt der unterbrochene Leerlaufprozess bei Rückkehr aus dem ↑Unterbrechungshandhaber genau an dieser Stelle wieder auf, nicht etwa an seiner Unterbrechungsstelle. Als Folge dieses Umstands wertet der Leerlaufprozess die Wartebedingung erneut aus, stellt fest (`*sign` $\neq$ `0`), die CPU nicht in den Schlafzustand versetzen zu müssen und verlässt seinen Ruheplatz (`relax`). Dieser Ansatz bedeutet letztlich, beim Betreten des Pfads eine ↑Koroutine einzurichten und eine dazu passende ↑Handhabe bekanntzugeben, um darüber die Unterbrechungsbehandlung mit einem ↑Koroutinenwechsel abschließen zu können:

```
inline void arena(plan_t plan) {                    /* bracketing construct */
    extern coroutine_t *idle;                           /* coroutine handle */
    if (plan == ENTER)                                    /* start of path? */
        asm volatile (                             /* yes, define restart point */
            "pushl $1f\n\t"                           /* setup point of restart */
            "movl %%esp, %0\n"                        /* setup coroutine handle */
            "1:"                                  /* come here upon resume(idle) */
            : "=m" (idle));
    else if (plan == LEAVE)                                   /* end of path? */
        if (FAS(&idle, 0))                 /* yes, cancel restart; interrupted?*/
            asm volatile ("addl $4, %esp");    /* no, remove restart point */
}
```

Beim Verlassen des Pfads muss ein ungültiger Rücksetzpunkt hinterlassen werden, um nicht fälschlicherweise noch bei der nächsten Unterbrechungsanforderung einen Koroutinenwechsel hin zum Leerlaufprozess abzusetzen. Wurde der Leerlaufprozess durch einen Koroutinenwechsel fort- beziehungsweise zurückgesetzt, ist die Koroutinenhandhabe bereits ungültig (`idle` $=$ `0`) und es ist keine „Aufräumarbeit“ zu leisten. Anderenfalls (`idle` $\neq$ `0`) wurde weder der Leerlaufprozess unterbrochen noch die CPU in den Schlafzustand versetzt, so dass der noch bestehende Hinweis auf den Rücksetzpunkt entfernt werden muss. Um einer möglichen ↑Wettlaufsituation vorzubeugen, ist das Lesen und Löschen des Wertes der Koroutinenhandhabe als ↑atomare Operation durchzuführen (↑FAS).
Das Beispiel geht davon aus, dass dem Koroutinenwechsel eine *komplexe Koroutine* zugrunde liegt. Stößt die Unterbrechungsbehandlung auf den Sonderfall, zum Leerlaufprozess zurückzusetzen (nämlich wenn gilt: `idle` $\neq$ `0`), ist als letzte Anweisung lediglich der Funktionsaufruf `resume(idle)` auszuführen. Da jedoch der Unterbrechungshandhaber auch in dem Fall immer nur verlassen und nicht als Koroutine unterbrochen wird, ist nur für die Fortsetzung der Koroutine des Leerlaufprozesses zu sorgen. Folgendes Beispiel skizziert die dazu erforderlichen letzten Maschinenbefehle einer Unterbrechungsbehandlung:

```
    ...                 # at this point, the processor status was restored
    cmpl $0, idle       # idle process interrupted?
    je 1f               # no, perform normal exit
    movl idle, %esp     # yes, switch stack to idle-process coroutine
    movl $0, idle       # avoid spurious resume: cancel restart point
    ret                 # resume execution of idle-process coroutine
1:                      # come here for normal exit
    iret                # return from interrupt
```

Zu beachten ist, dass all diese Maßnahmen dann als Teil einer Phase stattfinden, in der der Prozessor normalerweise untätig ist — somit die im Vergleich zum sperrenden Ansatz etwas höheren ↑Gemeinkosten kaum oder nicht ins Gewicht fallen und damit die erreichte Sperrfreiheit nicht zwingend zur Erhöhung der ↑Betriebslast beiträgt.

Leerlaufprozess (en.) ↑*idle process*. Bezeichnung für einen ↑Prozess, der den ↑Leerlauf von einem

↑Prozessor kontrolliert; der zwar logisch untätig ist, aber physisch für das System tätig sein kann. Es gibt zwei grundsätzlich verschiedene Modelle für einen solchen Prozess.
In dem einen Fall steht für den Leerlauf eine eigene ↑Prozessinkarnation zur Verfügung. Entsprechend wird der Leerlauf sodann durch einen physischen ↑Prozesswechsel gestartet, und zwar immer dann, wenn im Moment der Blockierung eines Prozesses die ↑Bereitliste keinen anderen Prozess für den Prozessor enthält. In dem Modell zieht der Leerlauf genau genommen immer zwei physische Prozesswechsel nach sich: erstens vom blockierenden zum leerlaufenden Prozess und zweitens vom leerlaufenden zum bereitgestellten Prozess. Dies geschieht auch dann, wenn der bereitgestellte Prozess sich als derjenige erweist, der vorher zum leerlaufenden Prozess gewechselt ist.
In dem anderen Fall übernimmt immer jener Prozess die Rolle der Leerlaufkontrolle, der jüngst zurückliegend gerade blockiert ist und in dem Moment keinen anderen für seinen Prozessor lauffähigen Prozess in der Bereitliste vorfindet. Damit kann ein beliebiger Prozess in diese Rolle gebracht werden, die Identität des leerlaufenden Prozesses ist nicht eindeutig bestimmt. Jedoch eröffnet sich so die Option, dem physischen Prozesswechsel vorzubeugen, wenn nämlich der Prozess in seiner Leerlaufphase deblockiert wird. Zu beachten ist, dass lediglich ein logischer Prozesswechsel vollzogen wird, nämlich vom ↑Prozesszustand laufend nach untätig: ein physischer Wechsel der Prozessinkarnation erfolgt nicht.

Exkurs Für gewöhnlich versetzt der den Leerlauf kontrollierende Prozess seinen Prozessor in einen speziellen Bereitschaftsbetrieb (*stand-by mode*), der zur Absenkung des Energiebedarfs der Hardware beiträgt. Damit geht faktisch der Stillstand (*stall*) dieses Prozesses einher. Die Maßnahmen dafür spielen sich auf zwei verschiedenen Ebenen ab, der prozessbezogenen Ebene einerseits und der prozessorbezogenen Ebene andererseits. Ein Beispiel für die prozessbezogene Ebene ist nachfolgend skizziert:

```
void stall() {                              /* process reduced to inaction */
    process_t *self = being(ONESELF);
    state(&self->mood, IDLE);                           /* enter idle state */
    while (!ahead(labor()))                                 /* out of work? */
        relax(batch(labor()));              /* yes, be asleep at the switch */
    state(&self->mood, CLEAR|IDLE);                     /* leave idle state */
}
```

Die gezeigte Operation (`stall`) bringt den gegenwärtig stattfindenden Prozess (`ONESELF`) dazu, den Leerlauf (`IDLE`) zu kontrollieren und belässt ihn in dieser Rolle, solange die Bereitliste (`labor`) leer ist, nämlich kein Kopfelement (`ahead`) enthält. In dieser Rolle instruiert er seinen Prozessor, sich auszuruhen (`relax`). Damit wechselt der Prozess auf die prozessorbezogene Ebene, die den eigentlichen Leerlauf herbeiführt: die ↑CPU in den ↑Schlafzustand versetzen. Diese ↑Aktion ist eine *bedingte Anweisung*, die nur ausgeführt werden darf, wenn die Schlafbedingung noch gilt, das heißt, falls zwischenzeitlich kein Prozess auf die Bereitliste gesetzt wurde. Um die Schlafbedingung überprüfen zu können, wird der prozessorbezogenen Ebene die Adresse der Stelle auf der Bereitliste mitgeteilt, wo der nächste Schub (`batch`) von Prozessen zu erwarten ist. Nimmt der kontrollierende Prozess eine mittlerweile wieder aufgefüllte Bereitlste war, beendet er seine Phase „scheinbarer Untätigkeit“.

leichtgewichtiger Prozess Bezeichnung für einen ↑Prozess, der als ↑Systemkernfaden mit anderen Prozessen seiner Art zusammen im gemeinsamen und durch ↑Speicherschutz isolierten ↑Adressraum stattfindet.

Leitweglenkung (en.) ↑*routing*. Vorrichtung (↑Systemfunktion) zum Festlegen der Route (Leitweg) für ein ↑Signal oder eine ↑Nachricht von seiner Quelle zum Ziel. Ursprünglich eine Einrichtung in der Telekommunikation, um die Wegewahl für Nachrichtenströmen bei der Übertragung in einem vermaschten ↑Netzwerk zu regeln. Im Falle von ↑Mehrkernprozessoren führt jedoch auch ein ↑PIC im Prinzip eine solche Wegewahl in Bezug auf eine ↑Unterbrechungsanforderung durch. Die Verfahren zur Wegewahl unterscheiden sich allerdings sehr

stark vom Einsatzbereich einer solchen Vorrichtung.

level-triggered interrupt (dt.) pegelgesteuerte ↑Unterbrechung, ↑Pegelsteuerung.

LF Abkürzung für (en.) *line feed*, (dt.) ↑Zeilenvorschub. Steuerzeichen, kodiert als $\texttt{0A}_{16}$ in ↑ASCII.

libc Bezeichnung einer ↑Bibliothek für ↑C, auch „*C standard library*“. Umfasst Makros, Typdefinitionen und Funktionen für die verschiedensten Zwecke: Zeichenkettenverarbeitung, mathematische Berechnungen, Ein-/Ausgabe, Speicherverwaltung, sowie betriebssystemnahe Operationen.

LIFO Abkürzung für (en.) *last in, first out*, (dt.) der umgekehrten Reihe nach; Lagerungsverfahren, gleichbedeutend mit ↑LCFS.

light-weight process (dt.) ↑leichtgewichtiger Prozess.

line printer (dt.) ↑Zeilendrucker.

link register (dt.) ↑Verbindungsregister.

link trap (dt.) ↑Bindungsfalle; Mechanismus in ↑Multics.

linker (dt.) ↑Binder.

linking (dt.) ↑Binden.

linking loader (dt.) ↑bindender Lader.

Linux Mehrplatz-/Mehrbenutzerbetriebssystem, von ↑UNIX abstammend. Erste Installation September 1991 (Intel 80386), programmiert in ↑C.

load module (dt.) ↑Lademodul.

loader (dt.) ↑Lader.

loading (dt.) ↑Laden.

location counter (dt.) ↑Adresszähler.

Loch Hohlraum im ↑Arbeitsspeicher, freier ↑Speicherbereich. Ein unbenutzter ↑Adressbereich bestimmter Länge, der zwar zur Speicherung von ↑Text oder ↑Daten zur Verfügung stehen könnte, aber (in logischer Hinsicht) keinem ↑Prozess bekannt ist.

Lochkarte Bezeichnung für ein ↑Speichermedium, auf dem ↑Daten mit Hilfe von Lochungen permanent aufbewahrt werden. IBM hatte eine Karte von 80 Spalten, 12 Zeilen und mit rechteckigen Löchern zur Kodierung patentieren lassen (1928), die das Standardformat in der ↑Stapelverarbeitung darstellt — ein Format, das übrigens auch für Bildschirmfenster mit einem Vorgabewert von 80×24 Zeichen (d.h., zwei Karten untereinander) nach wie vor präsent ist. Pro Spalte wird für gewöhnlich nur ein Zeichen kodiert. Zur Kodierung ganzzahliger Werte $[0, 9]$ werden die unteren 10 Lochpositionen (d.h., Zeilen; auch als „numerische Zone“ bezeichnet), von oben nach unten um eine Spalte nach rechts versetzt, genutzt. Die drei obersten Zeilen dienen der Zonenlochung (*zone punches*), ursprünglich nur um Vorzeichen und die Ziffer `0` zu kodieren: positives Vorzeichen in Zone (Zeile) 12, negatives Vorzeichen in Zone (Zeile) 11, `0` in Zone (Zeile) 10. Durch die Zonenbildung ist die Mehrfachlochung möglich, etwa um Ziffern von Buchstaben und Sonderzeichen zu unterschieden. Beispielsweise hat ein Buchstabe zwei Lochungen, die erste in Zone 12 bis 10 und die zweite für eine Ziffer $[1, 9]$. Derart kodiert ein Loch in Zone 12 und ein weiteres in der numerischen Zone die Buchstaben `A` bis `I`, mit `A` als Ziffer 1 und `I` als Ziffer 9. Entsprechendes für die Buchstaben `J` bis `R` mit einem Loch in Zone 11 und `S` bis `Z` mit einem Loch in Zone 10 (oberste Zeile der numerischen Zone), hier jedoch nur noch die Ziffern $[2, 9]$. Die Kodierung von Satz- und

Sonderzeichen geschieht analog. Diese Form der Informationsrepräsentation bildet nach wie vor die Grundlage für ↑EBCDIC. An der Oberkante der Karte wird oft das so in einer Spalte kodierte Zeichen zusätzlich noch für den Menschen lesbar dargestellt, so es sich um ein druckbares Zeichen handelt.
Für gewöhnlich trifft ein ↑Programm, das die auf der Karte kodierten Informationen verarbeiten soll, bestimmte Annahmen über die Zeilen-/Spaltenorganisation. So definiert(e) der ↑Übersetzer für ↑FORTRAN etwa folgenden Kartenaufbau:

Spalte	Leitkarte	Folgekarten
1–5	Anweisungsnummer	
6	leer oder die Ziffer `0`	eine Ziffer ungleich `0`
7–72	Anweisung	Anweisungsfortsetzung
73–80	optionale Kartenidentifikation	

Steht in der ersten Spalte das Fluchtsymbol (*escape character*) `C`, sind die Spalten 2–80 frei für einen Kommentar: der Übersetzer ignoriert alle folgenden Zeichen bis zum Zeilenende. Anderenfalls bieten Spalten 1–5 Platz zur Kodierung der Nummer einer bestimmten Anweisung innerhalb des Programms. Diese Nummer ist die Spungmarke für eine Sprunganweisung (`goto`) und legt damit die Anweisung fest, mit der die Programmabarbeitung im Falle einer Verzweigung fortgesetzt wird. Spalte 6 dient der Unterscheidung zwischen Leit- und Folgekarte. Mit der Leitkarte beginnt eine neue FORTRAN-Anweisung. Sollte diese Anweisung mehr als 65 Zeichen zur Kodierung benötigen, wird sie auf der jeweils nachfolgenden Karte fortgesetzt. Insgesamt kann eine solche Anweisung somit bis zu 10 aufeinanderfolgende Karten belegen. Die Kartenidentifikation in den Spalten 73–80 dient der durchgehenden Nummerierung aller Karten des Programms, einschließlich der Kommentarkarten. Für gewöhnlich erfolgt die Nummerierung beispielsweise in 10er Schritten: damit wird die Korrektur von Programmierfehlern erleichtert, wenn nämlich durch Änderung der Anweisung neue Karten in einen bestehenden ↑Stapel eingefügt werden müssen (diese werden dann in jeweils dazwischen liegenden 1er Schritten nummeriert).

Lochkartenleseprogramm Bezeichnung für ein ↑Programm, das eine ↑Lochkarte nach der anderen einliest und die darauf kodierten Informationen in den ↑Hauptspeicher bringt. Dazu muss das Programm selbst im Hauptspeicher zur Ausführung bereit stehen, wohin es (vor der Ära ↑nichtflüchtiger Speicher) von Hand geladen wird (↑*bootstrapping*). Nachdem das Programm durch ↑Ureingabe ins System eingespeist und gestartet worden ist, liest es für gewöhnlich ein auf Lochkarten gespeichertes und direkt von der ↑CPU ausführbares (d.h., binär kodiertes) Steuerprogramm an einen vorgegebenen Platz in den Hauptspeicher ein. Dieses Steuerprogramm nutzt sodann das urgeladene Programm oder verfügt über ein eigenes ↑Unterprogramm, um im normalen Betrieb Lochkarten einzulesen.

Lochstreifen Vorläufer der ↑Lochkarte. Anfangs ein Papierstreifen mit fünf Kanälen zur Übertragung von ↑Daten im ↑Fernschreibkode. Später um drei Kanäle erweitert für 8-Bit breite Kodes, insbesondere auch für ↑ASCII. Beiden Ausführungen gemeinsam ist in jeder Spalte (auch Reihe genannt) ein Führungsloch zwischen dem dritten und vierten Datenloch (von unten), wodurch 5-Kanal-Streifen auch durch 8-Kanal-Geräte verarbeitet (gelesen, gelocht) werden können. Die Kodierung erfolgt durch für gewöhnlich runde Löcher innerhalb eines quadratischen Rasters von 1/10 Zoll, womit pro Zoll 10 Zeichen (eins pro Spalte) dargestellt werden können. Bei einer Gesamtlänge von etwa 350 m können bis zu 120 000 Zeichen gespeichert werden. Neben Papier als Herstellungsmaterial, sind die Streifen aus Kunstoff oder einem Laminat von Kunststoff und Metall gefertigt. Durch Zusammenfügen der beiden Enden eines Streifens, lassen sich endlos laufende Steuerstreifen konstruieren.

lock-freedom (dt.) ↑Sperrfreiheit.

Löcherliste ↑Freispeicherliste, auf der jedes verzeichnete ↑Loch einem freien ↑Speicherbereich im ↑Hauptspeicher entspricht.

log-out (dt.) ↑Abmeldung.

login (dt.) ↑Anmeldung.

logische Adresse Bezeichnung für eine ↑Adresse, deren Definitionsbereich ein bestimmter ↑logischer Adressraum ist und die von der wirklichen Lokalität (↑reale Adresse) der durch sie bezeichneten ↑Speicherstelle im ↑Hauptspeicher abstrahiert. Wird eine solche Adresse von der ↑CPU im ↑Abruf- und Ausführungszyklus appliziert, ist ein ↑Busfehler in aller Regel ausgeschlossen. Jedoch kann es zu einer ↑Schutzverletzung kommen, nämlich wenn die Adresse außerhalb ihres Adressraums liegt (↑*segmentation fault*).

logische Synchronisation Synonym zu ↑unilaterale Synchronisation.

logischer Adressraum Bezeichnung für einen ↑Adressraum, der durch eine ↑abstrakte Maschine (↑Kompilierer, ↑Betriebssystem) definiert ist. Jede ↑Adresse in diesem Adressraum ist gültig für den in diesem Adressraum stattfindenden und diese Adresse erzeugenden ↑Prozess: sie repräsentiert eine ↑logische Adresse, die vor oder zur ↑Laufzeit von dem ↑Programm, das den Prozess beschreibt, auf eine ↑reale Adresse abzubilden ist. Vor Laufzeit meint Abbildung durch ↑Übersetzung, ↑Bindung oder spätestens zur ↑Ladezeit des Programms. Demgegenüber beansprucht die Abbildung zur Laufzeit eine ↑MMU, deren Datenstrukturen auf Veranlassung durch den ↑Lader vom Betriebssystem einzurichten sind: nämlich pro Adressraum mindestens eine ↑Seitentabelle oder ↑Segmenttabelle anlegen, je nach Art der MMU, und für jeden ↑Seitendeskriptor beziehungsweise ↑Segmentdeskriptor ein ↑Exemplar in entsprechender Anzahl gemäß der im ↑Lademodul enthaltenen und durch das Betriebssystem vorgegebenen Parameter für das ↑Text-, ↑Daten- und ↑Stapelsegment programmieren. Die Zielmenge der Abbildung ist in beiden Fällen ein ↑realer Adressraum, wobei dieselbe reale Adresse mindestens eine logische Adresse als Urbild hat (Surjektivität): mehrere logische Adressen (in der Regel) verschiedener Adressräume können auf dieselbe reale Adresse abbilden (↑*shared memory*). Für die abzubildenden Adressen kann ein ↑seitennummerierter Adressbereich den Rahmen bilden, wobei dieser entweder den gesamten (logischen) Adressraum abdeckt oder nur pro ↑Segment definiert ist. Die Art bestimmt die MMU, wodurch ein ↑seitennummerierter Adressraum oder ↑segmentierter Adressraum vorgegeben ist. Jedes einzelne Segment muss komplett und zusammenhängend im ↑Hauptspeicher vorliegen, damit das betreffende Programm ausgeführt werden kann. Jedoch ist die durch die ↑Ladeadresse vorgegebene Lokalität jeder ↑Seite/jedes Segments zur ↑Laufzeit des Programms veränderlich: die jeweiligen Strukturelemente (Seite, Segment) von dem ↑Prozessadressraum können im Hauptspeicher verschoben werden, ohne dass dies funktionelle Auswirkungen auf den Prozess hätte — er bewegt sich in einem oder mehreren Adressbereichen, die zwar möglicherweise jeweils wachsen oder schrumpfen, aber deren Adressen darin stets gleich bleiben.

lokale Ersetzungsstrategie Varinate einer ↑Ersetzungsstrategie, bei der immer nur ein ↑Umlagerungsmittel für die Ersetzung ausgewählt wird, das dem ↑Prozessadressraum zugeordnet ist, aus dem heraus der Zugriff auf ein nicht im ↑Hauptspeicher liegender Bestand von ↑Text oder ↑Daten erfolgte. Anders als die ↑globale Ersetzungsstrategie, wird die lokale Ausführung somit keine ↑Ausnahmesituation in einem „fremden“ ↑Prozess herbeiführen können — mehr dazu aber in SP2.

Lokalitätsprinzip (en.) ↑*principle of locality*. Bezeichnung für ein bestimmtes Schema, nach dem die von einem ↑Prozess generierte ↑Referenzfolge aufgebaut ist. Unterschieden wird *zeitliche* und *räumliche Lokalität*. Erstere bezieht sich auf die Wiederverwendung von ↑Daten oder ↑Betriebsmittel innerhalb einer vergleichsweise kleinen Zeitspanne, wohingegen letztere Bezug nimmt auf relativ eng benachbarte ↑Adressen. Eine spezielle Variante räumlicher Lokalität bildet die *sequentielle Lokalität*, wenn sich nämlich die Adressen in der betrachteten Zeitspanne linear entwickeln. Eine stark linear ausgeprägte Referenzfolge ist typisch für Schleifen (`while`, `for`) und ↑Rekursionen, die bei ↑Programmablauf nahezu stetige Adressmengen nicht nur im ↑Textsegment hervorbringen, sondern auch im ↑Datensegment (Durchlauf bei

Feldern oder verketteten Datenstrukturen) oder im ↑Stapelsegment (↑Aktivierungsblock bei rekursiven Durchläufen in mehreren Inkarnationen „nahtlos“ stapeln).
Dieses Prinzip erlaubt bis zu einem gewissen Grad die Vorhersage des Verhaltens der Prozesse in einem ↑Rechensystem und liefert damit die Grundlage nicht nur zur Optimierung des Rechnerbetriebs, sondern auch zur Durchsetzung bestimmter Gütemerkmale. Zeigen Prozesse eine *starke Lokalität*, ist dies beispielsweise für die Abschätzung der ↑Arbeitsmenge, ↑Umlagerung logisch zusammenhängender ↑Seiten oder Berechnung der (für ↑SPN, ↑HRRN oder ↑SRTF benötigten) Länge des nächstens ↑Rechenstoßes eines Prozesses sehr förderlich.

long-term scheduling (dt.) ↑langfristige Einplanung.

lost wake-up (dt.) entgangenes Aufwachen. Bezeichnung für eine ↑Wettlaufsituation, in der ein sich schlafen legender ↑Prozess den für ihn bestimmten Weckruf (*wake-up call*) verpasst. Typische Ursache für ein sich daraus ergebendes Fehlverhalten des Prozesses ist eine *bedingte Anweisung* nachfolgend skizzierten Musters:

```
if (!occur(&event))                          /* do I have to go to bed? */
    †                                      /* yes, hopefully undisturbed! */
    sleep(&event);            /* bed down: release CPU, enter sleep state */
```

Der Prozess stellt fest, dass ein für seinen weiteren Fortschritt relevantes Ereignis (`event`) noch nicht stattgefunden (`occur`) hat und wird sich daraufhin bis zum Ereigniseintritt schlafen legen (`sleep`). Wenn das Ereignis aber genau in der Zeitspanne zwischen der Feststellung der Wartebedingung und dem erfolgten Übergang in den Wartezustand eintritt (†), entgeht dem betreffenden Prozess das Wecksignal, da er eben noch nicht als wartend (d.h., schlafend) verzeichnet ist: dem das Ereignis anzeigenden (externen) Prozess ist der sich schlafen legende Prozess unbekannt.
Einem solchen Fehlverhalten kann in verschiedener Weise vorgebeugt werden. Ein Ansatz besteht darin sicherzustellen, dass das erwartete Ereignis ab dem Zeitpunkt der Abfrage (`occur`) und bis zum Zeitpunkt der Reaktion darauf (`sleep`) nicht angezeigt werden kann. Gemeinhin ist hierfür vor der Abfrage entweder eine ↑Unterbrechungssperre oder eine ↑Verdrängungssperre zu setzen, die dann zurückzunehmen ist, wenn die Wartebedingung nicht zutrifft (weil das Ereignis stattgefunden hat) oder sobald der Prozess schläft (er sich als dieses Ereignis erwartend bekannt gemacht hat):

- Die Unterbrechungssperre sorgt dafür, dass die ↑CPU eine an sie gestellte ↑Unterbrechungsanforderung nicht erfährt und damit auch nicht die ↑Unterbrechungsbehandlung stattfinden kann, in deren Verlauf gegebenenfalls das für den Prozess relevante Ereignis hervorgebracht wird. Gleichfalls kann allerdings, bei ↑Flankensteuerung, das von der ↑Peripherie gesandte Signal unwiederbringlich verloren gehen.
- Demgegenüber sorgt die Verdrängungssperre lediglich für die Aussetzung beziehungsweise Verzögerung der ↑Einlastung der CPU mit dem Prozess, der gegebenenfalls das von dem anderen Prozess erwartete Ereignis verursacht.

Grundsätzlich betreffen diese Sperren aber auch Vorgänge (d.h., Unterbrechungsbehandlungen und Prozesse), die überhaupt keinen Bezug zu dem das Ereignis erwartenden Prozess haben müssen. Dadurch wird ↑Nebenläufigkeit im System unnötig eingeschränkt. Darüber hinaus beugen diese Sperren nur der Anzeige eines Ereigniseintritts durch einen Vorgang vor, der sich mit dem Prozess auf demselben Prozessor zeitlich überlappt.
Ein anderer Ansatz ist es, dass der Prozess, bevor er die Ereignisabfrage tätigt, seine Absicht anzeigt, sich in absehbarer Zeit gegebenenfalls schlafen zu legen. Ab dem Moment der Anzeige kann dem Prozess jederzeit das Wecksignal zugestellt werden, auch wenn er selbst noch nicht schläft. Sollte die Wartebedingung zutreffen, wird sich der Prozess nur dann wirklich schlafen legen, wenn das Wecksignal bisher ausgeblieben ist. Im Gegensatz zum ersten Ansatz setzt dieses *sperrfreie Verfahren* nicht voraus, dass sowohl der ereigniserwartende als auch der -anzeigende Prozess auf demselben Prozessor residieren.

Ein typisches Lösungsmuster für beide Ansätze zeigt folgendes Beispiel, wobei Operationen zur Sperrung/Ankündigung (`catch`) und Entsperrung/Abkündigung (`annul`) den wettlaufkritischen Pfad im ↑Programm entsprechend eingrenzen:

```
catch(&event))                              /* be poised for a wake-up call */
if (!occur(&event))                              /* do I have to go to bed? */
    sleep(&event);                                  /* yes, bed myself down */
else                                                          /* no, stay awake */
    annul(&event);                          /* wake-up call no longer needed */
```

Der Prozess gibt bekannt, die Anzeige eines Ereignisses einfangen zu wollen (`catch`) und erklärt diese Anzeige für ungültig (`annul`), wenn das Ereignis eingetreten sein sollte. Letzteres geschieht implizit beim Schlafenlegen, sobald der Prozess gefahrlos die Kontrolle über den Prozessor abgegeben hat. Der Trick des sperrfreien Ansatzes besteht nun darin, dass der ein Ereignis einfangen wollende Prozess durch die Ereignisanzeige auf die ↑Bereitliste gelangt — auch dann, wenn er selbst noch nicht schläft: der betreffende Prozess wird immer der ↑Einplanung zugeführt. Damit geht dieser Prozess für die Einlastung nicht verloren. Der mögliche ↑Schwebezustand, den der Prozess durch die Bekanntgabe bereitwillig toleriert, nämlich tendenziell schlafen zu gehen und gleichzeitig auf der Bereitliste zu stehen, muss auch in anderen Fällen beachtet werden — beispielsweise bei der Blockierung eines Prozesses und seiner erneuten Bereitstellung vor der eigentlich beabsichtigten Prozessorabgabe oder die Bereitstellung eines Prozesses, der die Rolle als ↑Leerlaufprozess übernommen hat. Für ihn ist daher für gewöhnlich keine Sonderbehandlung erforderlich.

LRU Abkürzung für (en.) *least recently used*. Bezeichnung einer ↑Ersetzungsstrategie, um den Inhalt eines bestimmten Bereichs im ↑Hauptspeicher oder ↑Zwischenspeicher durch den Inhalt eines gleich großen, anderen Bereichs zu ersetzen. Im Falle von Hauptspeicher wird eine in einem ↑Seitenrahmen platzierte ↑Seite durch eine andere Seite ersetzt (↑virtueller Speicher). Im Falle von Zwischenspeicher wird eine darin platzierte ↑Zwischenspeicherzeile durch eine andere (aus dem Hauptspeicher gelesene) ersetzt. Ersetzt wird die Seite/Zeile, die kürzlich am wenigsten genutzt wurde.

m68k Kürzel für eine Prozessorserie (1979–1994) von Motorola, die vier Generationen von 16/32-Bit Prozessoren umfasste: 68000, 68020, 68040 und 68060. Die Modelle 68010 und 68030 waren lediglich kleinere Revisionen und etablierten keine neue Prozessorgenerationen. Das Modell 68050 erreichte niemals Produktreife. Der 68070 war eine lizensierte Variante des 68000 und wurde von Philips produziert. Die Prozessoren kamen bevorzugt in Arbeitsplatzrechner (↑PC) zum Einsatz (insb. Apple Macintosh, Amiga, Atari) und sind nach wie vor in eingebetteten Systemen weit verbreitet (Freescale).

Mach ↑Betriebssystemkern, erste Installation 1985 (DEC VAX). Entwicklung bis 1994 an der Carnegie Mellon University, USA, zur Forschung auf dem Gebiet der ↑Systemprogrammierung. Bildet die Basis unter anderem für ↑Darwin beziehungsweise ↑macOS.

macOS Mehrplatz-/Mehrbenutzerbetriebssystem. Abkürzung für (en.) *Macintosh Operating System* und ab Herbst 2016 gültige Bezeichnung. Auch bekannt als Mac OS X oder OS X, erste Installation 2001 (Cheetah, ↑PowerPC), Prozessorwechsel in 2005 (Tiger, Intel). Auf ↑UNIX (↑FreeBSD) zurückgehendes Betriebssystem.

main board (dt.) ↑Hauptplatine.

mainframe (dt.) Großrechner.

mainstore miss (dt.) ↑Hauptspeicherfehlzugriff.

Makrobefehl Bezeichnung für eine zu einer Einheit zusammengefasste Folge von Befehlen; Kurzform: Makro (Duden). Bei der ↑Übersetzung wird der in einem ↑Programm kodierte Makroaufruf durch die definierte Befehlsfolge ersetzt. Diese Makroexpansion übernimmt gewöhnlich ein Makroprozessor.

Mantelprozedur Umhüllung für ein ↑Unterprogramm, um es in andere Software zu integrieren. Beispiel ist etwa ein in ↑Assembliersprache zu formulierender Aufruf eines in ↑C vorliegenden Unterprogramms zur ↑Unterbrechungsbehandlung, der entsprechend ↑Aufrufkonvention die Inhalte der im Unterprogramm frei verwendbaren Register (↑nichtflüchtiges Register, *callee-saved*) sichert und wiederherstellt. Allgemein jede Art von Software zur Adaptation der formalen Schnittstelle eines Unterprogramms.

manuelle Rechnerbestückung ↑Betriebsart, bei der das ↑Rechensystem gesteuert durch Programmierpersonal nacheinander mit Arbeitspaketen bestückt wird, wobei jedes einzelne Paket durch ein ↑Programm beschrieben ist, von dem zu einem Zeitpunkt stets nur eins zur Ausführung bereit im ↑Hauptspeicher liegt (↑*uniprogramming*). Die Person führt für gewöhnlich folgende Schritte nacheinander aus:

1. die Anweisungen des auszuführenden Programms samt ↑Daten mit dem ↑Kartenlocher auf ↑Lochkarte kodieren und zur Eingabe in das Rechensystem aufbereiten
2. die Lochkarten zum Einlegen in den ↑Kartenleser bereithalten und je nach Art der Programmrepräsentation wie folgt verfahren:
   (a) ist das Programm von der ↑CPU ausführbar (d.h., binär kodiert), weiter bei 3
   (b) liegt das Programm dagegen in Quelltext vor:
      i. zuerst die Lochkarten mit dem direkt von der ↑CPU ausführbaren (d.h., binär kodierten) ↑Übersetzer der ↑Programmbibliothek entnehmen und einlegen
      ii. den Lochkartenleser starten und die Übertragung des Übersetzers in den Hauptspeicher abwarten
      iii. die Ausführung des Übersetzers, der das übersetzte Programm im Haupt- oder ↑Trommelspeicher belässt, durch Knopfdruck starten
      iv. sofern zweckmäßig, das übersetzte und ausführbare Programm zur Ablage in der Programmbibliothek gemäß 7a auf Lochkarten speichern
3. die Lochkarten mit dem (ggf. zu übersetzenden) Programm (ggf. als Eingabe für den Übersetzer) in den Kartenleser einlegen
4. den Kartenleser starten und die Übertragung des Programms in den Haupt- oder Trommelspeicher abwarten (ggf. nach vorheriger Übersetzung)
5. die Ausführung des Programms durch Knopfdruck starten
6. sofern erforderlich, die Lochkarten mit den von dem Programm zu verarbeitenden ↑Daten einlegen und den Kartenleser starten
7. je nach Art der gewünschten Ausgabe, Vorkehrungen zur Darstellung oder Aufbewahrung der Berechnungsergebnisse treffen:
   (a) soll die Ausgabe in maschinenlesbarer Form in eine ↑Bibliothek abgelegt oder durch ein anderes ↑Peripheriegerät noch weiterverarbeitet werden:
      i. zur Aufnahme der Ausgabedaten für eine ausreichende Anzahl leerer Lochkarten sorgen und in einen ↑Kartenstanzer einlegen
      ii. durch Knopfdruck einerseits den Stanzer aktivieren und andererseits die Ausgabe starten, anschließend die Beendigung des Stanzvorgangs abwarten
      iii. die gestanzten Lochkarten dem Stanzgerät entnehmen und zur Aufbewahung in einer Bibliothek geeignet verpacken oder weiter bei 7b
   (b) sofern gewünscht, die Ausgabe in eine für den Menschen lesbare Form bringen:
      i. für ↑Tabellierpapier sorgen und den ↑Zeilendrucker aktivieren oder
      ii. für Zeichenpapier/-material sorgen und den ↑Kurvenschreiber aktivieren
8. die Ausgabe abwarten, von der jeweiligen Ausgabestation abholen und ihrer Bestimmung zuführen

Wesentliche Arbeitserleichterung bringt die ↑automatisierte Rechnerbestückung, bei der die Schritte 2 bis einschließlich 7 als ↑Auftrag beschrieben sind, dessen automatische Bearbeitung (↑*batch processing*) sodann durch Bedienpersonal (↑*operator*) überwacht abläuft.

Markierungsbit Boole'sche Binärziffer, wobei der Ziffernwert einen bestimmten Zustand widerspiegelt, den es festzuhalten gilt. Auch kurz als *Merker* bezeichnet.

Maschinenbefehl Instruktion, elementare Anweisung in einem ↑Maschinenprogramm. Eine solche Anweisung besteht aus einen obligatorischen *Operationsteil*, der die ↑Aktion der Maschine bezeichnet, und einen optionalen *Operadenteil*, der diese Aktion mit den von der Maschine zu verarbeitenden Informationen verknüpft. Letzterer weist je nach Maschinenart eine unterschiedliche Anzahl und Form von Adressangaben auf: 0-Adressbefehl, ausschließlich implizite Adressierung der Operanden (Stapelmaschine); 1-Adressbefehl, implizite Adressierung des ersten und explizite Adressierung des zweiten Operanden (Akkumulatormaschine); 2-Adressbefehl, explizite Adressierung beider Operanden, wobei ein Operand gleichzeitig Quell- und Zieloperand ist (↑CISC); 3-Adressbefehl, explizite Adressierung der beiden Quell- und des einen Zieloperanden (↑RISC).

Maschinenkode Verschlüsselung von einem ↑Maschinenbefehl. Üblich ist die Darstellung eines solchen Befehls als Hexadezimalzahl: so bedeutet $\texttt{05}_{16}$ bei einem ↑x86-kompatiblen ↑Prozessor die Addition mit dem Akkumulator. Früher war auch die Darstellung als Oktalzahl verbreitet (↑Rechner der PDP-Familie).

Maschinenprogramm Bezeichnung für ein ↑Programm, das zum Ablauf oberhalb von einem ↑Betriebssystem bestimmt ist. Ein solches Programm besteht aus Instruktionen (jeweils auch als ↑Maschinenbefehl bezeichnet), die ein ↑Prozessor ausführen kann. Eine solche Instruktion kann als ↑Systemaufruf *explizit* auch eine bestimmte ↑Aktion eines Betriebssystems auslösen. Das Programm ist zudem *implizit* abhängig von einem Betriebssystem, wenn es von der Gültigkeit eines Systemzustands ausgeht, die es nicht selbst durch Systemaufrufe anfordert, sondern zu seiner ↑Ladezeit vom Betriebssystem zugesichert wird und zur ↑Laufzeit bestehen bleibt.

Maschinensprache Programmiersprache, deren Sprachelemente in Form von ↑Maschinenkode repräsentiert sind. Die eigentliche Programmiersprache einer ↑CPU.

Maschinenwort Informationseinheit in einem ↑Prozessor (↑CPU). Allgemein ausgelegt als Bitvektor, dessen Länge die ↑Wortbreite des Prozessors definiert.

Massenspeicher ↑Speicher zur dauerhaften Ablage sehr großer Mengen von ↑Daten aller Art. Oft Synonym für Sekundärspeicher. Umfasst jedoch auch Tertiärspeicher, der nicht permanent im ↑Rechensystem angeschlossen ist und sich in Form von Archiven zeigt.

MCU Abrzung für (en.) *microcontroller unit*, (dt.) Mikrokontrollereinheit.

medium-term scheduling (dt.) ↑mittelfristige Einplanung.

Mehradressraummodell Art von ↑Schutz in einem ↑Rechensystem, die sicherstellt, dass kein ↑Prozess aus seinen ↑Prozessadressraum ausbrechen kann. Genau genommen ist hier ein ↑schwergewichtiger Prozess gemeint: ein ↑leichtgewichtiger Prozess oder ↑federgewichtiger Prozess teilt sich mit anderen seiner Art implizit denselben globalen ↑Adressraum, er überlappt sich damit in Teilen seines eigenen Adressraums mit Teilen der eigenen Adressräume gleichgestellter (leicht-/federgewichtiger) Prozesse in diesem globalen Adressraum. Der globale Adressraum unterliegt dem ↑Speicherschutz, hier konkret auf Grundlage einer ↑MMU oder ↑MPU. Wesentlicher Aspekt dabei ist die ↑Adressraumisolation (im Gegensatz zum ↑Einadressraummodell), durch die ein Ausbrechen aus dem eigenen und damit gegebenenfalls Einbrechen in einen fremden Adressraum für jeden (schwergewichtigen) Prozess unterbunden wird.

Diese Isolation kann stark oder schwach ausgelegt sein. Ersterer Fall meint, dass der Prozessadressraum als *strikt privat* gilt, und zwar bezogen auf jedes im Rechensystem zur Ausführung kommende ↑Programm, nämlich ↑Maschinenprogramm und ↑Betriebssystem (z.B. ↑macOS). Demgegenüber ist bei letzterem Fall der Prozessadressraum *teilprivat* (z.B. ↑Windows NT und ↑Linux): agiert der Prozess im Maschinenprogramm (↑*user mode*), ist sein Adressraum beschränkt durch die Berechnungsvorschrift nur dieses Programms; agiert der Prozess im Betriebssystem (↑*system mode*), erweitert sich sein Adressraum und ist zusätzlich beschränkt um die Berechnungsvorschrift des Programms „Betriebssystem". Bei dieser Variante ist also der durch ein Maschinenprogramm belegte ↑Speicherbereich (ggf. auch mehrere davon) inhärenter Bestandteil des für das Betriebssystem definierten Adressraums. Technisch wird dies erreicht, indem der durch die ↑Adressbreite insgesamt mögliche Adressraum partitioniert wird. Gebräuchlich ist eine gleichmäßige (32-Bit Windows NT: 2 GiB jeweils für Benutzer- und Systemmodus) oder ungleichmäßige (32-Bit Windows NT *Enterprise Edition* und Linux: 3 GiB für Benutzer- und 1 GiB für Systemmodus) Aufteilung. Die Grenze zwischen beiden Adressraumpartitionen entspricht einem ↑Schutzgatter. Im Gegensatz dazu steht bei starker Isolation jedem Programm (also Maschinenprogrammen und Betriebssystem) jeweils ein Adressraum mit voller Adressbreite (virtuell) zur Verfügung. Neben eines größeren Adressraums ist der Vorteil dieses Ansatz gegenüber der schwachen Isolation, dass im Betriebssystem verborgene Programmierfehler die Integrität der Maschinenprogramme nicht verletzen können. Andererseits gestalten sich Zugriffe des Betriebssystems auf den Speicherbereich eines Maschinenprogramms, etwa als Folge von einem ↑Systemaufruf, schwieriger und mit weitaus höherer ↑Latenz: der Speicherbereich, auf den zugegriffen werden soll, muss explizit in den Betriebssystemadressraum eingeblendet und nach erfolgtem Zugriff wieder ausgeblendet werden. Bei schwacher Isolation dagegen ist die mit einem Systemaufruf übergebene Adresse auch im Betriebssystemdressraum gültig und direkt verwendbar. Jedoch muss das Betriebssystem hier vor Verwendung einer solchen Adresse eine *Integritätsprüfung* durchführen, um unautorisierten Zugriffen auf den für die Adressraumpartition des Betriebssystems definierten ↑Adressbereich vorzubeugen.
Im Grunde ist diese Schutzart ein Beispiel für ↑partielle Virtualisierung, indem nämlich vor allem Adressen beziehungsweise Adressbereiche virtualisiert werden und nicht notwendigerweise auch gleich noch der ↑Hauptspeicher. Adressen sind dadurch nicht mehr systemweit eindeutig (im Gegensatz zum Einadressraummodell), sondern nur noch bedeutsam in einem bestimmten Bezugssystem, das als ↑virtueller Adressraum definiert ist. Jeder schwergewichtige Prozess erhält einen eigenen virtuellen Adressraum, den er nicht aus eigenen Kräften nach Belieben ausdehnen kann und der ihn daher auch immun gegenüber unautorisierten Zugriffen anderer Prozesse macht (↑*protection domain*) — vorausgesetzt, das Betriebssystem genügt seiner Spezifikation und funktioniert damit in korrekter Weise.

Mehrbenutzerbetrieb Variante von ↑Dialogbetrieb, bei der das ↑Betriebssystem zu einem Zeitpunkt mindestene einen Teilnehmer den Rechenbetrieb ermöglicht (Gegenteil von ↑Einbenutzerbetrieb). Für gewöhnlich geht diese Variante mit ↑Mehrprogrammbetrieb einher, etwa indem jedem Teilnehmer ein eigener ↑Kommandointerpreter zur Dialogführung bereitgestellt wird. Allerdings kann allen Teilnehmern auch ein- und dasselbe ↑Exemplar eines solchen Interpreters zugeordnet sein, der dann als ↑nichtsequentielles Programm ausgelegt eben alle Teilnehmer bedient. Eine solche Dialogführung im ↑Einprogrammbetrieb unterstützt zwar mehrere Teilnehmer (↑Teilnehmerbetrieb, ↑Teilhaberbetrieb), kommt jedoch nicht dem ↑Schutz teilnehmerspezifischer ↑Daten entgegen.

Mehrfädigkeit Fähigkeit einer (realen/virtuellen) Maschine mehr als einen ↑Faden zugleich durch ↑Parallelverarbeitung ausführen zu können. Grundlage bildet ein ↑nichtsequentielles Programm, das die einem jeden Faden zukommende ↑Aufgabe beschreibt.

Mehrkernprozessor Bezeichnung für eine ↑CPU, die aus mehr als einen ↑Rechenkern besteht.

Mehrprogrammbetrieb Bezeichnung für die ↑Betriebsart eines Rechensystems, bei der mehr als ein ↑Programm zugleich im ↑Arbeitsspeicher zur Ausführung zur Verfügung stehen. Die

Programme werden im Grunde solange ausgeführt, bis sie von selbst die Kontrolle über den ↑Prozessor abgeben (*run to completion*). Dabei erfolgt die Kontrollübergabe in solchen Situationen, die die weitere Benutzung des Prozessors logisch bedingt im Programm nicht mehr erfordert: nämlich wenn ein ↑wiederverwendbares Betriebsmittel oder ein ↑konsumierbares Betriebsmittel (inkl. Ein-/Ausgabe) zum weiteren Fortschritt benötigt wird, aber nicht zur Verfügung steht, oder wenn die Programmausführung endet. Mehrfachnutzung des Prozessors wird zwar unterstützt, jedoch nur in räumlicher Hinsicht und in kooperativer Art und Weise eines Programmablaufs. Erst die Erweiterung um ↑Simultanverarbeitung sorgt für die Mehrfachnutzung in Raum und Zeit.

mehrseitige Synchronisation Synonym zu ↑multilaterale Synchronisation.

Mehrstromstapelmonitor Bezeichnung für einen ↑Stapelmonitor, der die gestapelten Aufträge als mehrfachen Verarbeitungsstrom ausführt. Jeder Verarbeitungsstrom entspricht einem ↑Maschinenprogramm, wovon mehrere verschiedene zugleich im ↑Hauptspeicher residieren (↑*multiprogramming*) und jedes einzelne zur ↑Laufzeit eine bestimmte Einzelarbeit (↑*job*) leistet. Sobald ein Maschinenprogramm aus Mangel an ↑Betriebsmittel nicht weiter ausgeführt werden kann, schaltet der Stapelmonitor um zu einen anderen Verarbeitungsstrom und nimmt damit die Ausführung eines anderen Maschinenprogramms auf: kooperative und dynamische ↑Ablaufplanung. Der Ausführungswechsel zwischen den verschiedenen Maschinenprogrammen geschieht implizit mit einer Betriebsmittelanforderung, wenn diese nämlich in dem Moment, wo sie von dem ↑Prozess gestellt wird, nicht erfüllbar ist. Die zusätzliche Last für das ↑Betriebssystem, im Vergleich zum ↑Einzelstromstapelmonitor, besteht in der ↑Betriebsmittelkontrolle und dem ↑Speicherschutz in Bezug auf den einzelnen (in seinem jeweiligen Maschinenprogramm stattfindenden) Prozess beziehungsweise ↑Prozessadressraum.

memory barrier (dt.) ↑Speicherbarriere.

memory consistency (dt.) ↑Speicherkonsistenz.

memory footprint (dt.) ↑Speichergrundfläche.

memory-mapped I/O (dt.) ↑speicherabgebildete Ein-/Ausgabe.

message (dt.) ↑Nachricht, Botschaft.

message passing (dt.) ↑Nachrichtenversenden, Botschaftenaustausch.

mitlaufende Planung (en.) ↑*on-line scheduling*. Modell der ↑Ablaufplanung, die ört- und zeitlich gekoppelt mit den einzuplanenden ↑Prozessen geschieht: sie findet im Hintergrund mitlaufend statt und erzeugt einen gemeinhin dynamischen ↑Ablaufplan, der zu einem gegenwärtigen Zeitpunkt (d.h., zeitnah zur Aufstellung) vom ↑Betriebssystem abgearbeitet wird. Der ↑Planer ist fester Bestandteil von oder aufgesetzter Teil (↑*user-level scheduling*) bei einem Betriebssystem (örtlich gekoppelt). Anders als ↑vorlaufende Planung wird der Ablaufplan in direkter Verbindung zu den jeweils darauf stehenden Prozessen aktualisiert und fortgeschrieben (zeitlich gekoppelt).
Diese Form von ↑Planung ist bei fehlendem ↑Vorwissen zu den einzuplanenden Prozessen obligatorisch, dass heißt, wenn ↑Prozessgrößen oder die zu erwartende Prozessanzahl unbekannt sind. Aber auch trotz Vorhandensein solchen Wissens kann es geboten sein, ein mitlaufendes Verfahren einer möglichen vorlaufenden Variante vorzuziehen, nämlich wenn Flexibilität im Vordergrund steht und der Dynamik im ↑Rechensystem Rechnung zu tragen ist. Letzterer Aspekt ist typisches Merkmal eines ↑Universalbetriebssystems.

mittelfristige Einplanung (en.) ↑*medium-term scheduling*. ↑Ablaufplanung auf mittlere Sicht, typischerweise im Millisekunden- oder Sekundenbereich. Eine in der Speicherverwaltung verankerte ↑Systemfunktion, die festlegt, welche Bereiche im ↑Hauptspeicher zu gegebener Zeit für einen ↑Prozess freizumachen oder zu belegen sind (↑*swapping*) — mehr dazu aber erst in VL 12.3.

MLFQ Abkürzung für (en.) *multi-level feedback queue*. Bezeichnung für eine ↑Planung von ↑Prozessen, bei der ....

MLQ Abkürzung für (en.) *multi-level queue*. Bezeichnung für eine ↑Planung von ↑Prozessen, bei der ....

MMU Abkürzung für (en.) *memory management unit*, (dt.) Speicherverwaltungseinheit.

Mnemon (gr.) erinnern.

mode change (dt.) ↑Moduswechsel.

modified bit (dt.) ↑Veränderungsbit (Patentwesen).

Modul Softwareeinheit mit eigenem Datenmodell und eigenem Satz von Operationen darauf. Die Daten sind nur über die moduleigenen Operationen zugänglich (*information hiding*).

Moduswechsel Übergang von einen ↑Arbeitsmodus in einen anderen.

Monitor Datentyp, Klasse mit impliziten Eigenschaften zur ↑Synchronisation; Programmierkonvention. Für alle Operationen, die auf ein ↑Exemplar eines solchen Datentyps angewendet werden, gilt ↑wechselseitiger Ausschluss. Diese Operationen müssen in der Datentypschnittstelle spezifiziert sein. Per Definition ist jedes ↑Unterprogramm, das eine solche Operation implementiert, ein ↑kritischer Abschnitt. Innerhalb eines solchen Abschnitts kann ein ↑Prozess eine ↑Bedingungsvariable verwenden, um sich auf ein ↑Ereignis zu synchronisieren beziehungsweise ein solches anzuzeigen.
Ursprünglich ein Konzept der *Typisierung* in höheren Programmiersprachen, indem nämlich die korrekte Verwendung von Operationen zur Synchronisation durch einen ↑Kompilierer abprüfbar wird und so Programmierfehler vermieden werden können. Die zum wechselseitigen Ausschluss erforderlichen Anweisungen erzeugt der Kompilierer, wie auch die Instruktionen, um einen kritischen Abschnitt zeitweilig verlassen zu können, ohne diesen in der Zwischenzeit gesperrt zu halten. Mangels Sprachunterstützung ist dieses Konzept jedoch viel mehr zur Programmierkonvention entartet, das heißt, dem Menschen als ↑Prozessor wird es erleichtert, ein (vermeintlich) korrektes ↑nichtsequentielles Programm zu formulieren.

motherboard (dt.) ↑Trägerleiterplatte.

mount (dt.) Befestigung, ↑Einhängen.

mount point (dt.) Punkt, an dem `mount(2)` ein ↑Dateisystem aufpfropft (↑*mounting point*).

mounting point (dt.) ↑Befestigungspunkt.

MPU Abkürzung für (en.) *memory protection unit*, (dt.) Speicherschutzeinheit.

MS-DOS Akbürzung für „*Microsoft Disk Operating System*“, Einplatz-/Einbenutzerbetriebssystem, erste Installation 1981 (Intel 8086).

multi-stream batch monitor (dt.) ↑Mehrstromstapelmonitor.

multi-user mode (dt.) ↑Mehrbenutzerbetrieb.

Multics Mehrplatz-/Mehrbenutzerbetriebssystem, Abkürzung für (en.) „*Multiplexed Information and Computing Service*“. Konzeptpapiere im Jahr 1965, erste echte Installation Dezember 1967 (GE 645), programmiert in EPL (*early* ↑PL/1), in Betrieb bis Oktober 2000.

multilaterale Synchronisation Bezeichnung einer ↑Synchronisation, die sich auf jeden beteiligten ↑Prozess auswirken kann. Jeder der betroffenen Prozesse bestreitet dasselbe Protokoll, um Synchronisation in Bezug auf eine bestimmte ↑Aktion oder ↑Aktionsfolge zu erzielen. Kommt hierzu ↑blockierende Synchronisation zur Geltung, wird jeder dieser Prozesse durch

das Protokoll blockiert werden können. Für ↑nichtblockierende Synchronisation ist davon auszugehen, dass dieses Protokoll jeden der beteiligten Prozesse die Aktion/Aktionsfolge wiederholen lässt. Diese Art der Synchronisation ist typisch, wenn Prozesse gleichzeitig ein ↑wiederverwendbares Betriebsmittel anfordern, das jedoch nur exklusiv belegt und genutzt werden darf. Eine besondere Variante davon ist ein ↑kritischer Abschnitt, für den ↑wechselseitiger Ausschluss von Prozessen sicherzustellen ist. Alle diese Verfahren wirken mehrseitig.

Multiplexverfahren Methode, deren Ursprung in der Nachrichtentechnik liegt, nämlich um mehrere Signale (↑konsumierbares Betriebsmittel) simultan über ein Medium (↑wiederverwendbares Betriebsmittel) übertragen zu können. In einem ↑Betriebssystemkern findet diese Methode Anwendung, um mehr als einen ↑Prozess zugleich auf dem ↑Prozessor einer (realen/virtuellen) Maschine stattfinden lassen zu können. Grundlage dafür bilden entsprechende Verfahren zur ↑Prozesseinplanung, die ↑Simultanverarbeitung ermöglichen.

multiprocessing (dt.) ↑Simultanverarbeitung.

multiprogramming (dt.) ↑Mehrprogrammbetrieb.

Multiprozessor ↑Rechner, der aus mehr als einer ↑CPU besteht.

multitasking (dt.) ↑Mehrprogrammbetrieb, wobei ↑Programm hier zu differenzieren ist. Im Vordergrund steht die ↑Aufgabe, die eben durch ein Programm beschrieben wird. Jedes Programm kann entweder nur eine oder mehrere Aufgaben beschreiben. Sind es mehrere Aufgaben, können diese von derselben oder verschiedenen Arten (Typen) sein. Zudem steht die wirkliche Ausprägung einer Aufgabe nicht im Vordergrund: eine Aufgabe kann einen eigenständigen ↑Handlungsstrang darstellen oder teilt sich einen solchen mit anderen Aufgaben.

multithreading (dt.) ↑Mehrfädigkeit.

Mutex Kunstwort, Abkürzung für (en.) ↑*mutual exclusion*; Sperrmechanismus, mit dem ↑wechselseitiger Ausschluss ausgeübt wird. Der Mechanismus sichert zu, dass nur der ↑Prozess die Sperre aufheben kann, der diese zuvor auch gesetzt hat. Typischerweise wird mit diesem Konstrukt ein ↑kritischer Abschnitt eingefasst. Einen damit eingefassten Abschnitt kann nur der Prozess entsperren, der diesen zuletzt sperrte, betrat und jetzt verlässt. Für jeden anderen Prozess scheitert die ↑Aktion — was in dem Fall auf einen Programmierfehler hindeutet und eigentlich eine ↑Ausnahmesituation darstellt, im Allgemeinen jedoch ohne Auswirkung für den offensichtlichen fehlgeleiteten Prozess bleibt. Grundlage für die Implementierung kann ein ↑binärer Semaphor sein: In der Sperroperation (`lock`) wird nach dem ↑P der gegenwärtige Prozess vermerkt, dessen Identität in der Entsperroperation (`unlock`) noch vor dem ↑V mit demjenigen Prozess überprüft wird, der den kritischen Abschnitt verlässt:

```
void lock(mutex) {
    P(mutex.sema);
    mutex.owner = self();
}
```

```
void unlock(mutex) {
    if (mutex.owner != self())
        raise(EPERM);
    mutex.owner = NULL;
    V(mutex.sema);
}
```

In dem Beispiel liefert `self()` entweder eine ↑PID oder den ↑Prozesszeiger, und mit `raise()` wird eine ↑Ausnahme erhoben, nämlich dass der gegenwärtige Prozess eine für ihn unerlaubte Operation durchführt. Dieses Beispiel geht davon aus, dass die ↑Ausnahmebehandlung richtigerweise die Termination des fehlgeleiteten Prozesses erzwingt und damit die Ausführung der gescheiterten Entsperroperation nicht wieder aufnimmt.

mutual exclusion (dt.) gegenseitiger oder ↑wechselseitiger Ausschluss.

Nachricht (en.) ↑*message*. Mitteilung, die ein ↑Prozess einem anderen Prozess in Bezug auf einen bestimmten Umstand die Kenntnis des neuesten Sachverhalts vermittelt (in Anlehnung an

den Duden). Für gewöhnlich umfasst diese Mitteilung eine gewisse Menge von ↑Daten, denen ein bestimmter ↑Speicherbereich zugeordnet ist. Die Mitteilung kann aber auch „datenlos“ erfolgen, beispielsweise indem ein wartender Prozess deblockiert wird und dieser dann bei Wiederaufnahme eben davon ausgehen darf, dass ein anderer Prozess ihm die Möglichkeit zur Fortsetzung vermittelt hat. Letztere Form wird auch als ↑Signal bezeichnet.

Nachrichtenversenden (en.) ↑*message passing*. Art der Kommunikation, bei der ein ↑Prozess einem anderen Prozess eine ↑Nachricht explizit zusendet. Eine zwingende Maßnahme, wenn den kommunizierenden Prozessen kein ↑gemeinsamer Speicher für den Austausch von ↑Daten zur Verfügung steht. Dies ist typisch für ein ↑Rechnernetz. Aber auch trotz gemeinsamem Speicher kann die nachrichtenorientierte Interaktion von Prozessen geboten sein, nämlich im Falle einer zu schwachen ↑Speicherkonsistenz. Wenn der logische Ablauf einer bestimmten Folge von Speicheroperationen indeterministisch ist, müssen die betreffenden Prozesse durch ein wenigstens in ihrer Gruppe einheitliches Protokoll explizit den Zugriff auf die gespeicherten Daten regeln. Im Zuge der notwendigen ↑Synchronisierung, was häufig den Hauptanteil an ↑Gemeinkosten ausmacht, senden die beteiligten Prozesse die Daten (für die Konsistenz innerhalb der Gruppe gelten muss) oder auch nur die ↑Referenz darauf gleich mit.
Der Vorgang, eine Nachricht zu versenden, kann *synchron* oder *asynchron* für den betreffenden Prozess erfolgen. Damit ist gemeint, dass der Prozess auf die Entgegennahme der Nachricht warten (synchron) oder nicht warten (asynchron) muss. Entgegengenommen wird die Nachricht entweder von dem Prozess, der die weitere Verarbeitung vornehmen wird (synchon), oder von einer ↑Entität, die lediglich für eine Zwischenspeicherung sorgt (asynchron). Ein Prozess, der eine Nachricht entgegennimmt, tut dies für gewöhnlich immer synchron: nämlich in dem Moment, sobald er den Auftrag dazu angenommen hat.
Zusätzlich zum Aspekt der Synchronizität kann eine Nachricht *gepuffert* oder *ungepuffert* von einem zum anderen Prozess übertragen werden. Letzteres bedeutet üblicherweise, dass der mit der Kommunikation verbundene Datentransfer durchgehend (*end-to-end*) erfolgt, und zwar über eine Art ↑Speicherdirektzugriff direkt heraus aus dem (ggf. durch ↑Speicherschutz isolierten) ↑Adressraum des die Nachricht versendenden hinein in den (ggf. ebenso isolierten) Adressraum des diese Nachricht entgegennehmenden Prozesses. Das Kommunikationsmodell sieht keine Pufferung der kompletten Nachricht vor, gleichwohl kann für die Durchführung des Transfers aus technischen Gründen eine Zwischenspeicherung von Nachrichtenfragementen erfolgen. Solche Gründe ergeben sich für gewöhnlich bei der Kommunikation über ein Rechnernetz, aber beispielsweise auch zur Unterstützung von ↑Umlagerung: um den eine Nachricht entgegennehmenden Prozess unabhängig von ihrer Anwesenheit im ↑Arbeitsspeicher voranschreiten zu lassen, kann ihre Zwischenspeicherung in einen vom ↑Betriebssystem dafür vorgesehenen ↑Speicherbereich zweckmäßig sein. Demgegenüber bedeutet die gepufferte Übertragung einer (gesamten) Nachricht, keine Zwischenspeicherung aus technischen Gründen. Vielmehr ist die Pufferung im Kommunikationsmodell verankert.
Gepufferter Nachrichtentransfer dient vornehmlich der zeitlichen Entkopplung von Sende- und Empfangsprozess: der Sendeprozess muss damit nicht die Bereitwilligkeit des Empfangsprozesses abwarten, eine Nachricht entgegenzunehmen. So wird dadurch insbesondere auch der asynchrone Versandt einer Nachricht ermöglicht (s.o.). Jedoch ist für letzteres die Nachrichtenpufferung nicht zwingend. Stattdessen kann das Kommunikationsmodell den zur Zusammenstellung einer zu sendenden Nachricht benötigten Speicherbereich als ↑wiederverwendbares Betriebsmittel explizit machen. Nach dem asynchronen Versenden der Nachricht kann der Sendeprozess durch ↑einseitige Synchronisation mit dem ↑Ereignis, das die Beendigung des Datentransfers anzeigt, auch ohne Zwischenspeicherung zeitlich unabhängig vom Empfangsprozess voranschreiten. Der Sendeprozess fragt von Zeit zu Zeit ab, ob dieses Ereignis bereits eingetreten ist und geht dazwischen anderen Dingen nach oder er blockiert sich auf die Ereignisanzeige durch den Empfangsprozess.

Name Bezeichnung; kennzeichnende Benennung von dem ↑Exemplar eines Programmtextes oder -datums (d.h., von einem ↑Unterprogramm oder einer Programmvariablen). Die Benennung kann eine Nummer oder ein ↑Symbol sein.

name binding (dt.) ↑Namensbindung.

name resolution (dt.) ↑Namensauflösung.

name space (dt.) ↑Namensraum.

Namensauflösung Vorgang, bei dem ein ↑Name umgewandelt wird in eine ↑numerische Adresse. Hier ist insbesondere der ↑Pfadname Ausgangspunkt für die Aufschlüsselung, die letztlich zur Lokalisierung der die gesuchte ↑Datei beschreibenden Datenstruktur (↑inode) im ↑Dateisystem führt. Beginnt der Pfadname mit einem ↑Separator, verläuft die Suche von der Wurzel des Dateisystems (↑*root directory*) ausgehend: absolute Bezeichnung/Adressierung der Datei. Anderenfalls startet die Suche an der Stelle im ↑Namensraum, wo sich der ↑Prozess gegenwärtig aufhält (↑*current working directory*): relative Bezeichnung/Adressierung der Datei. Nacheinander werden die einzelnen Pfadabschnitte, jeder ein ↑Name gefolgt von einem Separator oder dem Textende, in dem jeweiligen Verzeichnis gesucht. Jedes dieser Verzeichnisse ist eine spezielle Datei, die mit Hilfe der in ihrem ↑Indexknoten enthaltenen Informationen ausgelesen wird. So wird ein ↑Verzeichniseintrag nacheinander offengelegt und mit dem gesuchten Namen verglichen. Den Anfang nimmt der Knoten von dem ↑Wurzelverzeichnis beziehungsweise ↑Arbeitsverzeichnis. Entspricht der gesuchte Name einem Verzeichniseintrag, wird mit der verzeichneten ↑Indexknotennummer der nächste Indexknoten geladen. Ist der Indexknoten ↑Deskriptor einer weiteren Verzeichnisdatei und kann die Suche fortgesetzt werden, weil das Textende noch nicht erreicht ist, wiederholt sich der Ablauf. Im Fehlerfall (unbekannter Name, falscher Dateityp, ↑defekter Block) bricht die Suche ab. Am Textende angekommen, spiegelt der zuletzt geladene Indexknoten die gesuchte Datei wider.

Namensbindung Vorgang, bei dem eine bindende Beziehung zwischen ↑Name und ↑numerische Adresse eingegangen wird; Verbindung zwischen ↑symbolische Adresse und numerische Adresse herstellen. Hier insbesondere die Einrichtung einer ↑Verknüpfung zwischen ↑Dateiname und einem ↑Exemplar der die betreffende ↑Datei beschreibenden Datenstruktur (↑inode) im ↑Dateisystem. Dazu ist ein ↑Verzeichniseintrag anzulegen und diesem Eintrag ist ein freier ↑Indexknoten zuzuordnen. Letzteres meint, die ↑Indexknotennummer in den Eintrag zu speichern und dadurch die ↑Bindung zu manifestieren.

Namenskontext Sinnzusammenhang, in dem ein ↑Name steht; Bezugsrahmen von einem Namen. Der Kontext, in dem ein Name eindeutig ist. Die Eindeutigkeit sichert der Name selbst zu, indem er nicht noch einmal in dem Kontext definiert ist.

Namensraum Menge von eindeutigen Bezeichnungen, endlich. Jedes Element in dem in sich abgeschlossenen Raum ist ein ↑Name für eine ↑Entität. Dieser Raum (↑*name space*) entspricht einem ↑Adressraum, der den Kontext für eine ↑symbolische Adresse definiert und damit einen bestimmten, umfassenden ↑Namenskontext bildet. In dem Sinne ist ein Name, der letztlich ein ↑Symbol in einem noch zu übersetzenden/bindenden ↑Programm darstellt, gleichsam Synonym für eine besondere ↑virtuelle Adresse im korrespondierenden ↑Maschinenprogramm. Ein solcher Raum kann einen gegliederten oder ungegliederten Aufbau haben. Im gegliederten Fall ist der Raum als ↑hierarchischer Namensraum ausgeprägt, in dem die Teilepaare jeweils Namenskontexte sind und ein ↑Pfadname die bezeichnete Entität eindeutig identifiziert. Demgegenüber ist im ungegliederten Fall eine „flache Struktur“ des Raums typisch. In dem Fall muss der Name selbst die eindeutige Identifizierung der Entität sicherstellen. Die hierarchische Auslegung ist in Anbetracht der typischerweise sehr großen Anzahl zu benennender Entitäten (↑Datei) insbesondere für ein ↑Dateisystem äußerst zweckmäßig. Hier müssen zudem die gespeicherten Informationen zur Struktur und ↑Verknüpfung von Namenskontexten auch in das Dateisystem selbst integriert sein (↑Persistenz). Erst dadurch ist es möglich, nach erfolgtem ↑Einhängen eines Dateisystems die darin verzeichneten Dateien zu erkennen und zu gebrauchen.

Nebenläufigkeit Verhältnis von nicht kausal abhängigen Ereignissen, die sich also nicht beeinflussen. Ein Ereignis ist nebenläufig zu einem anderen, wenn es im Anderswo (d.h., weder in der Zukunft noch in der Vergangenheit) des anderen Ereignisses liegt, weder Ursache noch Wirkung ist: wenn zu dem anderen Ereignis keine Daten-, Kontrollfluss- oder Zeitabhängigkeit besteht.

Nebenläufigkeitssteuerung Verfahren, das trotz ↑Nebenläufigkeit einer ↑Aktion oder ↑Aktionsfolge die Entwicklung korrekter Berechnungen sicherstellt. Dabei kann eine ↑optimistische Nebenläufigkeitssteuerung oder eine ↑pessimistische Nebenläufigkeitssteuerung Herangehensweise praktiziert werden.

Netzwerk Vernetzung mehrerer voneinander unabhängiger elektronischer Geräte, die den Datenaustausch zwischen diesen ermöglicht; Kurzform: Netz (in Anlehnung an den Duden).

next fit (dt.) nächstbeste Passung: ↑Platzierungsalgorithmus.

nichtblockierende Synchronisation Gegenteil von ↑blockierende Synchronisation: jeder ↑Prozess versucht ungehindert, eine bestimmte ↑Aktion oder ↑Aktionsfolge durchzuführen, auch wenn dies gleichzeitig geschieht. Dabei darf die Berechnung dieser Aktion/Aktionsfolge dann jedoch nur lokale Signifikanz für jeden der beteiligten Prozesse haben. Um globale Signifikanz zu erreichen und damit eine Zustandsänderung für alle Prozesse sichtbar zu machen, ist jeder beteiligte Prozess verpflichtet, seine lokale Berechnung nach außen zu bestätigen. Die Bestätigung gelingt nur, wenn kein anderer Prozess diese zwischenzeitig bereits erreicht hat. Anderenfalls scheitert die Bestätigung und der betroffene Prozess wiederholt die Aktion/Aktionsfolge. Auch als ↑optimistische Nebenläufigkeitssteuerung bezeichnet.

nichtflüchtiger Speicher Klassifikation für einen ↑Speicher, der seinen Inhalt auch ohne Anliegen einer Betriebsspannung für längere Zeit erhält; Festspeicher; ↑Sekundärspeicher oder ↑Tertiärspeicher. Neben ↑Wechseldatenträger kommen davon heute (2016) vor allem gerade Magnetplatte und ↑SSD als fest in einem ↑Rechensystem eingebaute Datenträger zum Einsatz.

nichtflüchtiges Register (en.) ↑*non-volatile register*. Konzept der ↑Aufrufkonvention: der Inhalt eines solchen Prozessorregisters gilt als beständig. Vorkehrungen zur Sicherung und Wiederherstellung des Registerinhalts werden im ↑Unterprogramm bei Bedarf getroffen (*callee-saved*).

nichtsequentieller Prozess Bezeichnung für einen ↑Prozess, der durch ein ↑nichtsequentielles Programm definiert ist. Bei ↑Parallelverarbeitung kann eine ↑Aktion oder ↑Aktionsfolge in solch einem Prozess zeitlich überlappt und dadurch möglicherweise beschleunigt vonstattengehen. Jedoch birgt dies auch die Gefahr einer ↑Wettlaufsituation, deren Vorbeugung wiederum eine Geschwindigkeitseinbuße mit sich bringt.

nichtsequentielles Programm (en.) ↑*non-sequential program*. Bezeichnung für ein ↑Programm, das Konstrukte zur Formulierung explizit paralleler Abläufe verwendet. Die Konstrukte können in der Programmiersprache, in der das Programm formuliert wurde, enthalten sein oder werden sprachunabhängig durch eine Programmbibliothek (z.B. *POSIX Threads* bzw. *pthreads*; aber auch UNIX Signale, `signal(3)`) bereitgestellt.

NMI Abkürzung für (en.) *non-maskable interrupt*, (dt.) nichtmaskierbare ↑Unterbrechung.

no-op instruction (dt.) ↑Leerbefehl.

non-sequential program (dt.) ↑nichtsequentielles Programm.

non-volatile register (dt.) ↑nichtflüchtiges Register.

Notizblockspeicher Bezeichnung für einen ↑Speicher in der ↑CPU, der komplett durch Software verwaltet wird. Einem ↑Zwischenspeicher sehr ähnlich, jedoch mit dem ↑Speicherwort als kleinste Verwaltungseinheit. Neben Einzelwortzugriffe ermöglicht die CPU typischerweise durch ↑DMA den Transfer großer zusammenhängender Blöcke von ↑Daten vom/zum ↑Hauptspeicher Die ↑Zugriffszeit auf ein einzelnes Datum entspricht der ↑Taktfrequenz der CPU. Seine Kapazität reicht von 64 ↑Byte (Fairchild F8, 1975) bis zu einem als Notizblock- oder Zwischenspeicher konfigurierbaren Anteil von 16 GiB (Intel Knights Landing, 2013).

numeric punch card (dt.) ↑Lochkarte.

numerische Adresse ↑Adresse, die sich nur aus Ziffern zusammensetzt und eine Zahl darstellt. Beispiele dafür sind ↑reale Adresse, ↑logische Adresse oder ↑virtuelle Adresse einerseits und ↑Indexknotennummer oder ↑Blocknummer andererseits.

Nutzdaten Bezeichnung für ↑Daten, die bei Durchführung einer bestimmten ↑Aufgabe gebraucht (insb. auch Eingabe), verarbeitet, erzeugt, gespeichert und dargestellt (insb. auch Ausgabe) werden.

NVRAM Abkürzung für (en.) *non-volatile RAM*, (dt.) nichtflüchtiger ↑RAM.

object module (dt.) ↑Objektmodul.

Objective-C Programmiersprache: imperativ, objektorientiert (1981). Obermenge von ↑C.

Objekt Gegenstand, auf den das Handeln von einem ↑Subjekt gerichtet ist (in Anlehnung an den Duden); ↑Exemplar einer Datenstruktur, das eine bestimmte Region (↑Adresse oder ↑Adressbereich) im ↑Speicher belegt und einen Wert repräsentieren kann.

Objekt erster Klasse Exemplar eines bestimmten Bautyps, das im ↑Arbeitsspeicher abgelegt (d.h., einer Variablen zugewiesen) werden, ↑tatsächlicher Parameter beziehungsweise Funktionsergebnis sein oder zur ↑Laufzeit erstellt werden kann und eine eigene Identität besitzt. Für solch ein Objekt oder Konstrukt gibt es in seinem Bezugssystem (d.h., ↑Programm) keine Einschränkung in der Erzeugung oder Nutzung.

Objektkode Resultat der ↑Übersetzung, insbesondere ↑Assemblierung, von den in einem ↑Quellmodul enthaltenen Konstrukten für ein ↑Programm. Anweisungen, die überwiegend in Form von ↑Maschinenkode vorliegen, aber auch vorinitialisierte Daten umfassen. Der für ein ↑Maschinenprogramm zur Ausführung (in den ↑Arbeitsspeicher zu ladende) relevante Anteil von Text- und Datenbeständen in einem ↑Objektmodul.

Objektmodul ↑Datei mit einem ↑Programm als Eingabe für einen ↑Binder, gleichfalls aber auch Ausgabedatei von einem ↑Assemblierer. Neben ↑Text oder ↑Daten, ist die ↑Symboltabelle wichtigstes Merkmal dieser Datei.

offline scheduling (dt.) ↑vorlaufende Planung.

Oktett ↑Byte als Gruppierung gegliedert in exakt 8 Bits.

on-line scheduling (dt.) ↑mitlaufende Planung.

OpenVMS Mehrplatz-/Mehrbenutzerbetriebssystem, erste Installation 1977 (DEC ↑VAX). Eine Weiterentwicklung von ↑VMS insbesondere für Prozessoren der Klasse ↑Alpha. Darüberhinaus erfolgten Portierungen auf 64-Bit Prozessoren von Intel (Itanium) und AMD (x86-64).

operating mode (dt.) ↑Arbeitsmodus.

operating system (dt.) ↑Betriebssystem.

Operationskode Identifikation für einen Befehl einer (realen/virtuellen) Maschine. Beispielsweise der ↑Maschinenkode, der den ↑Maschinenbefehl einer CPU spezifiziert. Allgemein aber die Nummer des Befehls, der von einer Maschine durchgeführt werden soll.

Operationsprinzip Definition des funktionellen Verhaltens der ↑Architektur einer (realen/virtuellen) Maschine durch Festlegung einer ↑Informationsstruktur für diese Maschine und einer ↑Kontrollstruktur für den Ablauf von einem ↑Programm. Beide Strukturformen definieren die ↑Rechnerarchitektur.

operator (dt.) Bedienungsperson. Jemand, dessen Aufgabe es ist, maschinelle Anlagen oder Rechner zu kontrollieren und zu bedienen (Duden); Operateur.

operator panel (dt.) Bedienpult, Bedienungsfeld. Ursprünglicher Arbeitsplatz (↑Systemkonsole) des Betreuungspersonals (↑*operator*) von einem ↑Rechensystem.

optimistische Nebenläufigkeitssteuerung Gegenteil von ↑pessimistische Nebenläufigkeitssteuerung: jeder ↑Prozess lässt die kritische ↑Aktion oder ↑Aktionsfolge als eine ↑Transaktion stattfinden. Scheitert die Transaktion für einen Prozess, wird sie von ihm wiederholt bis sie gelingt oder der Prozess eine ↑Ausnahme erhebt. Typisches Beispiel für solch ein Ablaufmuster ist die ↑nichtblockierende Synchronisation. Dem Ansatz liegt die Annahme zugrunde, wonach ein Prozess mit anderen Prozessen nicht in Konflikt gerät, da eine gemeinsame und gleichzeitige Transaktion eher unwahrscheinlich ist.

orthogonaler Befehlssatz Bezeichnung für einen ↑Befehlssatz (↑ISA), bei der jeder ↑Maschinenbefehl jede ↑Adressierungsart verwenden kann. In solch einer ↑Rechnerarchitektur variieren Befehlstyp und Adressierungsart unabhängig voneinander. Ihr Befehlssatz macht es nicht zur Auflage, dass bestimmte Befehle nur spezielle ↑Register verwenden dürfen.

Ortstransparenz Eigenschaft, durch die einem ↑Subjekt (↑Prozess) der tatsächliche Ort von einem ↑Objekt verborgen bleibt. Der für den Zugriff verwendete ↑Name enthält keinen Hinweis darüber, ob das Objekt sich mit dem Subjekt denselben ↑Adressraum, ↑Rechenkern oder ↑Rechner teilt. Ein solcher Name repräsentiert eine ↑symbolische Adresse.

OS Abkürzung für (en.) *operating system*, (dt.) ↑Betriebssystem (BS).

overhead (dt.) ↑Betriebslast, ↑Gemeinkosten.

overlapped I/O (dt.) ↑überlappte Ein-/Ausgabe.

overlay (dt.) ↑Überlagerung.

own variable (dt.) ↑Eigenvariable.

P Abkürzung für (hol., Kunstwort) *prolaag*, für (dt.) erniedrigen; synonym zu (en.) *down*, *wait*, *acquire*.

page (dt.) ↑Seite.

page descriptor (dt.) ↑Seitendeskriptor.

page fault (dt.) ↑Seitenfehler.

page frame (dt.) ↑Seitenrahmen.

page table (dt.) ↑Seitentabelle.

paged segmentation (dt.) ↑seitennummerierte Segmentierung.

pager (dt.) ↑Seitenabrufer.

paging (dt.) ↑Seitenumlagerung, -adressierung, -nummerierung.

paging unit (dt.) ↑Seitenadressierungseinheit.

panic (dt.) ↑Panik.

Panik Gefahr für das Rechensystem bedeutende und im ↑Betriebssystem aufgetretene ↑Ausnahmesituation, die nicht behandelbar ist und den sofortigen Stillstand bewirkt.

parallel processing (dt.) ↑Parallelverarbeitung.

paralleler Prozess Synonym zu ↑nichtsequentieller Prozess.

paralleles Programm Synonym zu ↑nichtsequentielles Programm.

Parallelität ↑Nebenläufigkeit, wenn in einem ↑Rechensystem die Unabhängigkeit von Vorgängen gegeben ist. Dabei ist ein solcher Vorgang ein ↑Prozess, der durch Vervielfachung oder Mehrfachnutzung von einem ↑Prozessor echt- oder pseudoparallel zu anderen Prozessen stattfindet. Voraussetzung dabei ist die Fähigkeit zur ↑Parallelverarbeitung durch ein ↑Betriebssystem.

Parallelrechner Bezeichnung für einen ↑Rechner, der aus zwei oder mehreren realen Prozessoren aufgebaut ist und vor allem echte ↑Parallelverarbeitung unterstützt. Neben der Anzahl der Prozessoren, die millionenfache ↑Parallelität bedeuten kann — etwa Tihane-2: 32 000 Mehrkernprozessoren (Intel Xeon) jeweils 12-fach parallel (384 000 Kerne) plus 48 000 Hochleistungsprozessoren (Intel Xeon Phi) jeweils 57-fach parallel (2 736 000 Kerne), zusammen 3 120 000 Kerne — und einen entsprechend komplexen Systemaufbau mit sich bringt, spielt die Organisation von dem ↑Arbeitsspeicher eine zentrale Rolle in solchen Rechnern. Die Prozessoren können speicher- oder nachrichtengekoppelt sein. Letzteres impliziert für jeden ↑Prozess, dass der Austausch von ↑Daten mit anderen Prozessen immer nur indirekt geschehen kann (↑*message passing*). Demgegenüber bedeutet die speichergekoppelte Organisation der Prozessoren die Möglichkeit zum direkten Datenaustausch (↑*shared memory*), also ohne zwingend ↑Nachrichtenversenden betreiben zu müssen. In diesem Fall ist jedoch auf die ↑Speicherkonsistenz zu achten: für gewöhnlich ist in solchen Konstellationen nicht mehr garantiert, dass die Speicheroperationen von allen Prozessoren in derselben (sequentiellen) Reihenfolge sichtbar sind. So kann es zum Konflikt (↑*race condition*) kommen, wenn von verschiedenen Prozessoren aus auf dieselbe (Inkohärenz) oder eine andere (Inkonsistenz) gemeinsame Variablen zugegriffen wird. Ein ↑nichtsequentielles Programm für solche Rechner muss daher Grad und Art der durch die Hardware zugesicherten *Kohärenz* im ↑Zwischenspeicher und *Konsistenz* im ↑Arbeitsspeicher reflektieren. Sind die Zusicherungen zu schwach, müssen in dem Programm selbst Vorkehrungen zur Durchsetzung des jeweils geforderten Stärkegrads getroffen werden. Dies kann so weit gehen, speichergekoppelte Rechner als ein nachrichtengekoppeltes (verteiltes) System aufzufassen und grundsätzlich direktem Datenaustausch vorzubeugen.

Parallelverarbeitung ↑Betriebsart von einem ↑Rechensystem, durch die mehrere Abläufe von demselben ↑Programm oder verschiedener Programme parallel stattfinden können. Erreicht wird dies durch Vervielfachung oder Mehrfachnutzung des Prozessors, das heißt, räumliche oder zeitliche Partitionierung seiner Verarbeitungseinheiten. Ersteres begründet beispielsweise einen ↑Multiprozessor oder ↑Mehrkernprozessor, letzteres ist durch ↑partielle Virtualisierung eben nur der Verarbeitungseinheit erreichbar. Sind mehrere Abläufe in demselben Programm möglich, wird letzteres als *nichtsequentiell* bezeichnet.

Paravirtualisierung Form der ↑Selbstvirtualisierung: die ↑Virtualisierung erfolgt „nebenher“ oder „entlang“ (gr. *para*) der gewöhnlichen Funktion von einem ↑Betriebssystem. Im Unterschied zur ↑Vollvirtualisierung ist dem Betriebssystem die Tatsache bekannt, dass die Hardware (↑CPU, ↑Peripherie), die es eigentlich verwaltet, virtualisiert wird. So setzt das

Betriebssystem, wenn ein ↑sensitiver Befehl durch die (virtuelle) CPU ausgeführt werden soll, einen Aufruf an den ↑Hypervisor ab, der die Ausführung dieses Befehls dann so vornimmt, dass keine andere ↑virtuelle Maschine dadurch beeinträchtigt wird. Beispiel für solch einen Befehl ist etwa die ↑Unterbrechungssperre einer CPU (`cli` bei ↑x86): wäre ein solcher Befehl nicht virtualisierbar, würde seine direkte Ausführung in einer virtuellen Maschine die Unterbrechungsperre der realen Maschine setzen und damit den weiteren Betrieb anderer virtueller Maschinen zeitweilig aussetzen. In dem Fall sperrt der Hypervisor eine ↑Unterbrechung nur in der virtuellen Maschine, für die die Unterbrechungssperre wirken soll. Der Hypervisor unterbindet damit also keine ↑Unterbrechungsanforderung, die an die reale Maschine (d.h., CPU) geht. Andere Beispiele dieser Virtualisierungsart liefern ↑Gerätetreiber, aber auch die ↑Ablaufplanung in dem Betriebssystem: muss die virtuelle Maschine ↑Echtzeit zusichern, ist dem Hypervisor nötiges Wissen zu übermitteln, damit dieser die virtuelle Maschine rechtzeitig wieder in Betrieb nehmen kann, so dass jeder auf dieser Maschine stattfindende ↑Prozess seine ↑Echtzeitbedingung einhalten kann. Auch wenn diese Form der Virtualisierung demnach intransparent ist für das Betriebssystem, so ist sie transparent für jedes ↑Maschinenprogramm.

partielle Interpretation Ausführung der Anweisung einer (realen/virtuellen) Maschine durch verschiedene ↑Interpreter, deren Anordnung zueinander durch eine ↑funktionale Hierarchie definiert ist. In diesem arbeitsteiligen und strikt stapelorientierten Vorgang kooperieren Interpreter, die einerseits eine reale und andererseits wenigstens eine ↑virtuelle Maschine implementieren. Dabei definiert der Interpreter der realen Maschine die unterste Ebene in der Hierarchie, er startet den ↑Abruf- und Ausführungszyklus zur Interpretation. Eine ↑Ausnahmesituation, die zur ↑Unterbrechung der gegenwärtigen ↑Aktion auf tieferer Ebene führt, kann bewirken, dass ein Interpreter auf nächst höherer Ebene gestartet und somit eine virtuelle Maschine aktiviert wird. Dieser nachgeschaltete Interpreter läuft als Teil der ↑Ausnahmebehandlung, er sorgt letztlich dafür, dass die unterbrochene Aktion fortgesetzt wird. Entweder vollendet er diese Aktion, beendet damit auch die durch ihn bereitgestellte virtuelle Maschine, oder er leitet seinerseits eine Ausnahmebehandlung ein, aktiviert damit eine weitere virtuelle Maschine (Rekursion). Beendigung einer virtuellen Maschine bedeutet in solch einem Szenario, dass die (reale/virtuelle) Maschine, die zuvor aktiv war, die Kontrolle über die weitere Programmausführung zurück erhält. Typische Beispiele für diese Art der Interpretation sind der ↑Systemaufruf und ein ↑Seitenfehler, mit dem ↑Betriebssystem als Interpreter. Ein anderes Beispiel liefert ein ↑sensitiver Befehl, der durch einen ↑Hypervisor interpretiert werden muss. In jedem dieser Fälle fährt zeitweilig eine virtuelle Maschine hoch, um unterbrochene Aktionen zu vollenden und sich dann alsbald wieder abzuschalten.

partielle Virtualisierung Mehrfachnutzung einer ausgewählten Hardwareeinheit in einem ↑Rechensystem durch ein ↑Zeitteilverfahren. Ursprünglich nur bezogen auf die ↑CPU, um nämlich einem ↑Prozess die Illusion von einem eigenen Prozessor zu geben. Technische Grundlage dafür sind Systemfunktionen zur ↑Simultanverarbeitung. So können mehrere Prozesse zugleich stattfinden, indem jedem ein eigener virtueller Prozessor zur Verfügung steht. Das Konzept ist aber übertragbar auf andere Hardwareeinheiten, insbesondere dem ↑Hauptspeicher. Hier kann durch zeitweilige ↑Umlagerung ganzer Programme in den ↑Hintergrundspeicher derselbe Hauptspeicherbereich zur zeitweiligen Mehrfachnutzung durch mehrere Programme bereitgestellt werden.

Partition Teilbereich bestimmter Größe auf einem ↑Datenträger. Eine solche Aufteilung wird üblicherweise für ↑Ablagespeicher genutzt und besteht dann aus einem ↑Dateisystem, sie kann aber auch exklusiv als ↑Umlagerungsbereich konfiguriert sein. Auf demselben Datenträger können mehrere solcher Teilbereiche (ggf. unterschiedlicher Größe) eingerichtet sein.

partition (dt.) Teilung, ↑Partition.

passives Warten Verfahren, bei dem ein ↑Prozess untätig ein bestimmtes ↑Ereignis erwartet: im Gegensatz zu ↑aktives Warten, gibt der Prozess seinen ↑Prozessor während der gesamten

↑Wartezeit ab und blockiert von sich aus (↑Prozesszustand). Wenn im Moment der Blockierung ein anderer Prozess bereit steht, löst der im ↑Betriebssystem mitlaufende ↑Planer einen ↑Prozesswechsel aus. Tritt das erwartete Ereignis ein, wird der betreffende Prozess dadurch (genauer: durch den das Ereignis herbeiführenden (externen) Prozess) deblockiert und von dem Planer bereit gestellt oder sofort (d.h., als laufend) eingesetzt, je nach dem Verfahren zur ↑Prozesseinplanung (d.h.,) und ↑Operationsprinzip (d.h., erlaubter ↑Verdrängungsgrad) des Betriebssystems.
Dieses ↑Warteverhalten trägt zur Maximierung der ↑Auslastung einerseits der ↑CPU und andererseits der ↑Peripherie bei. Im Falle der CPU wird ein anderer Prozess produktive Arbeit (↑Rechenstoß) vollbringen können, dabei gegebenenfalls auch einen weiteren ↑Ein-/Ausgabestoß auslösen und somit ein ↑Peripheriegerät beanspruchen. Im Falle der Peripherie werden von verschiedenen Prozessen abgesetzte Ein-/Ausgabestöße mehrere Peripheriegeräte zugleich beschäftigen können. Letzterer Aspekt bedeutet aber auch, dass der auf der CPU gegenwärtig stattfindende Prozess durch eine ↑Unterbrechungsanforderung verzögert werden dann, die ihre Ursache in der Beendigung des Ein-/Ausgabestoßes eines anderen (vorigen) Prozesses hat. Sogar mehrere solcher Stöße könnten nebenläufig zum gegenwärtigen Prozess, der womöglich keinen eigenen Ein-/Ausgabestoß auslöst, stattfinden und alle könnten am Ende diesen Prozess unterbrechen. Damit beeinflussen sich Prozesse bei diesem Warteverfahren indirekt, obwohl sie nicht direkt gekoppelt sind und gegebenenfalls auch nichts voneinander wissen: es besteht ein hohes Potential für ↑Interferenz kausal unabhängiger Prozesse. Zudem ist der im Vergleich zu aktivem Warten anfallende Mehraufwand (↑*overhead*) für die gesamte ↑Aktionsfolge bis einschließlich Prozesswechsel nicht unerheblich. Dieser erhöht grundsätzlich die ↑Latenzzeit für jeden Prozess, der warten muss und er geht erst bei genügend viel produktive Arbeit, die während der Wartezeit vollbracht werden kann, im ↑Hintergrundrauschen unter. Latenzzeit wie auch Produktivitätsmaß sind noch bestimmbar, einerseits durch statische Analyse von dem zum Prozesswechsel führenden ↑Programm im Betriebssystem und andererseits durch dynamische Analyse der ↑Bereitliste, jedoch trifft dies nur bedingt auf die Wartezeit zu (vgl. ↑aktives Warten). Daher ist die Entscheidung, aktiv oder passiv zu warten auch nur bedingt eine effiziente Lösung. Gleiches gilt auch, wenn zunächst nur für eine kurze aktiv und dann, wenn das Ereignis immer noch nicht eingetreten ist, passiv gewartet wird.

PC Abkürzung für (en.) *program counter*, (dt.) ↑Befehlszähler oder *personal computer* (dt.) Arbeitsplatzrechner.

PCB Abkürzung für (en.) *process control block*, (dt.) ↑Prozesskontrollblock.

PCLSRing Ausgesprochen (en.) *program counter lusering*. Der Mechanismus in ↑ITS, um einen ↑Systemaufruf jederzeit quasiatomar ablaufen lassen zu können. In ITS scheint die durch einen Systemaufruf aktivierte ↑Systemfunktion auch trotz einer möglichen ↑Unterbrechungsanforderung immer ununterbrechbar stattzufinden. Kommt es zur ↑Unterbrechung, findet eine von zwei Maßnahmen statt:

1. Der Systemaufruf wird zurückgezogen, indem der ↑Befehlszähler auf den Anfang der Aufrufausführung im ↑Betriebssystem zurückgesetzt wird. Der mittlerweile bereits geänderte Zustand wird gesichert, so dass bei Wiederaufnahme der Ausführung der Systemaufruf dort fortfährt, wo er zuvor unterbrochen worden ist.
2. Der Systemaufruf schließt ab, wenn dies mit nur noch wenigen Befehlen möglich ist.

Dadurch wird sichergestellt, dass kein ↑Prozess einen anderen Prozess (inkl. er selbst), der sich in einem ↑Systemaufruf befindet, observieren kann. In anderen Systemen (↑Mach) findet diese Technik Verwendung, um die ↑Unteilbarkeit einer ↑Aktion (↑Elementaroperation) die emuliert eine teilbare ↑Aktionsfolge bildet, logisch durchzusetzen. Ein ↑kritischer Abschnitt kann darüber ebenfalls abgesichert werden, insbesondere wenn in dem Abschnitt explizit ein ↑Prozesswechsel programmiert ist: nach Wiederaufnahme des weggeschalteten Prozesses, wird dieser in einen Wiederanlauf (*restart*) des kritischen Abschnitts gezwungen.

PDP 11 Kleinrechnerfamilie (1970–1990, DEC): 16-Bit, wortorientiert, mikroprogrammierbar; nahezu ↑orthogonaler Befehlssatz, ↑speicherabgebildete Ein-/Ausgabe; acht ↑Unterbrechungsprioritätsebenen, jeder ↑Unterbrechungsvektor bildet ein Tupel von ↑PC und ↑PSW, das bei einer ↑Unterbrechung in die entsprechenden ↑Prozessorregister geladen wird. Die ↑Rechner fanden insbesondere zum ↑Echtzeitbetrieb weitläufig Verwendung, hier insbesondere unter ↑RSX 11.

PDP 11/40 Kleinrechner (1970) aus der ↑PDP 11 Familie. Für das ↑Betriebssystem standen firmeneigene (insb. ↑RSX 11), aber insbesondere auch eine große Anzahl nicht-proprietärer Lösungen, vor allem ↑UNIX, zur Verfügung.

PDV Abkürzung für ↑Prozessdatenverarbeitung.

Pegelsteuerung Auslöser für die ↑Unterbrechung ist eine Mindestzeit an Beständigkeit des Pegelstands auf der Signalleitung (↑*interrupt line*) zur ↑CPU. Für eine ↑Unterbrechungsanforderung erniedrigt (erhöht) die ↑Peripherie den Pegelstand und hält diesen Zustand, bis der ↑Unterbrechungshandhaber der anfordernden Peripherie die Unterbrechung bestätigt. Im Moment dieser Bestätigung, und zwar an dem entsprechenden ↑Peripheriegerät, wird der dort eingestellte Pegelstand zurückgenommen. Stellt dasselbe Gerät sofort nach der Bestätigung und noch während die ↑Unterbrechungsbehandlung läuft eine weitere Anforderung, kommt diese normalerweise implizit bei der Rückkehr aus der Unterbrechungsbehandlung zur Geltung — oder auch bereits vorher, wozu der Unterbrechungshandhaber jedoch die für die CPU in dem Moment gültige ↑Unterbrechungsprioritätsebene explizit herabsetzen muss (↑privilegierter Befehl). Damit werden auch wiederholte Anforderungen (*reassertion*) nicht verpasst, im Gegensatz zur ↑Flankensteuerung. Jedoch darf der Unterbrechungshandhaber die Bestätigung nicht auslassen, da er sonst bei Rückkehr aus der Unterbrechungsbehandlung sofort wieder aufgerufen wird, damit seinen erneuten Aufruf nicht etwa mit einer weiteren Unterbrechung in Verbindung bringen kann und, was viel gravierender ist, der unterbrochene ↑Prozess womöglich niemals fortgesetzt wird. Auslassen der Bestätigung einer pegelgesteuerten Unterbrechung kann ↑Panik auslösen.

Performanz Leistungsfähigkeit eines einzelnen Systembestandteils, eines Subsystems oder eines ganzen Systems, Hardware oder Software.

peripheral (dt.) ↑Peripheriegerät.

Peripherie Gesamtheit der an einer ↑Zentraleinheit angeschlossenen Geräte, jedes davon auch als ↑Peripheriegerät bezeichnet.

Peripheriegerät ↑Steuerung durch einen ↑Prozessor benötigendes Gerät in einem ↑Rechensystem. Jede Form von Ein-/Ausgabegerät (z.B. Tastatur, Bildschirm, Maus, Platte; Sensoren, Aktoren), aber auch Erweiterungskarten beziehungsweise Anschlüsse zur seriellen/parallelen ↑Ein-/Ausgabe von ↑Daten.

persistentes Betriebsmittel Synonym zu ↑dauerhaftes Betriebsmittel.

Persistenz Speicherbarkeit einer ↑Entität; die Fähigkeit von einem System, seine innere Struktur sowie die Zusammengehörigkeit und gegliederte Zusammenstellung seiner Einzelbestandteile dauerhaft speichern zu können. Hierzu kommt der ↑Ablagespeicher zum Einsatz.

pessimistische Nebenläufigkeitssteuerung Bezeichnung für eine ↑Nebenläufigkeitssteuerung, die die Annahme trifft, dass ein ↑Prozess mit wenigstens einem anderen Prozess wahrscheinlich in Konflikt gerät, wenn eine gemeinsame ↑Aktion oder ↑Aktionsfolge gleichzeitig stattfindet. Dem möglichen Konflikt wird vorgebeugt, indem die Prozesse die betreffende Aktion/Aktionsfolge nur als Ganzes nacheinander durchführen können. Dies lässt sich durch ↑blockierende Synchronisation erreichen oder auf Basis einer ↑Ablaufplanung, die die Prozesse (in der kritischen Phase) strikt nacheinander stattfinden lässt.

Pfadname Benennung einer ↑Entität durch Angabe des Pfads zu ihr im ↑Namensraum. Der Pfad bildet eine Zeichenfolge, in der ↑Name und Trenntext (↑Separator) abwechselnd auftreten, beispielsweise:

- `Multics>ist>aktueller>denn>je` oder
- `UNIX/ist/ohne/Multics/undenkbar` und `lebt/weiter/in/Linux` oder
- `Windows\hat\viel\mehr\von\VMS\als\allgemein\bekannt\ist`

Der abschließende Name, nämlich das Blatt eines baumartig aufgebauten Namensraums, bezeichnet die Entität (im Beispiel: `je`, `undenkbar`, `Linux`, `ist`). Die Namen davor, durch den Trenntext jeweils separiert, bezeichnen einen ↑Namenskontext, der für gewöhnlich — insbesondere in einem ↑Dateisystem — als ↑Verzeichnis implementiert ist.

PGAS Abkürzung für (en.) *partitioned global address space*. Ein ↑virtueller Adressraum, der auseinanderliegende ↑Hauptspeicher verschiedener ↑Rechner zusammenschließt: für gewöhnlich liegen die Hauptspeicher über ein ↑Rechnernetz verteilt vor (↑DSM). In diesem Adressraum ist jeder einzelne Hauptspeicher einem bestimmten ↑Adressbereich eindeutig zugeordnet.

PIC Abkürzung für (en.) *programmable interrupt controller*, (dt.) programmierbare Unterbrechungssteuereinheit.

PID bkürzung für (en.) *process identifaction*, (dt.) ↑Prozessidentifikation.

PIO Abkürzung für (en.) *programmed input/output*, (dt.) ↑programmierte Ein-/Ausgabe.

PL/1 Abkürzung für (en.) *programming language # 1*, prozedurale, imperative Programmiersprache (1964).

placement policy (dt.) ↑Platzierungsstrategie.

Planer ↑Programm zur Erstellung und Verwaltung von einem ↑Ablaufplan, das nach zwei verschiedenen Modellen zum Einsatz kommt: vor (*off-line*, statisch) oder zur (*on-line*, dynamisch) ↑Laufzeit der Abarbeitung des Ablaufplans. Auch eine Kombination beider Varianten ist möglich. In dem Fall reflektiert der statische Ablaufplan für jeden einzelnen darin aufgeführten ↑Prozess Vorwissen zu Daten- und Kontrollflussabhängigkeiten, das dem ↑Betriebssystem initial für eine Gruppe von Prozessen übergeben wird. Zur Laufzeit wird dieser Plan dann entsprechend der dynamischen Vorgänge im ↑Rechensystem fortgeschrieben. Der statische Ansatz findet bevorzugt bei Spezialsystemen Verwendung, sofern das benötigte Vorwissen vorhanden ist und die Aufstellung des Plans in angemessener Zeit möglich ist: die Erstellung eines solchen Plans ist typischerweise ein NP-schweres Entscheidungsproblem, das in Abhängigkeit von der Prozessanzahl wie auch dem Grad der Prozessabhängigkeiten gegebenenfalls eine unvertretbar hohe Berechnungszeit bedingt. Demgegenüber ist der dynamische Ansatz in Spezial- und Universalsystemen verbreitet, in letzteren wegen dem dort oft nicht vorhandenem Vorwissen eben auch unabdinglich. In diesem Fall der (mit den zu planenden Prozessen) im Betriebssystem mitlaufenden Ablaufplanung wird letztere als ↑Unterprogramm aufgerufen, wann immer Gründe für eine Aktualisierung des Plans vorliegen. Diese Gründe sind die Zulassung, Bereitstellung, Blockierung, Unterbrechung, Verdrängung, Suspendierung oder Beendigung eines Prozesses.

Planung (en.) ↑*scheduling*. Ausarbeitung eines Plans, der festlegt, wann ein ↑Auftrag einem ↑Prozessor zur Verarbeitung übergeben werden soll (↑*dispatching*). Je nach Art des Prozessors unterscheiden sich für gewöhnlich die Planungsverfahren, indem ihre Methoden auch technische Merkmale der jeweiligen ↑Bedienstation berücksichtigen. Darüber hinaus ist es nicht ungewöhnlich, für dieselbe Prozessorklasse oder dasselbe Prozessorexemplar mehrere solcher Verfahren zu haben.
Ist der Auftrag ein ↑Prozess und der Prozessor eine ↑CPU oder ein ↑Rechenkern, wird auch

von ↑Ablaufplanung gesprochen. Für gewöhnlich werden hier drei verschiedene Ebenen unterschieden (von oben nach unten): ↑langfristige Einplanung, ↑mittelfristige Einplanung und ↑kurzfristige Einplanung. Aufträge zur Ein- oder Ausgabe von ↑Daten unterliegen ebenfalls einer Planung, die zudem sehr stark durch die technischen Merkmale von dem jeweiligen ↑Peripheriegerät beeinflusst ist. Grundsätzlich ist derartige Planung der Auftragsverarbeitung notwendig, wenn zu einem Zeitpunkt mehr Aufträge anstehen können als Prozessoren für deren Verarbeitung verfügbar sind. Eine zentrale Bedeutung kommt dabei dem ↑Einplanungsalgorithmus zu, der über die Reihenfolge und Häufigkeit der ↑Einlastung eines Prozessors entscheidet.

Plattenspeicher ↑Datenträger, für gewöhnlich in Form einer Scheibe mit einer magnetischen Oberfläche (*platter*). Im Betrieb rotiert die Scheibe mit hoher Geschwindigkeit. Die ↑Daten werden blockweise durch Lese-/Schreibköpfe von der Plattenoberfläche abgerufen beziehungsweise auf die Plattenoberfläche aufgebracht.

Plattensteuerung Bezeichnung für die Steuereinheit (*controller*) von einem ↑Plattenspeicher.

Platzierungsalgorithmus Lösungs- und Bearbeitungsschema, Handlungsvorschrift zur ↑Speicherzuteilung, zentraler Rechenvorgang einer ↑Platzierungsstrategie. Ist charakterisiert durch die Reihenfolge von Einträgen auf einer Liste, die freie Abschnitte beliebiger Länge im ↑Hauptspeicher erfasst (↑*hole list*), und einem oder mehrerer Kriterien, nach denen genau einer dieser Einträge für die Speicherzuteilung auf Anforderung durch einen ↑Prozess ausgewählt wird. Die Einträge können einerseits der Größe nach auf- oder absteigend oder anderseits der ↑Adresse nach und dann in aller Regel nur aufsteigend sortiert sein. Bezugspunkt dabei ist jeweils die Größe beziehungsweise Adresse des durch einen Eintrag repräsentierten freien Hauptspeicherabschnitts.
Bei Sortierung nach aufsteigender Größe (↑*best fit*) ist das Ziel, den ↑Verschnitt möglichst klein zu halten: hier wird der Eintrag gewählt, der die beste Passung, also den kleinsten Rest, bringt. Bleibt ein Rest übrig, muss dieser mit einem zweiten Suchlauf durch die Liste neu einsortiert werden. Da jeder übrig bleibende Rest darüberhinaus einen Eintrag mit noch kleinerer Größe erzeugt, wird ↑externer Verschnitt immer mehr wahrscheinlich. Dieses Problem wird vermieden mit einer Sortierung der Listeneinträge nach absteigender Größe (↑*worst fit*). Hier wird der Eintrag gewählt, der die schlechteste Passung hat, also den größten Rest liefert. Die Annahme dabei ist, dass ein gegebenenfalls übrig bleibender Rest wahrscheinlich immer noch groß genug für eine nachfolgende Speicherzuteilung und damit also brauchbar ist. Bleibt ein Rest übrig, muss auch dieser mit einem zweiten Suchlauf durch die Liste neu einsortiert werden.
Wiederholte Suchläufe werden vermieden, wenn die Einträge der Adresse nach sortiert sind und die Speicherzuteilung immer den hinteren Teil eines genügend großen Abschnitts zurück liefert. Ein sehr einfaches Verfahren ist es dann, die Liste immer vom Anfang her nach dem erstbesten Abschnitt (↑*first fit*) zu durchsuchen. Dieses Verfahren hinterlässt vorne eher kleinere Einträge und führt somit dazu, dass nachfolgende Suchläufe wahrscheinlich länger benötigen, um überhaupt in die Nähe brauchbar großer Abschnitte zu kommen. Das lässt sich vermeiden, indem ein Suchlauf immer nach dem Eintrag startet, der im vorangegangenen Suchlauf für die Speicherzuteilung verwendet wurde: ab dieser Stelle wird der nächstbeste Abschnitt (↑*next fit*) auf der Liste gesucht. Am Ende der Liste angekommen wird ab Listenanfang bis zum Startpunkt weitergesucht (↑*circular first fit*).
Wurde die Liste komplett abgesucht, ohne einen passenden freien Abschnitt zu finden, scheitert bei allen Verfahren die Speicherzuteilung. Solche Fehlläufe werden noch durch das Phänomen der ↑Fragmentierung begünstigt, mit dem alle Verfahren konfrontiert sind, die mit freien Speicherabschnitten beliebiger und unterschiedlicher Länge umgehen müssen. Fragmentierung kann jedoch durch ↑Verschmelzung eines frei werdenden Abschnitts mit einem oder zwei auf der Liste bereits verzeichneten freien Abschnitten vermindert werden. Dadurch werden einerseits größere und andererseits weniger freie Abschnitte generiert, was die Wahrscheinlichkeit von Fehlläufen senkt (größere freie Abschnitte) und den Aufwand für Suchläufe

reduziert (weniger Listeneinträge).

Platzierungsstrategie Verfahrensweise nach der die ↑Speicherzuteilung geschieht. Sie bestimmt die ↑Adresse von einem freien Platz (↑*hole*) im ↑Arbeitsspeicher, wo nach erfolgter Speicherzuteilung an einen ↑Prozess Bestände von ↑Text oder ↑Daten entsprechend der Platzgröße abgelegt werden können. Die freien Plätze stehen auf einer Liste (↑*hole list*), deren Organisation und Repräsentation einerseits vom ↑Platzierungsalgorithmus und andererseits vom Geltungsbereich (↑Laufzeitsystem/↑Betriebssystem) der Speicherzuteilung abhängig ist. Ist letzterer das Betriebssystem, bestimmt die Art des Strukturelements (↑Seite, ↑Segment) von einem ↑Prozessadressraum maßgeblich die technische Auslegung dieser Liste. Liegt dem ↑Prozess ein ↑seitennummerierter Adressraum zugrunde, erfolgt die Speicherzuteilung seitenweise und damit in einer Größe, die immer Vielfaches der Größe einer ↑Seite ist. Da jede Seite immer nur in exakt einen ↑Seitenrahmen „eingespannt“ (platziert) wird und alle Seitenrahmen gleich groß (wie auch die Seiten) sind, ist jeder freie Seitenrahmen gut zur Platzierung einer Seite im ↑Hauptspeicher. Um zu verbuchen, ob ein Seitenrahmen frei oder belegt ist, reicht ein Bit: 1 = `true` = frei (Loch), 0 = `false` = belegt. Alle Seitenrahmen des Hauptspeichers werden dann über eine Bitleiste (*bit string*) erfasst, die als Tabelle (*bit map*) gespeichert ist. Für eine Speicherzuteilung von n_b ↑Byte sind dann $n_p = (n_b + sizeof(page) - 1)/sizeof(page)$ beliebige Bits in dieser Bitleiste zu finden, die jeweils einen freien Seitenrahmen markieren (d.h., auf 1 gesetzt sind). Bestimmt dagegen ein ↑segmentierter Adressraum das Herrschaftsgebiet von einem Prozess, erfolgt die Speicherzuteilung segmentorientiert und damit in einer Größe, die immer Vielfaches der Größe des Strukturelements eines Segments ist. Handelt es sich bei diesem Element um eine Seite (↑*segmented paging*), ist der Fall allerdings einfach und die Zuteilung kann auf Basis einer Bitleiste wie eben zuvor beschrieben geschehen. Bei „reiner“ ↑Segmentierung jedoch ist das Byte das Strukturelement eines Segments. In dem Fall kann die Größe jedes Segments einer beliebigen Anzahl von Byte im Bereich $[0, 2^n - 1]$ entsprechen. Die Segmentgröße ist einerseits durch das ↑Programm und andererseits durch einen Prozess in dem Programm bestimmt. Verschiedene Prozesse (verschiedener Programme) beanspruchen damit verschieden große Segmente im Hauptspeicher, um stattfinden zu können. Enden Prozesse und werden daraufhin die durch sie beanspruchten Segmente im Hauptspeicher freigegeben, entstehen freie Bereiche (Löcher) unterschiedlicher Größe. Eine Bitleiste zur Verwaltung dieser freien Bereiche scheidet wegen dem sehr ungünstigen Kosten-Nutzen-Verhältnis aus, da pro Byte ein Bit notwendig wäre: bei einem 4 GiB großen Bereich im Hauptspeicher könnte (bei erlaubter kleinster Segmentgröße von einem Byte) jedes zweite Byte belegt/frei sein, was eine Bitleiste von 2^{31} Bits Länge beziehungsweise Bittabelle von 256 MiB Größe beanspruchen würde. Um die Bitleiste auch hier effizient nutzen zu können, müsste einem Segment ein viel größeres Strukturelement (z.B. eine Seite) zugrunde gelegt werden. Bleibt es jedoch bei dem Byte, muss ein Eintrag auf der Liste freier Bereiche im Hauptspeicher die ↑Adresse und die Länge (in Byte) eines solchen Bereichs verzeichnen. Da die Anzahl dieser Einträge variiert, ist bei der technischen Umsetzung der Liste auf eine geeignete *dynamische Datenstruktur* zurückzugreifen. Dabei sind die Einträge in dieser Datenstruktur nach Kriterien sortiert, die der Platzierungsalgorithmus jeweils vorgibt. Dieser Ansatz wird nicht nur von einem Betriebssystem verfolgt, wenn segmentierte Adressräume zu verwalten sind, sondern auch von einem Laufzeitsystem, wenn nämlich der ↑Haldenspeicher innerhalb eines Prozessadressraums verwaltet werden muss.

plotter (dt.) ↑Kurvenschreiber.

plug and play (dt.) sofort betriebsbereit; automatische Inbetriebnahme von einem ↑Peripheriegerät allgemein im Moment seines physischen Anschlusses an das ↑Rechensystem, speziell auch beim Einlegen von einen diesbezüglichen ↑Wechseldatenträger (z.B. ein Speicherstab, ↑USB) und dem implizit folgenden ↑Einhängen von dem darauf gespeicherten ↑Dateisystem.

port-mapped I/O (dt.) ↑isolierte Ein-/Ausgabe, mittelbar durch einen Anschluss (*port*).

power-on reset (dt.) ↑Einschaltrückstellung.

PowerPC ↑Rechnerarchitektur einer von den Unternehmen Apple, IBM und Motorola gemeinsam spezifizierten ↑CPU (1991). Ein 64-Bit ↑RISC, von dem jedoch auch 32-Bit Varianten vor allem für den Bereich eingebetteter Systeme existieren.

präemptive Planung (en.) ↑*preemptive scheduling*. Modell der ↑Ablaufplanung, durch die ↑Prozesse gezwungen werden können, die Kontrolle über den ↑Prozessor ab- und an einen anderen Prozess weiterzugeben. Anders als ↑kooperative Planung kann einem im ↑Prozesszustand laufend befindlichen Prozess sein Prozessor zugunsten eines auf der ↑Bereitliste stehenden Prozesses entzogen werden (↑*preemption*). Beispiele für solch ein Verfahren sind ↑RR, ↑VRR, ↑SRTF und ↑MLFQ.
Je nach dem vor allem durch Struktur und Aufbau des ↑Betriebssystems möglichen ↑Verdrängungsgrad wird der erzwungene ↑Prozesswechsel mehr oder weniger zeitnah zu dem ↑Ereignis stattfinden, das die Ursache für die Neuzuteilung des Prozessors ist. Typischerweise ruft eine ↑Unterbrechungsanforderung dieses Ereignis hervor, in deren weiteren Behandlungsverlauf es dann zum Aufruf des ↑Planers kommen kann. ↑Einplanung und ↑Einlastung sind sowohl räumlich als auch zeitlich gekoppelt, wobei letztere jedoch immer nur dann zum zeitnahen Prozesswechsel führt, wenn erstere dies auch vorgibt. Kommt es zur Entscheidung, den Planer aufzurufen, bestimmen ↑Einplanungslatenz und ↑Einlastungslatenz zusammen die Zeitspanne bis zum Prozessorentzug beziehungsweise zur Prozessorzuführung.

pre-paging (dt.) ↑Seitenvorabruf.

preemption (dt.) ↑Verdrängung.

preemption point (dt.) ↑Verdrängungspunkt.

preemptive scheduling (dt.) ↑präemptive Planung.

present bit (dt.) ↑Anwesenheitsbit.

Primärspeicher Klassifikation für ↑Registerspeicher, ↑Zwischenspeicher, ↑Notizblockspeicher und ↑Hauptspeicher/↑Arbeitsspeicher; „erstrangiger Speicher“, der zwar über keine große Kapazität verfügt, dafür jedoch eine niedrige ↑Zugriffszeit mit sich bringt.

primitive Koroutine Bezeichnung für eine ↑Koroutine, die zusammen mit anderen ihrer Art ein und denselben ↑Laufzeitstapel besitzt (analog zur ↑Routine). Die ↑Handhabe für einen ↑Handlungsstrang innerhalb einer solchen Koroutine ist ein ↑Befehlszählerwert. Dieser Wert ist der Zeiger auf den ↑Maschinenbefehl, der nach erfolgtem ↑Koroutinenwechsel als nächster ausgeführt wird. Anders als eine ↑komplexe Koroutine.

principle of locality (dt.) ↑Lokalitätsprinzip.

Priorität Stellenwert, den ein ↑Prozess innerhalb einer Rangfolge einnimmt (in Anlehnung an den Duden). Dieser Wert kann *statisch* (unveränderlich) oder *dynamisch* sein. Im statischen Fall, erhält der Prozess spätestens bei seiner Erzeugung einen bestimmten Rang und behält diesen bis zu seiner Zerstörung. Demgegenüber wird im dynamischen Fall der Rang eines Prozesses während seiner Lebenszeit vom ↑Einplanungsalgorithmus aktualisiert. Beispielsweise kann bei ↑Echtzeitbetrieb der Rang eines Prozesses umso mehr ansteigen, je weniger Zeit dem Prozess bis zum Termin bleibt (↑EDF). Im ↑Dialogbetrieb kann der Rang eines Prozesses umso mehr ansteigen, je kürzer sein ↑Rechenstoß ist. Dem liegt die Annahme zugrunde, dass ein kurzer Rechenstoß auf einen ein-/ausgaberührigen und damit interaktiven Prozess hindeutet, dem eine möglichst kurze ↑Antwortzeit zu geben ist. Umgekehrt kann der Rang eines Prozesses umso mehr abnehmen, je länger sein Rechenstoß und damit auch seine Phase der Interaktionslosigkeit dauert.

privater Semaphor Bezeichnung für einen ↑Semaphor, bei dem ↑P nur für denjenigen ↑Prozess möglich ist, der Eigentümer dieses Semaphors ist. Demgegenüber ist ↑V jedem anderen

Prozess möglich. Originär ein ↑allgemeiner Semaphor mit Wertebereich $[-1, 1]$, implizit vorbelegt mit dem Wert 0. Alternativ kann auch ein ↑binärer Semaphor die Basis bilden, mit `false` als Initialwert. Hauptzweck dieses Semaphors besteht in der ↑Synchronisation eines Prozesses auf ein ↑Ereignis, das den weiteren Verlauf des Prozesses bestimmt. Das Ereignis kann von einem beliebigen Prozess, auch ihm selbst, signalisiert werden, oft als Ergebnis der bei einer Berechnung erreichten Zustandsänderung. Varianten erlauben die Speicherung von mehr als einem Ereignis und umfassen demzufolge den Wertebereich $[-1, n]$, mit $n \geq 1$.

privileged mode ↑*system mode*.

privilegierter Befehl ↑Maschinenbefehl, dessen direkte Ausführung durch eine (reale/virtuelle) Maschine nur erlaubt ist, wenn die ↑CPU im privilegierten ↑Arbeitsmodus tätig ist.

probabilistic scheduling (dt.) ↑probabilistische Planung.

probabilistische Planung (en.) ↑*probabilistic scheduling*. Modell der ↑Ablaufplanung unter Berücksichtigung von Unsicherheiten oder Wahrscheinlichkeiten zu einer oder mehrerer ↑Prozessgrößen. Im Gegensatz zu ↑deterministische Planung geschehen die ↑Prozesse zufällig, zu unvorhersehbaren Zeitpunkten und oft auch für unvorhersehbare Zeitspannen. Die Prozessreihenfolge ist nicht durch Vorbedingungen im Voraus festgelegt, sondern entsteht erst zur ↑Laufzeit und als Folge von ↑Ereignissen.
Charakteristisches Merkmal ist die ↑mitlaufende Planung, da ↑Vorwissen zur Bildung und Aktualisierung der Prozessreihenfolge nicht zur Verfügung steht und die benötigten Informationen nur durch Beobachtung der Prozesse selbst gesammelt werden können. Grundlage bildet somit *dynamisches Wissen* über den gegenwärtigen Systemzustand, von dem ausgehend Entscheidungen für zukünftige Prozesse getroffen werden. Die mit diesen Entscheidungen einhergehende Optimierung in Bezug auf das ↑Einplanungskriterium für einen Prozess ist nur näherungsweise möglich. Die entstehende Prozessreihenfolge reflektiert dieses Wissen entsprechend eines oder mehrerer dieser Kriterien und wird dem angestrebten Optimierungsziel wahrscheinlich nahe kommen. Typische Beispiele sind ↑SPN, ↑HRRN und ↑SRTF.

process control (dt.) ↑Prozesssteuerung, Prozessverarbeitung.

process descriptor (dt.) Prozessdeskriptor, ↑Prozesskontrollblock.

process factor (dt.) ↑Prozessgröße.

process scheduling (dt.) ↑Prozesseinplanung.

process table (dt.) ↑Prozesstabelle.

process-based operating system (dt.) ↑prozessbasiertes Betriebssystem.

processing time (dt.) ↑Bearbeitungszeit.

processor status word (dt.) ↑Prozessorstatuswort.

Produktionsbetrieb Bezeichnung für den Betrieb eines ↑Rechensystems, um Leistungsdaten zu produzieren. Neben einer Funktionsprüfung geschieht dabei auch die Bestimmung relevanter Systemparameter (d.h., nichtfunktionaler Merkmale) in Abhängigkeit von einem vorgegebenen *Anwendungsprofil*. Je nach ↑Betriebsart des Rechensystems finden diese Parameter Berücksichtigung als ↑Einplanungskriterium, hier insbesondere ↑Antwortzeit, ↑Durchlaufzeit, ↑Durchsatz und ↑Prozessorauslastung.

program status word (dt.) ↑Programmstatuswort.

Programm Festlegung einer Folge von Anweisungen für einer ↑Prozessor, nach der die zur Bearbeitung einer (durch einen Algorithmus wohldefinierten) Handlungsvorschrift erforderlichen Aktionen stattfinden sollen; Konkretisierung eines Algorithmus. Von statischer Natur, erst durch den Prozessor kommen die Anweisungen zur Ausführung.

Programmablauf ↑Prozess.

Programmbibliothek Gesamtheit mehrerer häufig verwendeter Softwarebestände (↑Modul, ↑Unterprogramm, ↑Exemplar eines Datentyps), die in wenigstens einem ↑Objektmodul enthalten sind. Typischerweise sind jedoch mehrere Objektmodule in der ↑Bibliothek zusammengefasst, wie beispielsweise im Falle der ↑libc. In einer solchen Bibliothek ist jedem einzelnen Softwarebestand ein ↑Name (↑Symbol) eindeutig zugeordnet. Die Bibliothek selbst ist in Form einer ↑Datei im ↑Ablagespeicher vorrätig.

Programmiermodell Bezeichnung für den in einem ↑Programm direkt sicht-, nutz- oder programmierbaren ↑Registersatz von einem ↑Prozessor.

programmierte Ein-/Ausgabe Methode des Transfers von ↑Daten zwischen ↑Peripherie und ↑Hauptspeicher unter Hilfestellung durch die ↑CPU. Im Gegensatz zu ↑DMA führt die CPU ein ↑Programm aus, um ein ↑Byte nach dem anderen zu transferieren. Die damit verbundene Ein-/Ausgabe in Bezug auf einen bestimmten ↑Speicherbereich läuft voll und ganz unter Kontrolle der CPU ab. Folglich kann die CPU in der Zeit keine anderen Berechnungen durchführen: Ein-/Ausgabe und Berechnungen überlappen sich nicht.

Programmstatuswort Abkürzung ↑PSW, Jargon (IBM) für die Kombination von ↑Prozessorstatuswort und ↑Befehlszähler.

progress guarantee (dt.) ↑Fortschrittsgarantie.

PROM Abkürzung für (en.) *programmable ROM*, (dt.) programmierbarer ↑ROM.

Proportionalität Verhältnismäßigkeit, Angemessenheit (Duden). In Bezug auf ein ↑Betriebssystem ist damit insbesondere das Verhältnis zwischen der für einen Anwendungsfall akzeptablen Leistung einerseits und der technisch erreichbaren effektiven Leistung andererseits gemeint. Nicht immer stößt das technisch Machbare auf anstandslose Akzeptanz beim Menschen. Die Konsequenz daraus kann sein, ein ↑Rechensystem für bestimmte Benutzer/innen gezielt so so betreiben, wie sie es bislang gewohnt sind, obwohl dadurch die Leistungsfähigkeit des Systems bei weitem nicht ausgeschöpft wird. Typische Beispiele sind etwa die bewusste Verlängerung von ↑Antwortzeiten oder ↑Durchlaufzeiten für die von den betreffenden Personen ausgelösten Aufträge an das Betriebssystem. Allgemein besagt dieser Aspekt, ein ↑Einplanungskriterium gerade noch gut genug für die jeweils gegebene ↑Anwendung zu erfüllen und damit ein zufriedenstellendes Maß an Leistungsfähigkeit zu gewährleisten.

protection (dt.) ↑Schutz.

protection domain (dt.) ↑Schutzdomäne.

protection ring (dt.) ↑Schutzring.

Prozedurfernaufruf Aufruf, der den aktuellen ↑Handlungsstrang verlässt und ein ↑Unterprogramm in einem gegebenenfalls anderen, dem ↑Speicherschutz unterliegenden ↑Adressraum zur Ausführung bringt. Ursprünglich ein Konzept ausschließlich für kooperiende Programme, die verteilt auf einem ↑Rechnernetz ablaufen. In abgewandelter Form jedoch auch anwendbar zur lokalen Kommunikation zwischen Handlungssträngen desselben oder verschiedener Adressräume desselben Rechners.

Prozess ↑Programm in Ausführung durch einen ↑Prozessor. Ein sich über eine gewisse Zeit erstreckender ↑Programmablauf, eine ↑Aktionsfolge. Von dynamischer Natur, bestimmt durch das Programm und den erst zur ↑Laufzeit bekannten Eingabedaten.

Prozessadressraum ↑Adressraum von einem ↑Prozess. In Abhängigkeit von der ↑Betriebsart unterliegt dieser Adressraum gegebenenfalls dem ↑Speicherschutz.

prozessbasiertes Betriebssystem (en.) ↑*process-based operating system*. Bezeichnung für ein ↑Betriebssystem, dessen grundlegendes Konzept zur Repräsentation einer autarken ↑Aktion oder ↑Aktionsfolge der ↑Prozess ist: jeder mögliche ↑Handlungsstrang in dem System benötigt eine ↑Prozessinkarnation, um stattzufinden. Die Besonderheit eines solchen Betriebssystems besteht darin, jeden dieser Stränge mit einem eigenen ↑Laufzeitstapel zu versehen. Im Gegensatz dazu steht ein ↑ereignisbasiertes Betriebssystem, das alle Handlungsstränge auf ein und demselben Laufzeitstapel ablaufen lässt.
Im Betriebssystem jedem Handlungsstrang einen eigenen Laufzeitstapel zuzuordnen, trägt wesentlich dazu bei, sowohl die ↑Einplanungslatenz als auch die ↑Einlastungslatenz für einen Prozess reduzieren zu können. Einer engen räumlichen und zeitlichen Kopplung von ↑Einplanung und ↑Einlastung steht nichts entgegen, zumindest soweit es die ↑Architektur des Betriebssystems betrifft. Kommt ein Prozess nach erfolgter Einplanung gegebenenfalls trotzdem nicht sofort zum Zuge, liegt dies in erster Linie an der Art und Weise der ↑Ablaufplanung: wenn im Betriebssystem ↑präemptive Planung stattfindet, ist damit die grundlegende Voraussetzung zur zeitnahen Einlastung eintreffender Prozesse gegeben. In zweiter Linie könnten nur noch Maßnahmen zur ↑Synchronisation einen möglichen ↑Prozesswechsel innerhalb des Betriebssystems verhindern beziehungsweise verzögern.

Prozessbevorzugung Begünstigung von einem bestimmten ↑Prozess aus einer Gruppe von Prozessen, deren jeweiliger ↑Prozesszustand sie als bereit zur ↑Einlastung auszeichnet. Ein optionales Merkmal der ↑Einplanung von Prozessen, das durch analytische oder konstruktive Maßnahmen erreicht wird. Bei analytischer Herangehensweise ist beispielsweise der ↑Rechenstoß oder die ↑Zwischenankunftszeit eines jedes Prozesses Gegenstand der Untersuchung, so dass etwa der Prozess mit der jeweils kürzesten Stoßlänge (↑SPN, ↑HRRN, ↑SRTF) oder höchsten Ankunftsrate (↑RM) bevorzugt eingelastet wird. Demgegenüber greift eine konstruktive Herangehensweise auf strukturelle Mittel zurück, um einem Prozess Vorrang gegenüber anderen Prozessen zu geben. Beispielsweise eine Vorzugsliste, um interaktive Prozesse vorrangig einzulasten (↑VRR) oder eine Hierarchie von ↑Bereitlisten, in der Prozesse zunächst oben (hohe Priorität) einsteigen, dann aber mit länger werdenden Rechenstößen schrittweise nach unten (niedrigere Priorität) durchgereicht werden (↑FB, ↑MLFQ).

Prozessdaten Informationen, analoge oder digitale ↑Daten, die mittels Sensoren einem bestimmten physikalisch-technischen Prozess entnommen und zur ↑Prozesssteuerung verwendet werden. Die Daten repräsentieren Messwerte unterschiedlichster Art, beispielsweise Temperatur, Druck, Füllstand, Spannung, Zerfall, Geschwindigkeit, Höhe, Position, aber auch Umsatz.

Prozessdatenverarbeitung ↑Steuerung oder ↑Regelung von physikalisch-technischen Prozessen durch ein unter ↑Echtzeitbetrieb laufendes ↑Rechensystem. Die zu verarbeitenden ↑Prozessdaten repräsentieren den aktuellen Zustand des physikalisch-technischen Prozesses. Als Folge der Verarbeitung dieser Daten wird der physikalisch-technische Prozess in einen neuen Zustand überführt.

Prozessdomäne Herrschaftsbereich von einem ↑Prozess, definiert durch das ihn bestimmende ↑Programm und repräsentiert durch seinen ↑Adressraum. Bei zusätzlichem ↑Speicherschutz (optional) ist es dem Prozess unmöglich, seine Domäne unkontrolliert zu verlassen und in die eines anderen Prozesses einzudringen.

Prozesseinplanung Vorgang der ↑Ablaufplanung, die auf Grundlage einer ↑Prozessinkarnation operiert und (relativen) Zeitpunkt wie auch Reihenfolge der ↑Einlastung der ↑CPU damit festlegt.

Prozessgröße (en.) ↑*process factor*. Maß der Ausdehnung von einem ↑Prozess in Raum und Zeit, quantitative Eigenschaft oder technisches Merkmal einer ↑Prozessinkarnation. Als Raumgröße gilt typischerweise die ↑Speichergrundfläche, wohingegen eine Zeitgröße die ↑Ankunftszeit, ↑Bereitzeit, ↑Startzeit, ↑Bedienzeit, ↑Bearbeitungszeit, ↑Ausführungszeit, ↑Wartezeit, ↑Endzeit oder eine ↑Frist meint.

Prozessidentifikation Nummer (auch: ↑Name) von einem ↑Prozess.

Prozessinkarnation ↑Exemplar von einem ↑Prozess. Dem ↑Adressraum des Prozesses ist (virtueller) ↑Speicher und dem Prozess ein ↑Laufzeitkontext zugewiesen.

Prozesskontrollblock Datenstruktur zur Beschreibung einer ↑Prozessinkarnation; ↑PCB. Zentrales Objekt, um für einen ↑Prozess alle von ihm belegten oder benötigten ↑Betriebsmittel, seinen ↑Prozesszustand, seinen ↑Laufzeitkontext, seine Rechte und seine Verwandtschaftsbeziehungen zu anderen Prozessen zu erfassen. Muss ein Prozess auf die Zuteilung eines Betriebsmittels oder aus anderen Gründen warten, erfasst der PCB den Grund des Wartens und enthält gegebenenfalls auch entsprechende Attribute für seine Platzierung auf einer ↑Warteschlange. Um einen Prozess erzeugen, das heißt, eine Prozessinkarnation einrichten zu können, muss das ↑Betriebssystem noch einen PCB im ↑Hauptspeicher belegen können. Für jeden existierenden Prozess gibt es eine solche Datenstruktur. Typischerweise sind diese alle in der ↑Prozesstabelle des Betriebssystems zusammengefasst.

Prozessor Verarbeitungseinheit; aktive Komponente in einem ↑Rechensystem, die eine Transformation auf ihren ↑Daten vornimmt. Die Vorschrift für diese Transformation liefert ein ↑Programm, das in Form von Hard-, Firm- oder Software vorliegt.

Prozessorauslastung Zeitanteil der produktiven Arbeit von einem ↑Prozessor. Unproduktiv ist der Prozessor typischerweise, wenn er sich im ↑Leerlauf befindet.

Prozessorkonsistenz Modell der ↑Speicherkonsistenz, nach dem jede von einem einzelnen ↑Prozessor auf den ↑Arbeitsspeicher durchgeführte Schreiboperation für andere Prozessoren in genau der Reihenfolge sichtbar ist, wie sie von ihm ausgegeben wurde. Allerdings können Schreiboperationen, die von verschiedenen Prozessoren ausgegeben wurden, von diesen Prozessoren in unterschiedlicher Reihenfolge sichtbar sein.

Prozessorregister Behältnis in der ↑CPU, um ↑Daten zu speichern. Jedes ↑Exemplar davon ist höchstens eine ↑Wortbreite groß. Es dient der eher kurzzeitigen Aufbewahrung von einem ↑Speicherwort/↑Maschinenwort oder Teilen davon, um in der CPU durch einen ↑Maschinenbefehl verarbeitet werden zu können. Die ↑Zugriffszeit auf das aufbewahrte Datum entspricht der ↑Taktfrequenz der CPU.

Prozessorstatus Bezeichnung des im ↑Programmiermodell der ↑CPU für einen ↑Prozessor definierten Zustands.

Prozessorstatuswort Bezeichnung für ein ↑Speicherwort, dessen Inhalt im ↑Unterbrechungszyklus zusammengestellt wird und das im ↑Stapelspeicher lokalisiert ist. Neben der Kopie vom ↑Statusregister umfasst dieses Wort typischerweise weitere ↑Daten zur Statusüberwachung der ↑CPU und zur Steuerung ihrer Operation am Ende von dem Unterbrechungszyklus, der zur ↑Ausnahmebehandlung im ↑Betriebssystem geführt hat.

Prozessortakt Bezeichnung für die kleinste Phase im Rythmus synchronisierter Vorgänge in einem ↑Prozessor (in Anlehnung an den Duden). Bestimmt zusammen mit der ↑Taktfrequenz die für einen ↑Befehlszyklus jeweils zu veranschlagende Zeit.

Prozesssteuerung Führung eines externen (physikalisch-technischen) Prozesses durch ein ↑Rechensystem, bei der ein fortlaufender Informationsaustausch erfolgt, um diesen Prozess zu überwachen, zu steuern und/oder zu regeln. Dazu werden die Vorgänge innerhalb des Rechensystems durch ↑Echtzeitbetrieb gelenkt.
Der externe Prozess ist nicht mit einem „internen ↑Prozess“ zu verwechseln: ersterer ist kein ↑Programm in Ausführung, gleichwohl ist ein solches Programm für seine ↑Steuerung oder ↑Regelung durch ein Rechensystem erforderlich. Bei ↑Programmablauf werden ↑Prozessdaten verarbeitet, wodurch für gewöhnlich der externe Prozess eine bewusste, gewollte und gezielte Beeinflussung erfährt (↑PDV).

Prozesstabelle Feld begrenzter Länge im ↑Betriebssystem, wobei jedes Feldelement ein ↑Prozesskontrollblock ist. Normalerweise eine statische Datenstruktur, deren Größe (d.h., Anzahl der Feldelemente) ein *Systemparameter* zur Konfigurierung des Betriebssystem bildet. Ist jeder einzelne Tabelleneintrag für eine ↑Prozessinkarnation belegt, kann ein weiterer ↑Prozess erst dann erzeugt und eingerichtet werden, wenn ein anderer Prozess terminiert, seine Prozessinkarnation gelöscht und dadurch ein Tabelleneintrag frei wurde.

Prozesswechsel ↑Aktionsfolge, die den aktuellen ↑Prozess aus- und einem ausgewählten Prozess einsetzt. Die Aktionsfolge sichert den ↑Laufzeitkontext des aktuellen Prozesses und stellt dann den Laufzeitkontext des einzusetzenden Prozesses wieder her. Darüberhinaus ist der ↑Prozesszeiger zu aktualisieren. Dieser Zeiger ist die ↑Adresse vom ↑PCB des zum Zeitpunkt auf dem ↑Prozessor stattfindenden Prozesses: für das ↑Betriebssystem hat der Prozesszeiger eine ähnliche Funktion, wie der ↑PC für die ↑CPU. Die Umschaltung zwischen Prozessen ist die zentrale Funktion der ↑Einlastung, für sie ist der ↑Umschalter zuständig.
Normalerweise erfolgt der Wechsel vom aktuellen hin zu einen anderen Prozess. Jedoch kann es auch Situationen geben, wo beide Prozesse identisch sind: beispielsweise wenn der aktuelle Prozess als ↑Leerlaufprozess agiert, also logisch blockierte, jedoch durch seine Deblockierung zwischenzeitlich wieder bereitgestellt und vom ↑Planer als nächster einzusetzender Prozess ausgewählt wurde. Ob ein Prozess zu sich selbst wechseln kann, hängt ganz von der Schnittstelle und der Implementierung von dem ↑Unterprogramm ab, das typischerweise für solch einen Wechsel aufzurufen ist. Dieses Unterprogramm benötigt dann zwei Referenzparameter, die jeweils einen ↑PCB adressieren, und es muss den PCB des aktuellen Prozesses nach erfolgter Kontextsicherung aktualisiert haben, bevor auf den PCB des einzusetzenden Prozesses zugegriffen wird. Auch wenn technisch möglich: Ein Prozesswechsel zu sich selbst bringt für den aktuellen Prozess keinen Fortschritt und bedeutet daher nur Unkosten.

Exkurs Einen Prozess aus- und einen anderen Prozess einzusetzen, also den Prozessor mit einem Prozess zu belegen (`seize`), ist ein nicht allzu komplizierter Vorgang, der in letzter Konsequenz aber immer prozessorabhängig sein wird. Nachfolgend ein einfaches Beispiel, das den prinzipiellen Ablauf eines solchen Wechsels veranschaulicht:

```
void seize(process_t *task) {                          /* dispatch process */
    process_t *last = being(ONESELF);      /* access process control block */
    state(&last->mood, CLEAR|RUNNING);       /* cancel running state, only */
    last->flow.crux = shift(task);         /* switch process, save context */
    state(&last->mood, RUNNING);             /* define running state, also */
}
```

Um einen zeitweilig ausgesetzten Prozess später wieder aufnehmen zu können, muss sein ↑Prozessorstatus gesichert werden. Dazu wird eine diesbezügliche ↑Handhabe im PCB des betreffenden Prozesses vermerkt (`flow.crux`). Den Zugriff auf den PCB des jeweils stattfindenden Prozesses ermöglicht eine Funktion (`being`), die die Adresse der dem PCB entsprechenden Datenstruktur berechnet und als Prozesszeiger zurückliefert.
In dem Beispiel bildet eine komplexe ↑Koroutine die Basis der ↑Prozessinkarnation: jedem Prozess ist damit ein eigener ↑Laufzeitstapel zugeordnet. Darüber hinaus besteht der invariant zu haltende ↑Koroutinenstatus aus jenen ↑Prozessorregistern, deren Inhalte über Funktionsaufrufe hinweg erhalten blieben (↑*non-volatile register*). Damit kann die Prozessumschaltung auf einen ↑Koroutinenwechsel wie folgt abgebildet werden:

```
couroutine_t shift(process_t *next) {                   /* switch process */
    process_t *self = being(ONESELF);      /* access process control block */
    coroutine_t *last;                        /* return value placeholder */
    unload(INNER|CDECL, 0);                 /* backup own processor state */
    last = resume(next->flow.crux);           /* switch complex coroutine */
    reload(INNER|CDECL, 0);                 /* define own processor state */
    return last;                  /* deliver handle of previous coroutine */
}
```

Prozessumschaltung (`shift`) wie auch Koroutinenwechsel (`resume`) sind hier jeweils eine Funktion, die einem wieder aufgenommenen Handlungsstrang die Handhabe des vorangegangenen Handlungsstrangs liefert: letzterer hat die Einlastung des Prozessors veranlasst. Diese Handhabe ermöglicht die spätere Fortsetzung des zeitweilig ausgesetzten Prozesses, sie identifiziert seinen gesicherten Prozessorstatus und ist daher (durch `seize`, s.o.) im PCB des betreffenden Prozesses einzutragen.
Wie das Beispiel (`shift`) zeigt, stellt jeder Prozess eigenständig den Prozessorstatus wieder her (`reload`), den er zuvor selbst gesichert hat (`unload`). Als Sicherungsspeicher dient der ↑Laufzeitstapel des Prozesses (`INNER`, wobei `CDECL` auch nur die nichtflüchtigen Register einbezieht). Der Wechsel zwischen Prozessen ist damit prozessspezifisch ausgelegt, und zwar hinsichtlich Aufbau und Struktur von dem jeweiligen Prozessorstatus. Den Prozessen gemeinsam ist lediglich das jeweils verwendete Koroutinenkonzept.
Wie weiter oben erwähnt, wird die Adresse des PCB des jeweils stattfindenden beziehungsweise eines Prozesses berechnet. Dafür ist die folgende Funktion zuständig:

```
inline process_t being(pid_t name) {      /* deliver process control block */
    return (name == ONESELF) ? gizmo() : point(name);
}
```

Funktionswert ist ein Prozesszeiger, der entweder den auf dem Prozessor gerade stattfindenden (`ONESELF`) oder einen anderen (↑Prozessidentifikation ungleich `ONESELF`) Prozess lokalisiert. Im letzteren Fall wird der PCB auf einer Liste nachgeschlagen (`point`), die typischerweise in Form der ↑Prozesstabelle implementiert ist.
Um den zum Zeitpunkt stattfindenden Prozess zu lokalisieren, gibt es für gewöhnlich zwei Herangehensweise. Der eine Ansatz verwendet pro Prozessor eine globale Variable, die vor oder nach dem Koroutinenwechsel mit dem Zeiger auf den PCB des einzusetzenden Prozesses zu aktualisieren ist. Das wäre die zusätzliche Aufgabe der oben beschriebenen Funktion `shift`, bedeutet dann aber, dass direkt vor oder nach `resume` eine Koroutine läuft, die nicht durch den dann jeweils aktuellen Prozesszeiger eindeutig identifiziert werden kann. Die beiden Einzelschritte (Zeigeraktualisierung, Koroutinenwechsel) bilden keine ↑unteilbare Anweisung, sie können eine ↑Wettlaufsituation hervorbringen, wenn ↑präemptive Planung im Hintergrund läuft. Bei diesem Ansatz muss daher präemptive Planung durch eine ↑Verdrängungssperre zeitweilig unterbunden werden.
Der andere Ansatz geht davon aus, dass der PCB eindeutig durch den ↑Stapelzeiger eines als komplexe Koroutine ausgelegten Prozesses lokalisierbar ist. Da der Wechsel zwischen dieser Art von Koroutinen immer auch auf den Austausch des Stapelzeigerinhalts hinausläuft, wird damit dann automatisch auch und atomar der Prozesszeiger umgesetzt. Dazu ist der PCB oder ein Zeiger darauf entweder am Anfang oder am Ende des ↑Stapelsegments eines jeweiligen Prozesses zu platzieren. Wenn darüber hinausgehend die Stapelsegmente aller Prozesse eine feste Zweierpotenzgröße besitzen, kann durch Maskierung des Inhalts des ↑SP die Adresse des PCB leicht bestimmt werden:

```
inline process_t gizmo() {
    return (process_t *)(((int)sp() & PMASK) + PFILL - sizeof(process_t));
}
```

Die Zweierpotenzgröße des Stapelsegments sei mit `SSIZE` bezeichnet. Der Wert von `PMASK` ist dann `-SSIZE`. Angenommen der Wert von `PFILL` sei definiert als `sizeof(process_t)`, dann liefert die Funktion die Adresse des am Anfang des Stapelsegments liegenden PCB: die CPU führt mit dem Stapelzeigerwert (`sp`) als Operand die einfache bitweise Operation *UND* aus. Ist der Wert von `PFILL` dagegen `SSIZE` gleichgesetzt, zeigt die gelieferte Adresse auf den PCB am Ende des Stapelsegments: die CPU addiert zur zwischenberechneten Basisadresse die Differenz $SSIZE - sizeof(PCB)$. Im Gegensatz zum ersten Ansatz mit der Zeigervariablen, die für gewöhnlich mit einem einzigen ↑Maschinenbefehl gelesen werden kann, sind hier nunmehr, neben dem Lesen des Stapelzeigerregisters, noch ein oder zwei weitere Maschinenbefehle notwendig. Jedoch entfällt dann bei präemptiver Planung die Notwendigkeit

zur Verdrängungssperre, die wenigstens drei bis vier Maschinenbefehle erfordert und, was schwerer wiegt, darüber hinaus gleichzeitige Prozesse zeitweilig unterbindet.

Prozesszeiger Zeiger auf den ↑Prozesskontrollblock von dem ↑Prozess, der gegenwärtig auf einem ↑Rechenkern stattfindet. Im Allgemeinen verwaltet ein ↑Betriebssystem pro Rechenkern einen solchen Zeiger.

Prozesszustand Situation, in der sich ein ↑Prozess augenblicklich befindet. Der Prozess ist bereit (*ready*) zur ↑Einlastung, wenn alle von ihm benötigten ↑Betriebsmittel bis auf den ↑Prozessor zur Verfügung stehen und letzteren aber mit anderen Prozessen teilen muss. In solch einem Fall muss der Prozess innehalten, bis der für ihn geeignete Prozessor frei wird. Ein Prozess, der stattfindet, ist laufend (*running*) auf einem Prozessor. Dieser Prozess kann zugunsten eines anderen Prozesses pausieren, dem dann der Prozessor zugeteilt wird. Der damit einhergehende ↑Prozesswechsel geschieht freiwillig oder unfreiwillig (*präemptiv*). Benötigt ein stattfindender Prozess ein weiteres Betriebsmittel, das in dem Moment jedoch nicht zur Verfügung steht, wird er blockiert (*blocked*). Dem folgt ein Prozesswechsel, wenn der Prozess sich den Prozessor mit anderen Prozessen teilen muss und wenigstens einer der anderen Prozesse in Bereitschaft steht. Steht in dem Moment kein anderer Prozess zur Einlastung bereit, wird der Prozessor untätig (↑*idle*). In der Situation wird nur ein externer Prozess dem Prozessor wieder eine Tätigkeit verschaffen können, beispielsweise durch eine ↑Unterbrechungsanforderung. Mit Zuteilung des benötigten Betriebsmittels wird ein blockierter Prozess deblockiert und wieder bereitgestellt. Diese Zustandsübergänge kontrolliert ein im ↑Betriebssystem mitlaufender (*on-line*) ↑Planer.
Darüber hinaus kann sich ein Prozess auch noch in einem ↑Schwebezustand befinden, der für gewöhnlich jedoch nur aufgrund bestimmter ↑Betriebsarten möglich ist. Typische Beispiele sind im Zusammenhang mit der ↑Umlagerung von Prozessen zu finden, aber gerade auch beim Zusammenspiel von ↑Unterbrechungsbehandlung und ↑mitlaufende Planung.

Exkurs Die Art und Weise der Zustandsverwaltung zur ↑Prozesseinplanung ist nicht unwesentlich bestimmt durch die jeweilige Betriebsart. Wenn etwa strikt ↑kooperative Planung die Grundlage bildet, dabei dann sogar Bereitstellungen von Prozessen aus der Unterbrechungsbehandlung heraus unzulässig sind und das alles nur für einen ↑Uniprozessor ausgelegt werden muss, dann bildet jeder Zustandsübergang jedes Prozesses implizit eine synchrone Operation. Dies ändert sich, wenn ↑präemptive Planung in Betracht zu ziehen ist, da dann eine Unterbrechungsanforderung den Planer ins Geschehen bringen kann: folglich ist auch im Uniprozessorfall jeder Zustandsübergang schon durch eine ↑Elementaroperation zu gewährleisten. Die Art der ↑Synchronisation der betreffenden ↑Aktionen kann hinfällig sein und ist durch eine andere Variante zu ersetzen, wenn ein ↑Multiprozessor als Basis angenommen werden muss. Daher ist es geboten, die betreffenden Operationen durch eine geeignete *funktionale Abstraktion* zu verallgemeinern.
Die technische Repräsentation des typischerweise im ↑Prozesskontrollblock enthaltenen Zustandsattributs zur Prozesseinplanung kann kompakt in Form einer *Bitleiste* ausgelegt sein. In dem Fall sind komplexe Operationen (↑*read-modify-write*) erforderlich, um die Zustandsänderungen für einen Prozess herbeizuführen. Finden diese Operationen vor dem Hintergrund von ↑Parallelverarbeitung statt, ist deren ↑Unteilbarkeit gefordert. In diesem Fall bietet sich als Alternative die Repräsentation als *Zeichenfeld* an, wodurch die Zustandsänderungen auf Lese-/Schreiboperationen zurückgeführt werden können und allein durch ↑Speicherkohärenz der korrekte Ablauf sichergestellt ist. Nachfolgend werden beide Varianten kurz vorgestellt, wobei für beide dieselben Definitionen zugrunde gelegt werden:

```
typedef enum mood {                         /* scheduling states of a process */
    READY    =0,
    RUNNING  =1,
    BLOCKED  =2,
    IDLE     =3,
    /* following are commands, only, for state management */
    CHECK    =(1 << 29),                        /* read whole state variable */
    FLUSH    =(1 << 30),                      /* define whole state variable */
    CLEAR    =(1 << 31)              /* cancel selected state-variable flag */
} mood_t;
```

Dieser ↑Datentyp definiert sowohl vier *Zustandswerte* entsprechend eingangs beschriebener Standardsituationen eines Prozesses als auch *Operationsmerker* zur Zustandsmanipulation. Mit den Operationsmerkern wird angezeigt, den aktuellen Zustand zum Prüfen (`CHECK`) zu lesen, ihn zu leeren und mit einem Zustandsmerker zu definieren (`FLUSH`) oder ihn um einen Zustandsmerker zu bereinigen (`CLEAR`). Ist bei der Operation auf dem Zustand kein Operationsmerker spezifiziert, wird der angegebene Zustandsmerker als Element in die Zustandsmenge aufgenommen. Die Verwendung dieser Merker wird weiter unten im Zusammenhang mit der funktionalen Abstraktion (`state`) zur Zustandsmanipulation deutlich.
Die vier Zustandswerte entsprechen entweder einer Bitposition in der Bitleiste oder einer Zeichenposition im Zeichenfeld, an der jeweils der Wahrheitswert zur Gültigkeit (*true*, 1) oder Ungültigkeit (*false*, 0) des betreffenden Zustands gespeichert ist. Im gegebenen Beispiel kodieren damit vier Bits beziehungsweise ↑Bytes den als *Menge* formulierten Zustand eines Prozesses, das Zeichenfeld hat damit vier Einträge (`NMOOD`) vom Typ `int8_t` und ist so platzkompatibel zum Typ `int32_t`. Zur Kodierung dieser vier Positionswerte sind zwei Bits erforderlich, aus denen sodann mit einer Maskenoperation (`MOOD`) der Index für das Zeichenfeld bestimmt wird. Nachfolgend die dementsprechenden Deklarationen:

```
#define NMOOD sizeof(long)              /* number of states or state bytes */
#define MOOD(m) (m & (NMOOD - 1))      /* relevant bits per state variable */

typedef union state {                          /* scheduling-state attribute */
    int32_t unit;                               /* read-modify-write variant */
    int8_t quad[NMOOD];                                /* load/store variant */
} state_t;
```

Zur Verallgemeinerung der verschiedenen Operationsvarianten zur Zustandsverwaltung ist der Datentyp einer Zustandsvariablen als Vereinigung (*union*) ausgelegt, wodurch die beiden hier gezeigten Komponenten (`unit`, `quad`) dieselbe ↑Adresse zugewiesen bekommen. Entsprechend dieser beiden grundlegenden, auf die Datenstruktur bezogenen Varianten lassen sich verschiedene Funktionsvarianten bilden. Nachfolgend werden zwei solcher Funktionsvarianten vorgestellt, die erste nur zum Gebrauch auf einem Uniprozessor bei strikt synchronen Abläufen und die zweite für Uni- und Multiprozessor. Zunächst die erste Variante, die sich vor allem dadurch auszeichnet, dass die Zustandsmanipulationen durch komplexe Operationen (*read-modify-write*) bewerkstelligt werden:

```
inline int32_t state(state_t *this, mood_t mood) {    /* read-modify-write */
    if (!(mood & CHECK)) {
        if (mood & CLEAR)
            this->unit &= ~(1 << MOOD(mood));
        else if (mood & FLUSH)
            this->unit = (1 << MOOD(mood);
        else
            this->unit |= (1 << MOOD(mood);
    }
    return this->unit;
}
```

Diese Funktionsvariante operiert nur auf der Bitleiste, entsprechend der Operationsmerker löscht sie ein einzelnes Zustandsbit (`CLEAR`: logisches UND), definiert die komplette Zustandsvariable (`FLUSH`: Zuweisung) oder setzt ein einzelnes Zustandsbit (sonst: logisches ODER). Jede dieser Operation samt Zuweisung muss unteilbar durchgeführt werden, sollte ein Prozess im Verlauf einer Unterbrechungsbehandlung oder von einem anderen Prozessor ausgehend bereitgestellt werden können. Dies würde eine weitere Funktionsvariante ergeben, bei der, in Abhängigkeit vom ↑Befehlssatz der ↑CPU oder geeigneten Standardfunktionen des ↑Kompilierers, die betreffenden Anweisungen (`&=`, `|=`) durch atomare *ϕ-and-fetch*-Operationen, mit ϕ gleich `andl` beziehungsweise `orl`, ersetzt sind.
Bei der gezeigten Implementierung ist zu beachten, dass der Kompilierer alle bedingten Anweisungen auflöst, da die Funktion inzeilig (`inine`) übersetzt und durch ↑Konstantenpropagation schließlich die relevante Operation bereits zur Übersetzungszeit festgelegt wird. Um also hier die Anweisung des längsten Pfads im ↑Quellmodul (`|=`) in Aktion zu bringen, wird die CPU nicht alle Fälle überprüfen müssen, sondern kann stattdessen direkt den relevanten ↑Maschinenbefehl (`orl`) ausführen. Diese Optimierungsart trifft auch auf die zweite Funktionsvariante zu, deren besonderes Merkmal jedoch darin besteht, die Zustandsmanipulationen nur mit einfachen Lese-/Schreibbefehlen durchzuführen:

```
inline int32_t state(state_t *this, mood_t mood) {          /* load/store */
    if (!(mood & CHECK)) {
        if (mood & CLEAR)
            this->quad[MOOD(mood)] = 0;
        else if (mood & FLUSH)
            this->unit = (1 << (MOOD(mood) * 8));
        else
            this->quad[MOOD(mood)] = 1;
    }
    return this->unit;
}
```

Das Zustandsattribut (`mood`) gibt hierbei den Indexwert für das Zeichenfeld, durch den der zu manipulierende Zustandsmerker (`quad[...]`) selektiert wird. Bei der `FLUSH`-Anweisung ist zudem die ↑Bytereihenfolge der CPU zu berücksichtigen und sicherzustellen, dass bei der Zuweisung an die komplette Zustandsvariable tatsächlich auch die dem Zeichenfeldeintrag entsprechende Byteposition definiert wird. Für ↑x86 (*little endian*) ist die hier gezeigte „Normierung“ durch Multiplikation des Indexwerts (d.h., Zustandsattribut `mood`) mit der Bitanzahl (`8`) pro Byte stimmig: es wird das jeweils i-te Byte in der Bitleiste (`unit`) oder dem Zeichenfeld (`quad`) auf den Wert Eins gesetzt.
Voraussetzung dafür, eben nicht auf Komplexbefehlen entsprechenden Anweisungen zurückgreifen zu müssen, die gegebenenfalls jeweils in atomarer Ausfertigung vorzuliegen haben, ist die Darstellung der Zustandsmenge als Zeichenfeld. Unter der Annahme von Speicherkohärenz ist damit insbesondere bei der möglichen asynchronen Bereitstellung eines Prozesses für die korrekten Zustandsübergänge gesorgt. Diese Variante empfiehlt sich bei einem ↑RISC, dessen Befehlssatz gemeinhin überhaupt keine komplexen arithmetisch-logischen Operationen anbietet, die direkt im ↑Hauptspeicher wirken. Aber auch bei einem ↑CISC (wie x86), der solche Operationen im Befehlssatz aufweist, kann die zweite hier diskutierte Funktionsvariante von Vorteil sein, da einfache Zuweisungsoperationen für gewöhnlich schneller ablaufen als komplexe Rechenoperationen.

Pseudobefehl Anweisung an einen ↑Assemblierer. Beispielsweise zur Angabe des Programmabschnitts (↑Text, ↑Daten, ↑BSS), auf die sich die nachfolgenden Anweisungen beziehen, um den ↑Adresszähler des aktuellen Segments auf eine bestimmte Grenze auszurichten, einem ↑Symbol einen Wert zuzuweisen oder Informationen für den ↑Binder aufzubereiten.

Pseudodatei Bezeichnung für eine ↑Datei, deren eigentliche Bedeutung lediglich nachgeahmt ist, um im ↑Betriebssystem verfügbare ↑Daten oder ↑Betriebsmittel auch einem ↑Prozess

im ↑Maschinenprogramm kontrolliert zugänglich machen zu können. Im Falle von ↑UNIX-artigen Betriebssystemen ist das beispielsweise die ↑Gerätedatei oder `proc(5)`.

PSW Abkürzung für (en.) ↑*processor status word* oder ↑*program status word*.

Puffer Ursprünglich die Bezeichnung für eine federnde Vorrichtung an Vorder- und Rückseite eines Schienenfahrzeugs zum Auffangen von Stößen (Duden). Im übertragenen Sinne ein ↑wiederverwendbares Betriebsmittel (Vorrichtung) zum Ausgleich der durch die unterschiedlichen Geschwindigkeiten zweier oder mehrerer ↑Prozesse (Schienenfahrzeuge) hervorgerufenen Wirkungskräfte (Auffangen von Stößen).
Entgegen der landläufigen Nutzung dieses Mittels zur Zwischenspeicherung von ↑Daten, steht aber eben dieser Aspekt der Aufbewahrung von Informationen hier gerade nicht im Vordergrund: der ↑Speicherbereich ist lediglich Mittel zum Zweck, nämlich eine sich auf einen bestimmten Prozess auswirkende ↑Aktionsfolge eines anderen Prozesses derart zu puffern, dass beide keine oder nur geringe Verzögerung erfahren. Typischerweise kooperieren die betreffenden Prozesse in der Durchführung einer Berechnung. Dabei markieren bestimmte Zwischenschritte jeweils ein ↑Ereignis, das eine kausale Abhängigkeit zwischen den Prozessen definiert. Jedes dieser Ereignisse ist von dem einen Prozess (Produzent) zu dem anderen Prozess (Konsument) zu kommunizieren und wird dazu in Form bestimmter Daten (↑Signal, ↑Nachricht: ↑konsumierbares Betriebsmittel) kodiert.

Puffern Konzept einer Methode, die Auswirkung eines Vorgangs auf einen Gegenstand oder anderen Vorgang mildern, abschwächen zu können (in Anlehnung an den Duden). Ursprüngliche Bedeutung ist die Entkopplung von (externen/internen) ↑Prozessen unterschiedlicher Geschwindigkeiten. Diese Entkopplung wird erreicht, indem der eine Prozess ↑Daten, die ein bestimmtes ↑Ereignis repräsentieren, zur späteren Verarbeitung durch den anderen Prozess zwischenspeichert. Der dafür benötigte ↑Speicherbereich wird als ↑Puffer bezeichnet. Zu beachten ist jedoch, dass die Speicherung selbst nicht im Vordergrund steht und nur als ein Mittel zum Zweck für den Ausgleich verschiedener Wirkungskräfte anzusehen ist.

punched paper tape (dt.) ↑Lochstreifen.

Quellmodul ↑Datei mit einem ↑Programm als Eingabe für einen ↑Übersetzer. Gleichfalls aber auch Ausgabedatei eines Instruments zur Programmerzeugung, das heißt, eines Editors, Übersetzers (insb. ein ↑Kompilierer) oder Generators.

Querschnittsbelang (en.) ↑*cross-cutting concern*. Bezeichnung für einen bedeutsamen Aspekt in der Software, der erst wie bei einem in Querrichtung durch einen Körper geführten Schnitt sichtbar werden würde (in Anlehnung an den Duden). Ein in einem ↑Programm beschriebenes charakteristisches Merkmal, das nicht an einer Stelle der Software konzentriert in Erscheinung tritt, sondern mehrfach in identischer oder leicht abgewandelter Form vorhanden ist. Dieses Merkmal kann mangels geeigneter Modularisierungskonzepte in der gewählten Programmiersprache nicht einfach als einzelnes Modul separat dargestellt werden. Meist bringt ein solches Merkmal eine bestimmte nichtfunktionale Eigenschaft zum Ausdruck. Typisches Beispiel dafür sind Programmstellen, an denen unter bestimmten Umständen etwa für die zusätzliche ↑Synchronisierung von ↑Prozessen zu sorgen ist.

queue (dt.) ↑Schlange.

race condition (dt.) ↑Wettlaufsituation.

race hazard (dt.) Laufgefahr, ↑Wettlaufsituation.

RAM Abkürzung für (en.) *random access memory*, (dt.) ↑Direktzugriffsspeicher.

re-entrance (dt.) ↑Wiedereintritt.

re-entrant (dt.) ablaufinvariant, eintrittsinvariant, wiedereintrittsfähig. Eigenschaft zur ↑Eintrittsinvarianz eines ↑Programms.

read-modify-write Bezeichnung für einen aus drei Teilschritten bestehenden Zyklus bei der Ausführung von einem komplexen ↑Maschinenbefehl: erstens, ein ↑Speicherwort auslesen und in den ↑Prozessor laden; zweitens, den geladenen Wert im Prozessor verändern; drittens, den geänderten Wert in dasselbe Speicherwort zurückschreiben. Der Maschinenbefehl beschreibt eine ↑Aktionsfolge, die zwar eine logisch zusammenhängende Einheit bildet, jedoch keine ↑atomare Operation darstellt. Das bedeutet, nach jedem Teilschritt kann ein anderer Prozessor denselben Maschinenbefehl angewendet auf demselben Maschinenwort ausführen. Folge davon ist eine möglicherweise inkonsistente Berechnung.

ready list (dt.) ↑Bereitliste.

real time (dt.) ↑Echtzeit.

real-time mode (dt.) ↑Echtzeitbetrieb.

reale Adresse ↑Adresse, die dem ↑Datenbus den Zugriff auf ein bestimmtes ↑Speicherwort im ↑Hauptspeicher oder auf ein spezielles ↑Ein-/Ausgaberegister von einem ↑Peripheriegerät (↑*memory-mapped I/O*) ermöglicht.

realer Adressraum Bezeichnung für den ↑Adressraum, der durch die reale Maschine (Hardware) definiert ist. Dieser Adressraum ist nicht zwingend linear aufgebaut, auch wenn die ↑CPU entsprechend ihrer ↑Adressbreite und dem daraus resultierenden ↑Adressbereich die Generierung einer beliebigen ↑Adresse darin zulässt. Adressbereiche in diesem Adressraum können undefiniert sein, Lücken bilden. Konsequenz daraus ist, dass nicht jede von der CPU applizierte Adresse auch gültig ist, das heißt, einen Adressaten (↑Speicherwort, ↑Peripheriegerät) hat. Die Verwendung einer solchen ungültigen Adresse in einem ↑Prozess ist ein schwerwiegender Fehler (↑*bus error*). Damit ein Prozess in einem solchen Adressraum nicht scheitert, muss sein ↑Programm auf die Adressbereiche mit gültigen Adressen abgebildet worden sein. Die Abbildung ist spätestens zur ↑Bindezeit zu leisten. Verfügt die CPU über eine ↑MPU, steckt das ↑Betriebssystem dazu obere und untere Grenzen für das Programm ab, so dass der zugehörige Prozess nur für ihn gültige Adressen generieren und so aus sein ↑Text-, ↑Daten- und ↑Stapelsegment nicht ausbrechen kann. Gleichwohl muss jedes einzelne ↑Segment komplett und zusammenhängend im ↑Hauptspeicher vorliegen. Darüberhinaus ist die durch die ↑Ladeadresse vorgegebene Lokalität jedes dieser Segmente zur ↑Laufzeit des Programms unveränderlich.

Rechenanlage Gesamtheit der ein ↑Rechensystem konstituierenden Hardware (↑Rechner und ↑Peripherie). Eine durch (wenigstens) ein ↑Programm gesteuerte, ↑Daten verarbeitende Anlage, deren Ausmaß und ↑Betriebsart durch den jeweiligen Einsatzzweck bestimmt ist.

Rechenkern Prozessorkern einer ↑CPU, auf dem ein ↑Prozess stattfinden kann. Eine ↑CPU kann mehrere solcher Kerne enthalten und so Prozesse gleichzeitig stattfinden lassen.

Rechenstoß Häufung von Aktivität in Bezug auf die ↑CPU (↑CPU *burst*). Die CPU führt einen ↑Maschinenbefehl nach dem anderen aus, bestimmt durch das ↑Programm: sie verarbeitet einen Stoß von Maschinenbefehlen bezogen auf einen bestimmten ↑Prozess. Der betreffende Prozess ist laufend (↑Prozesszustand). Für Anfang und Ende von einem solchen „Ausführungsbündel“ gibt es eine physische und eine logische Sicht. Die physische (einfachere) Sicht nimmt die Position der realen Maschine ein: der Stoß beginnt mit jedem Neustart (*reset*) oder Erwachen aus dem ↑Schlafzustand der CPU und endet mit jedem Eintritt in diesen. Demgegenüber gestaltet sich die logische Sicht, die eine ↑abstrakte Maschine in Form von dem ↑Betriebssystem als Grundlage nimmt, etwas schwieriger. Letztlich führt hier die Differenzierung zwischen Prozess und ↑Prozessinkarnation, beziehungsweise der jeweilige ↑Arbeitsmodus der CPU, genau genommen zu verschiedenen Arten und damit auch Start-

und Endpunkten eines solchen Stoßes (nachfolgend im Uhrzeigersinn verdeutlicht).
Demzufolge nimmt ein Prozess seinen Stoß beim physischen ↑Prozesswechsel auf, also wenn von einer ↑Prozessinkarnation zu einer anderen umgeschaltet wird (00:00 Uhr). Im Moment dieser Umschaltung (↑*dispatching*) handelt die CPU für das Betriebssystem (↑*privileged mode* bzw. ↑*system mode*). Dieser Stoß setzt sich fort bis zum Verlassen des Betriebssystems und zur erstmaligen oder wiederholten Aufnahme der Ausführung von dem ↑Maschinenprogramm, für das die Prozessinkarnation eingerichtet wurde. Im Moment von dem damit verbundenen ↑Moduswechsel wird die CPU ihre Handlung für das Betriebssystem beenden und für das Maschinenprogramm (↑*unprivileged mode* bzw. ↑*user mode*) beginnen (06:00 Uhr). Die Prozessinkarnation vollzieht mit dieser Umschaltung letztlich einen logischen Prozesswechsel, da durch Aufnahme der Ausführung des Maschinenprogramms die Entwicklung eines neuen Stoßes von Maschinenbefehlen eines anderen Programms und damit auch in einem anderen ↑Adressraum beginnt. Diese Entwicklung setzt sich bis zur Unterbrechung des Prozesses fort, nämlich wenn das Maschinenprogramm durch eine ↑synchrone Ausnahme oder ↑asynchrone Ausnahme verlassen und das Betriebssystem wieder betreten wird. Im Moment des damit verbundenen erneuten Moduswechsels wird die CPU ihre Handlung für das Maschinenprogramm beenden und für das Betriebssystem beginnen (18:00 Uhr). Erneut kommt es zum logischen Prozesswechsel, da durch Aktivierung des Betriebssystems die Entwicklung eines neuen Stoßes von Maschinenbefehlen eines anderen Programms in einem anderen Adressraum beginnt. Dieser neue Stoß entwickelt sich weiter bis zum (a) Verlassen des Betriebssystems und zur Rückkehr zum Maschinenprogamm (06:00) oder (b) physischen Prozesswechsel, nämlich wenn die zugehörige Prozessinkarnation weggeschaltet wird (24:00 Uhr).
Dieser Kreislauf zeigt einen langen Stoß für die Prozessinkarnation, der jedoch aus wenigstens drei kleineren Stößen für die logisch verschiedenen Prozesse dieser Inkarnation zusammengesetzt ist: letztere bilden Stoßabschnitte, die verschiedenen Phasen einer Prozessinkarnation entsprechen. Bestimmte Abschnitte (18:00–06:00 ohne und 00:00–06:00 sowie 18:00–24:00 mit Wechsel der Prozessinkarnation) sind dem Betriebssystem, ein anderer Abschnitt (06:00–18:00) ist ein- oder mehrfach dem Maschinenprogramm zuzurechnen. Dies gilt grundsätzlich für jede Prozessinkarnation, die durch ein Betriebssystem zur Ausführung eines Maschinenprogramms bereitgestellt wird. Wissen über die Länge der jeweiligen Stöße in zeitlicher Hinsicht gibt Aufschluss über die von einer Prozessinkarnation selbst verrichteten und anderen übertragenen Arbeit, letzteres insbesondere in Bezug auf den ↑Ein-/Ausgabestoß, den der Prozess während seiner Verweildauer im Betriebssystem ein- oder mehrfach auslösen kann. Die ↑Daten dienen allgemein Abrechnungszwecken (↑*accounting*) und liefern Hinweise über den Ausnutzungsgrad von einem ↑Rechensystem (z.B. ↑UNIX: `getrusage(2)`).

Rechensystem Einheit aus technischen Anlagen, Bauelementen und Programmen zur Verarbeitung von ↑Daten und Durchführung von Berechnungen. Zentrales Konzept ist der ↑Rechner, der allein oder im Verbund vernetzt mit anderen (gleichen/ungleichen) Einheiten seiner Art eine bestimmte Funktion für einen bestimmten Anwendungsfall ausübt. Dieser Anwendungsfall kann in besonderem Maße auf einen ganz bestimmten Zusammenhang ausgerichtet sein oder verschiedenste Bereiche umfassen, das heißt, einerseits ein Spezialsystem (*special-purpose system*) oder anderseits ein Universalsystem (*general-purpose system*) bedingen.

Rechenzeit Zeit, die für die Operationen eines ↑Rechensystems zur Bewältigung einer ↑Aufgabe benötigt wird (in Anlehnung an den Duden). Die zur ↑Laufzeit von einem ↑Programm beanspruchte Zeit zur Durchführung von Berechnungen. Genau genommen die Zeit, die durch einen ↑Rechenstoß verbraucht wird.

Rechner Programmgesteuerte, elektronische Rechenanlage (↑*computer*).

Rechnerarchitektur ↑Architektur von einem ↑Rechner oder ↑Rechensystem. Diese ist bestimmt durch ein ↑Operationsprinzip für die Hardware und einer Struktur gegeben durch Art/Anzahl der ↑Betriebsmittel (Hardware) und die sie verbindenden Kommunikationseinrichtungen, einschließlich der dafür definierten Kommunikations- und Kooperationsregeln.

Rechnernetz Zusammenschluss mehrerer voneinander unabhängiger ↑Rechner über ein ↑Netzwerk, der die Kommunikation der zusammengeschlossenen Einheiten ermöglicht. Dabei folgt die Kommunikation zwischen den verschiedenen Rechnern einem bestimmten Protokoll, das für gewöhnlich wenigstens paarweise einheitlich ist. Im gesamten Netz können mehrere Protokolle möglich und zugleich aktiv sein (z.B. ↑UDP, ↑TCP und ↑IP). Die Struktur der Verbindungen (*Topologie*) der Rechner kann sehr unterschiedlich ausgeprägt sein. Typische Formen sind Linie, Ring, Stern, Baum, Bus, (teil-) vermascht und vollvermascht (jeder Rechner mit jedem anderen). Auch hier können im Netz mehrere Formen zugleich ausgeprägt sein.

reentrancy (dt.) ↑Eintrittsinvarianz.

reference (dt.) ↑Referenz.

reference bit (dt.) ↑Referenzbit.

reference string (dt.) ↑Referenzfolge.

Referenz ↑Speicherstelle, auf die verwiesen wird, weil sie Auskunft über etwas geben kann (in Anlehnung an den Duden). Der Bezug auf einen bestimmten, im ↑Speicher vorrätigen Bestand von ↑Text oder ↑Daten (dem Speicherinhalt an der Speicherstellle).

Referenzbit ↑Speichermarke im ↑Seitendeskriptor, die von der ↑MMU beim Zugriff (ausführen, lesen, schreiben) auf die betreffende ↑Seite gesetzt wird.

Referenzfolge Reihe von zeitlich aufeinanderfolgenden, von einem ↑Prozess generierte Referenzen auf den ↑Arbeitsspeicher. Der Verlauf dieser Reihe ist bestimmt durch die Struktur von dem ↑Programm, das den Prozess festlegt und der ausgeführten Funktion in Abhängigkeit von den Eingabedaten. Jede Referenz entspricht einer ↑Adresse.

Regelkreis Wirkungsablauf, der die Beeinflussung einer physikalischen Größe (Regelgröße) in einem bestimmten technischen Prozess erreicht und dabei Abweichungen von einem vorgegebenen Sollwert (Führungsgröße) kontinuierlich und für gewöhnlich in Schritten (Stellgröße) entgegenwirkt; ein sich selbst regulierendes geschlossenes System (Duden). Dabei bildet ein technischer Prozess die „Gesamtheit von aufeinander einwirkenden Vorgängen in einem System, durch die Materie, Energie und Information umgeformt, transportiert und gespeichert wird“ (DIN IEC 60050-351). Typische Beispiele für solche Prozesse finden sich in der Fertigungs-, Verfahrens- oder Mess- und Prüftechnik, bei der technische Anlagen und Maschinen (d.h., technische Arbeitsmittel) durch ein ↑Rechensystem bedient, gesteuert und überwacht werden.
Ein solcher Kreislauf zur ↑Regelung des technischen Prozesses muss stabil sein, soll ein gutes Führungs- und Störverhalten aufweisen und soll robust sein. Der Kreislauf ist stabil, wenn die Regelung keine Dauerschwingungen oder aufklingende Schwingungen erzeugt. Das Führungsverhalten beschreibt die Auswirkung von Aufschaltungen der (positiven oder negativen) Führungsgröße auf die Regelgröße. Dem wirken für gewöhnlich Störgrößen aus der Umgebung der Regelstrecke entgegen (Störverhalten), die so auszuregeln sind, dass die Regelgröße wieder den Wert der Führungsgröße annimmt. Der Kreislauf ist robust, wenn der durch Dauerbelastung und einhergehender Materialermüdung hervorgerufene Einfluss von Parameteränderungen auf Regler und Regelstrecke in vorgegebenen Toleranzbereichen bleibt.

Regelung Vorgang in einem ↑Regelkreis, bei dem durch ständige Kontrolle und Korrektur eine physikalische, technische oder ähnliche Größe auf einem konstanten Wert gehalten wird (Duden). Dazu ist der Istwert der zu beeinflussenden Größe, als *Regelgröße* bezeichnet, stetig zu messen und mit einem Sollwert, die sogenannte *Führungsgröße*, zu vergleichen. Die Messung erfolgt direkt in oder an der *Regelstrecke*, das heißt, dem die zu beeinflussende Größe enthaltenden Teil des Regelkreises. Beispiele für eine Regelstrecke sind Stoffströme (d.h., Gas- oder Flüssigkeitsströme, Papier-, Kunststoff- oder Metallbahnen), Wärmeerzeugungs- und

Kühlungsanlagen, Wärmetauscher, chemische Reaktoren, Kernreaktoren, Fahr- oder Flugzeuge, elektrische Maschinen und nachrichtentechnische Geräte (d.h., Rundfunkgeräte) — aber auch eine ↑CPU: nämlich um Überhitzung zu vermeiden und Durchschmelzen vorzubeugen (↑*dark silicon*). Abweichungen zwischen Regel- und Führungsgröße ergeben jeweils eine bestimmte *Regeldifferenz*, aus der ein *Regler* eine *Stellgröße* berechnet. Letztere wirkt stellend (erhöhen, verringern) auf die Regelgröße ein, und zwar derart, dass trotz (unbekannter) *Störgrößen* die Abweichung verringert und schließlich minimal gehalten wird Eine solche Rückkopplung fehlt bei der ↑Steuerung. Der gesamte Vorgang ist zyklisch und unterliegt Zeitvorgaben, die durch die physikalischen Eigenschaften der Regelstrecke definiert sind und eine bestimmte ↑Echtzeitbedingung manifestieren.

Register Behältnis in einem ↑Prozessor oder in einem ↑Peripheriegerät, nämlich als ↑Speicher oder Übertragungsmedium von ↑Daten: daher auch differenziert in ↑Prozessorregister und ↑Ein-/Ausgaberegister. Ohne weitere Qualifizierung ist erstere Art implizit gemeint.

Registersatz Menge aller ↑Prozessorregister. Typischerweise differenziert nach folgenden Arten: Datenregister, zur Speicherung von Operanden und Rechenergebnissen; Adressregister, zur Adressierung von Operanden und Maschinenbefehlen; Spezialregister, zur Betriebsstandanzeige (↑Statusregister) oder Adressierung spezieller Programmsegmente. Neben den dem ↑Programmiermodell zugerechneten Registern umfasst die Menge auch interne Register, auf die von einem ↑Programm heraus nicht zugegriffen werden kann.

Registerspeicher Bezeichnung von ↑Speicher in der ↑CPU mit sehr geringer Kapazität (2 bis 1024 ↑Prozessorregister) und extrem kurzer ↑Zugriffszeit für die Zwischenspeicherung von ↑Daten. Die Gesamtheit aller Prozessorregister bildet den ↑Registersatz.

Rekursion Bezeichnung für den Vorgang in einem ↑Programmablauf, wenn ein ↑Unterprogramm in seiner Definition selbst nochmals aufgerufen wird. Dieser rekursive Aufruf kann direkt in dem betreffenden Unterprogramm oder indirekt in einem anderen Unterprogramm kodiert sein, wobei letzteres auf einem Aufrufpfad heraus aus ersterem Unterprogramm erreichbar ist. Indirekt rekursive Aufrufe sind *wechselseitig rekursiv*, da die betreffenden Unterprogramme durch (direkte oder indirekte) wechselseitige Verwendung voneinander definiert sind. Ein wichtiger Aspekt dabei ist die Existenz einer Abbruchbedingung für die Entfaltung der rekursiven Definition eines Unterprogramms. Diese Bedingung muss sicherstellen, dass ein rekursiver Aufruf des Unterprogramms (spätestens zur ↑Laufzeit) in endlich vielen Schritten aufgelöst werden kann.
In ↑Betriebssystemen ist diese Art des Aufrufs von Unterprogrammen eher selten anzufinden und wenn dort überhaupt, dann nur mit größter Vorsicht zu verwenden. Grund dafür ist die für gewöhnlich starke und gegebenenfalls durch statische Programmanalyse auch nicht mehr bestimmbare maximale Ausdehnung des ↑Laufzeitstapels eines ↑Prozesses, wenn ein solcher Aufruf durch den ↑Kompilierer nicht in eine herkömmliche Programmschleife umgewandelt werden kann. Die Ausdehnung des Laufzeitstapels über seine Grenze hinweg löst für gewöhnlich ↑Panik aus. Beispiele für diese Aufrufart finden sich unter anderem bei der ↑Unterbrechungsbehandlung (insb. ↑FLIH).

release time (dt.) ↑Bereitzeit.

relocating loader (dt.) ↑verschiebender Lader.

relocation (dt.) ↑Relokation.

Relokation Zuweisung einer ↑Ladeadresse an eine Stelle im ↑Maschinenprogramm, an der eine ↑absolute Adresse verwendet wird. Die Stellen sind in der von einem ↑Assemblierer pro ↑Objektmodul anteilig erzeugten und von einem ↑Binder für das ↑Lademodul aus den Anteilen zusammengestellten ↑Relokationstabelle verzeichnet. Für jede dieser Stelle wird die dort

benötigte effektive absolute Adresse bestimmt. Grundlage bildet die für das Maschinenprogramm geltende ↑Relokationskonstante, um die der Wert von dem an einer solchen Stelle verwendetem ↑Symbol verschoben wird: $Adresse_{abs} = Wert(Symbol) + Relokationskonstante$. Der Wert des Symbols wird der ↑Symboltabelle entnommen, die ebenfalls anteilig vom Assemblierer pro Objektmodul erzeugt und vom Binder für das Lademodul entsprechend zusammengestellt wurde. Dabei wird der Binder, soweit möglich, die Werte der für Adressen stehenden Symbole bereits derart aktualisieren, dass diese jeweils relativ zum logischen Basiswert 0 von dem jeweiligen ↑Text- und ↑Datensegment des Maschinenprogramms ausgerichtet sind: die aktualisierten Werte bilden damit Adressen innerhalb ihres jeweiligen Segments, zu denen dann die Relokationskonstante addiert wird, um die Ladeadresse zu erhalten.

Relokationskonstante Festwert für die Justierung von ↑Text oder ↑Daten beim ↑Laden (↑*relocation*). Der Wert dieser Konstante hängt vor allem davon ab, in welcher Art von ↑Adressraum ein ↑Maschinenprogramm ablaufen soll. Ist durch die ↑Betriebsart ein ↑realer Adressraum vorgegeben, definiert die Konstante die ↑Adresse im ↑Hauptspeicher, an der das Programm geladen werden kann. Wird dagegen ein ↑logischer Adressraum oder ↑virtueller Adressraum durch die Betriebsart vorgegeben, bestimmt das ↑Betriebssystem die ↑logische Adresse beziehungsweise ↑virtuelle Adresse, an der das Programm liegen soll.

Relokationstabelle Liste von Zeigern auf die zu ändernden Stellen im ↑Objektkode, um ein ↑Maschinenprogramm vom ↑Binder zusammenzufügen, zu binden, oder durch den ↑Lader nachträglich noch verschieden zu können. Jede dieser Stelle gibt an, wo eine ↑Ladeadresse einzutragen ist.

remote operation (dt.) ↑abgesetzter Betrieb.

replacement policy (dt.) ↑Ersetzungsstrategie.

rerun (dt.) ↑Wiederholungslauf.

rerun bit Steuerungsbit im ↑Prozessorstatuswort, Schaltvariable als Ausprägung eines booleschen Datentyps: im Falle von `true` wird die ↑CPU angewiesen, einen ↑Wiederholungslauf durchzuführen; anderenfalls (`false`) ruft die CPU ganz normal den nächsten ↑Maschinenbefehl in Folge ab und beginnt mit seiner Ausführung. Typischerweise deutet die CPU dieses Bit im Moment der Wiederaufnahme von einem zuvor unterbrochenen ↑Prozess, nämlich am Ende vom zugehörigen ↑Unterbrechungszyklus.

rescheduling (dt.) ↑Umplanung.

resident monitor (dt.) ↑residentes Steuerprogramm.

resident set (dt.) ↑Residenzmenge.

residentes Steuerprogramm Bezeichnung für ein ↑Programm, das den Ablauf anderer Programme organisiert und überwacht (Duden). Der im ↑Hauptspeicher residente, ein „embryonales ↑Betriebssystem“ bildende Teil einer ↑Systemsoftware. Dieses Programm wird für gewöhnlich durch ein ↑Urladeprogramm selbst in den Hauptspeicher gebracht, wenn der betreffende ↑Rechner hochfährt, und verbleibt dort solange, bis dieser Rechner herunterfährt. Die Eigenschaft, resident zu sein, besteht daher lediglich im besonderen Verhältnis zum ↑Maschinenprogramm, das nämlich eins nach dem anderen (mit für gewöhnlich nicht immer derselben Funktion) das ↑Rechensystem durchläuft und damit die eigentliche transiente Komponente im System darstellt.
Die Konsequenz aus dieser Eigenschaft ist, dass sowohl für eine logische als auch physische Entkopplung von Maschinen- und Steuerprogramm gesorgt werden muss: um das Maschinenprogramm nachträglich laden zu können, ist es logisch vom Steuerprogramm zu entkoppeln; um das Steuerprogramm für die gesamte Betriebsdauer unbeschadet zu belassen, ist es physisch vom Maschinenprogramm zu entkoppeln. Beide Arten stellen für sich eigenständige

Programme dar, die jeweils mit wohldefinierten Funktionen versehen sind.
Logische Entkopplung meint, dass das ↑Binden dieser Programm weder räumlich noch zeitlich zusammenhängend geschieht. Dies bedeutet insbesondere, dass dem ↑Binder die ↑absolute Adresse von einem im Maschinenprogramm benutzten ↑Unterprogramm des Steuerprogramms für gewöhnlich nicht bekannt ist: die ↑symbolische Adresse dieses Unterprogramms kann nicht aufgelöst werden, womit die für ein ausführbares Maschinenprogramm erforderliche ↑Bindung nicht möglich ist. Um zwar logisch entkoppelt dennoch Funktionen des Steuerprogramms in Anspruch nehmen zu können, werden die Adressen all dieser „exportierten Funktionen“ in eine Sprungtabelle eingetragen. Dies geschieht, wenn das Steuerprogramm gebunden wird oder es sich initialisiert. Die Adresse dieser Tabelle ist dem Binder bekannt oder er hinterlässt beim Binden den Hinweis, an welcher Adresse sie im Hauptspeicher vom Steuerprogramm zu platzieren ist. Darüber hinaus erhält jede dieser exportierten Funktion eine Nummer, die den Eintrag der zugehörigen Funktionsadresse in der Tabelle identifiziert. Der Aufruf einer Funktion des Steuerprogramms erfolgt sodann indirekt über die Sprungtabelle. Dieses Verfahren entspricht in den wesentlichen Grundzügen der programmiersprachlichen Auslegung von einem ↑Systemaufruf.
Demgegenüber meint physische Entkopplung vor allem ↑Speicherschutz für das Steuerprogramm (↑*fence register*, ↑*bounds register* oder ↑*base/limit register*), um eben dafür zu sorgen, dass fehlerhafte Maschinenprogramme schlimmstenfalls immer nur lokale Auswirkungen auf sich selbst, das heißt, auf ↑Text und ↑Daten in ihrem jeweils eigenen ↑Adressraum haben. Jedes Maschinenprogramm kommt strikt räumlich isoliert vom Steuerprogramm zur Ausführung. Damit ist aber ein Aufruf der Steuerprogrammfunktionen über die Sprungtabelle nicht mehr in der Form möglich, wie die logische Entkopplung des Maschinenprogramms es vorsah: die Tabelle liegt logisch genau zwischen zwei Programmen, die nunmehr aber durch Speicherschutz physisch voneinander getrennt sind. Da Hardware die Grenze zwischen Maschinen- und Steuerprogramm sichert, ist ein spezieller ↑Maschinenbefehl für den Aufruf von Steuerfunktionen aus dem Maschinenprogramm heraus erforderlich. Gleiches gilt für die Rückkehr zum Maschinenprogramm, heraus aus dem Steuerprogramm. Die Technik dafür liefert die ↑partielle Interpretation. Dabei wandert die Sprungtabelle in den Adressraum des Steuerprogramms, sie wird von einem ↑Systemaufrufzuteiler genutzt, um die mittels des speziellen Maschinenbefehls angeforderte Steuerprogrammfunktion aufzurufen. Der Aufruf im Maschinenprogramm verläuft nunmehr über einen ↑Systemaufrufstumpf, der die gegebene ↑Aufrufkonvention entsprechend der von ↑CPU und Steuerprogramm vorgegebenen Merkmale abbildet. Damit ist der Systemaufruf wie in seiner für gewöhnlich technischen Umsetzung realisiert.

Residenzmenge Menge des im ↑Hauptspeicher zu einem Zeitpunkt für einen ↑Prozess vorgehaltenen Text- und Datenbestands (↑*resident set*), wobei zur Aufbewahrung des Gesamtbestands ↑seitennummerierter virtueller Speicher als Grundlage genommen wird. Die Bestandsgröße in beiden Fällen ist ganzzahliges Vielfaches einer ↑Seite. Zu beachten ist der Unterschied zur ↑Arbeitsmenge eines Prozesses, deren Seiten zwar im Hauptspeicher residieren, die jedoch nur einen Teil aller residenten Seiten des Prozesses ausmachen: die Arbeitsmenge eines Prozesses ist Teilmenge der Residenzmenge des Prozesses. Darüberhinaus unterliegt jede Seite zuletzt genannter Sorte der Möglichkeit der individuellen ↑Verdrängung aus ihrem ↑Seitenrahmen, solange sie (nämlich als ↑Betriebsseite) nicht auch der Arbeitsmenge des Prozesses angehört: eine Residenzmengenseite kann sehr wohl einzeln verdrängt werden, wohingegen eine Arbeitsmengenseite immer nur im Verbund derselben Arbeitsmenge verdrängt, das heißt, aus- und eingelagert wird — es sei denn, die Mächtigkeit der Arbeitsmenge ist 1.

resource (dt.) ↑Betriebsmittel, ↑Ressource.

resource management (dt.) ↑Betriebsmittelverwaltung.

response time (dt.) ↑Antwortzeit.

Ressource (en.) ↑*resource*. Lehnwort aus dem Französischen: (dt.) Auskommen, Mittel, Hilfsmittel. Bezeichnung für ein einzelnes durch ein ↑Betriebssystem zur Verfügung gestelltes Mittel, um einen ↑Prozess betreiben zu können (↑Betriebsmittel). Grundsätzlich benötigt jeder Prozess (in einem ↑Rechensystem) jeweils wenigstens ein ↑Exemplar von drei Hardwareklassen solcher Mittel:

1. den ↑Hauptspeicher, in dem das ↑Programm für den Prozess vorliegen muss,
2. einen ↑Prozessor, um den Prozess überhaupt stattfinden lassen zu können und
3. die ↑Peripherie, damit der Prozess mit dem „Außenwelt“ kommunizieren kann.

Für gewöhnlich kommen für jedes dieser Exemplare weitere Mittel (Software) hinzu, die in der von einem Prozess zu leistenden Arbeit begründet sind. Für die Klasse ↑wiederverwendbares Betriebsmittel sind etwa dynamische Datenstrukturen wie auch Datenpuffer, Auftragsdeskriptoren oder Kontrollblöcke typische Beispiele, für die Klasse ↑konsumierbares Betriebsmittel sind dies Signale oder Nachrichten.

resume (dt.) fortsetzen; die Ausführung einer ↑Koroutine übernehmen. Einen ↑Koroutinenwechsel durchführen, indem von einem ↑Handlungsstrang der die Kontrolle über den ↑Prozessor abgebenden Koroutine hin zu einem Handlungsstrang der diese Kontrolle annehmenden Koroutine umgeschaltet wird.
Liegt der ↑CPU ein ↑orthogonaler Befehlssatz zugrunde, kann diese Operation leicht durch einen einzigen ↑Maschinenbefehl ausgedrückt werden. Beispiel dafür ist/war die ↑PDP 11/40, bei der die ↑Aktion `JSR PC, @(SP)+` einem Koroutinenwechsel entspricht. Diese Aktion (*jump to subroutine*):

1. sichert die auf dem ↑Laufzeitstapel liegende und von dort (mittels `@(SP)+`) entfernte Fortsetzungsadresse der zu aktivierenden Koroutine in ein internes ↑Prozessorregister,
2. legt den Inhalt des ↑Verbindungsregisters (`PC`) auf den Stapel ab und sichert damit die Fortsetzungsadresse der die Kontrolle über die CPU abgebenden Koroutine,
3. sichert die Adresse des (`JSR`) folgenden Maschinenbefehls als Rücksprungadresse in das angegebene Verbindungsregister (`PC`: diesbez. ein ↑Leerbefehl im gg. Fall) und
4. weist dem Befehlszähler die im ersten Schritt gesicherte Zieladresse zu, führt damit den Sprung aus und schaltet so zur adressierten Koroutine um.

Am Beispiel dieses Befehls wird vor allem deutlich, dass, analog zu ↑Routinen, Koroutinen in urwüchsiger Ausprägung (Reinform, *minimale Basis*) denselben Laufzeitstapel gemeinsam haben: ↑primitive Koroutine. Demgegenüber gibt es Modelle (*minimale Erweiterungen*), die jeder Koroutine einen eigenen Laufzeitstapel zugestehen: ↑komplexe Koroutine.

Ringschutz Abschirmung einer einzelnen ↑Entität oder einer Menge von Enitäten durch einen ↑Schutzring. Dazu bildet eine solche Enität ein ↑Objekt, das einen bestimmten Schutzring als Attribut besitzt und damit logisch auf eben diesen Ring residiert. In dem Ansatz liegen mehrere Schutzringe konzentrisch übereinander, wobei der unterste (innerste), Ring 0, die höchste Privilegstufe definiert und jeder darüber liegende (äußere) Ring in Folge jeweils eine niedrigere Privilegstufe hat.
Für jeden ↑Prozess gelten festgelegte Regeln, damit der Zugriff auf ein Objekt gelingt. Dazu ist neben der Ringzugehörigkeit des Objekts auch die des Prozesses von Bedeutung. Maßgeblich für die Ringzugehörigkeit des Prozesses ist der Kontext, in dem sich der Prozess jeweils befindet: nämlich der den Prozess definierende ↑Text eines bestimmten Objektes und damit der Ring dieses Objektes. Nur Zugriffe innerhalb eines Rings oder von einen inneren auf einen äußeren Ring sind zulässig. Zugriffe von einen äußeren auf einen inneren Ring werden abgefangen (↑*trap*) und der ↑Teilinterpretation unterzogen. Die Folge davon ist entweder ein ↑Schutzfehler mit anschließendem Prozessabbruch oder die Zugriffserlaubnis. Wird dem Prozess der Zugriff direkt oder indirekt (durch Teilinterpretation) gewährt, ist er in der

Lage, die ↑Daten eines Objekts zu lesen oder zu schreiben beziehungsweise den Objekttext auszuführen. Letzteres ermöglicht den Prozess zum Schutzringwechsel: der Prozess agiert danach auf dem Ring, der durch das Objekt definiert ist, dessen Text den weiteren Prozessverlauf nunmehr vorschreibt.
Diese Technik wurde erstmals mit ↑Multics vorgestellt, in dem Fall mit 64 logischen (durch das ↑Betriebssystem implementierten) und 8 realen (durch die Hardware bereitgestellten) Ringen. Intel hatte (lange nach Multics) mit dem ↑i286 eine CPU mit vier Ringen auf den Markt gebracht. Alle nachfolgenden Prozessoren dieser Familie stellen dieses Merkmal weiterhin zur Verfügung.

RISC Abkürzung für (en.) *reduced instruction set computer*, (dt.) ↑Rechner mit vermindertem (primitiven) ↑Befehlssatz. Der ↑Prozessor in solch einem Rechner verfügt über vergleichsweise wenige, dafür aber hoch optimierte Befehle. Ein weiteres typisches Merkmal ist, dass Berechnungen ausschließlich ↑Prozessorregister als Platzhalter für die Operanden verwenden, wozu letztere explizit durch Ladebefehle aus dem ↑Arbeitsspeicher zu lesen sind und das Berechungsergebnis explizit durch einen Speicherbefehl dahin zurückzuschreiben ist (*load/store architecture*). Für gewöhnlich sind die Befehlslängen gleich, so dass jeder Maschinenbefehl gleich viel Platz im Arbeitsspeicher belegt. Darüber hinaus ist bei dieser ↑Rechnerarchitektur angestrebt, jeden Befehl in gleicher Anzahl von Taktzyklen auszuführen. Anders als beim ↑CISC ist der Grundgedanke hinter der ↑Rechnerarchitektur, ↑Performanz vor allem durch Schlichtheit zu steigern und die ↑semantische Lücke nur durch Maßnahmen in Software zu verkleinern. Typische Beispiele sind die Prozessoren der ↑PowerPC-Familie.

RM Abkürzung für (en.) *rate monotonic*. Bezeichnung für ↑deterministische Planung von unterbrechbaren, periodischen ↑Prozessen. Dabei nimmt die ↑Priorität eines Prozesses zu, je kürzer seine Periode und je höher damit seine Ankunftsrate ist. Diese Festlegung der jeweiligen Priorität geschieht statisch (↑*offline scheduling*). Das auf ↑Vorwissen basierende Verfahren findet für gewöhnlich beim ↑Echtzeitbetrieb Verwendung.

memory chunk (dt.) ↑Speicherstück.

ROM Abkürzung für (en.) *read-only memory*, (dt.) ↑Festwertspeicher.

root directory (dt.) ↑Wurzelverzeichnis.

root file system (dt.) ↑Stammdateisystem.

round robin Abgeleitet von (frz.) *ruban rond*, dem Sinn nach (dt.) Kreisband. Ursprünglich die Bezeichnung für einen Bitt- und Beschwerdebrief (an den frz. König, 17./18. Jhd.), der von den Verfassern in Kreisform unterschrieben wurde, um dadurch Spekulationen über eine mögliche Rangfolge oder der Identifikation von „Rädelsführern“ zu erschweren und so der eventuellen Hinrichtung vorzubeugen. Übernommen als bildhafte Beschreibung für ein ↑Zeitteilverfahren, das die ↑präemptive Planung von ↑Prozessen ermöglicht (Kleinrock, 1964).

Routine Bezeichnung für ein kleineres ↑Programm oder Teil eines Programms mit einer bestimmten, gewöhnlich häufiger benötigten Funktion (Duden).

routing (dt.) Wegewahl, ↑Leitweglenkung.

RR Abkürzung für (en.) ↑*round robin*. Bezeichnung für eine Strategie zur ↑Planung der Bearbeitung von ↑Aufgaben durch einen ↑Prozessor im Reihumverfahren (↑Rundlauf). Dem grundlegenden Schema nach ↑FCFS, jedoch steht die ↑präemptive Planung im Zentrum des Verfahrens, womit auch die Einstufung als ↑mitlaufende Planung begründet ist. Der die Aufgaben verarbeitende Prozessor ist allgemeint charakterisiert als ↑wiederverwendbares Betriebsmittel, das im ↑Zeitteilverfahren benutztbar sein muss: das heißt, dieser Prozessor kann kein ↑unteilbares Betriebsmittel sein. Damit kann einer laufenden Aufgabe zu gegebener Zeit zugunsten einer bereitstehenden Aufgabe der Prozessor entzogen und später wieder zugeteilt

werden (↑Verdrängung). Typischerweise die von einem Prozessor zu bearbeitende Aufgabe einem ↑Prozess und der Prozessor selbst entspricht der ↑CPU beziehungsweise einer ihrer möglicherweise mehreren ↑Rechenkerne.
Bei dem Verfahren erhält ein Prozess den Prozessor immer nur für eine ↑Zeitscheibe lang zugeteilt, dabei ist die Länge der Zeitscheibe für gewöhnlich für alle Prozesse gleich. Innerhalb dieses Zeitintervalls endet der ↑Rechenstoß des Prozesses entweder freiwillig oder unfreiwillig, letzteres als Folge einer ↑Unterbrechungsanforderung beim Ablauf der Zeitscheibe. Die Unterbrechung des Rechenstosses führt zur erneuten ↑Einplanung (FCFS) des betreffenden Prozesses. Letzteres geschieht jedoch nur dann, wenn im Moment des Zeitscheibenablaufs wenigstens ein anderer Prozess (auf der ↑Bereitliste stehend) die Prozessorzuteilung erwartet. Ist dies der Fall, wird der der nächste bereitstehende Prozess in Folge (FCFS) der ↑Einlastung des Prozessors zugeführt. Dieser Vorgang wiederholt sich fortlaufend.
Das mit FCFS bestehende Problem des ↑Konvoieffekts wird stark abgeschwächt. Da im Allgemeinen davon auszugehen ist, dass ein-/ausgabeintensive Prozesse ihre Rechenstöße noch vor Ablauf der Zeitscheide beenden, den Prozessor von selbst abgeben, und nur rechenintensive Prozesse durch Ablauf ihrer Zeitscheibe unterbrochen werden, ist dieser Effekt allerdings nicht gänzlich vermeidbar. Jedoch ist die Schwere des Problems nicht mehr durch die für gewöhnlich unbekannte Rechenstoßlänge bestimmt, sondern durch die bekannte Zeitscheibenlänge und damit auch begrenzt. Je größer letztere allerdings ist, umso stärker entartet das Verfahren zum bloßem FCFS und umso mehr gewinnt der Konvoieffekt an Bedeutung; je kleiner sie ist, desto häufiger erfolgen programmierte Unterbrechungen und desto höher sind die ↑Gemeinkosten des Verfahrens. Als Faustregel gilt daher, dass die Länge der Zeitscheibe etwas größer sein soll als die Dauer eines typischen Rechenstoßes. Genau genommen ist dieser Parameter jedoch abhängig von Art und Aufbau einer jeweiligen ↑Anwendung und ist durch Testläufe zu bestimmen, er basiert aber nicht selten auch nur auf Erfahrungswerte: typische Standards sind 100ms (↑BSD, ↑Linux) oder Abstufungen von 20ms, 40ms, 60ms und 120ms (↑Windows NT, je nach Konfiguration).

Exkurs Im Vergleich zu FCFS (vgl. S. 46) besteht der Unterschied in der Implementierung vor allem in der Notwendigkeit (a) einen ↑Zeitgeber samt ↑Gerätetreiber und diesbezüglicher ↑Unterbrechungsbehandlung vorzusehen sowie (b) zur ↑Synchronisation der Operationen des mitlaufenden ↑Planers. Genau genommen ist jedoch zwischen der synchronen und asynchronen Auslegung von FCFS zu differenzieren: nur wenn strikt synchrones FCFS die Basis für das hier vorgestellte präemptive Verfahren bildet, sind entsprechende Erweiterungen (nichtfunktionaler Art) vorzunehmen, um nämlich die benötigte asynchrone Nutzung zu ermöglichen. Die asynchrone Form von FCFS setzt für gewöhnlich bereits synchronisierte Planeroperationen voraus. Hier ist dann nur noch eine Vorkehrung zu treffen, die den gegenwärtig stattfindenden (d.h., unterbrochenen) Prozess zwingt, den Prozessor freiwillig abzugeben. Dazu muss im Rahmen der durch Ablauf der Zeitscheibe ausgelösten Unterbrechungsbehandlung aber lediglich eine Planeroperation zur bedingten Prozessorabgabe (`check`, S. 47) abgesetzt werden.
Sind die Planeroperationen durch ↑nichtblockierende Synchronisation geschützt, können sie jederzeit auch (vom Gerätetreiber ausgehend) asynchron aufgerufen und durchgeführt werden. Anderenfalls darf die Operation zur unbedingten Prozessorabgabe (`pause`, S. 47) nur dann asynchron zur Wirkung kommen, wenn im Moment des Zeitscheibenablaufs der Planer inaktiv ist. Eine einfache Lösung, die allerdings nur vor Asynchronität durch Ablauf der Zeitscheibe auf einem ↑Uniprozessor schützt, zeigt nachfolgendes Beispiel:

```
void check() {                                  /* try to reschedule processor */
    backlog_t *list = labor();                                 /* use ready list */
    if (ahead(list))                             /* any other process present? */
        if (FAA(&list->omen, 1) == 0)    /* yes, scheduler already active? */
            pause();                                        /* no, reschedule... */
}
```

Dabei wird mit ↑FAA geprüft, ob der Planer bereits aktiv ist und gleichzeitig der Aktivzustand ($omen > 0$) des Planers definiert. Beide Schritte geschehen atomar. Ist der Planer bereits aktiv, endet die Operation ohne Prozessorabgabe. In dem Fall wird jedoch der Zustand ($omen > 1$) hinterlassen, dass nämlich die Prozessorabgabe angefordert wurde. Alle anderen Planeroperationen müssen dann nach folgendem Muster umgestaltet werden:

```
{
    backlog_t *list = labor();                          /* use ready list */
    FAA(&list->omen, 1);                               /* scheduler active */
    ...                                  /* do respective scheduler operation */
    if (FAA(&list->omen, -1) > 0) {     /* scheduler inactive: preemption? */
        list->omen = 0;                             /* yes, quit that request */
        pause();                                      /* release processor... */
    }
}
```

In dem Beispiel werden alle kritischen Planeroperationen durch eine ↑Verdrängungssperre geschützt. Da im gesperrten Fall eine Verdrängungsanforderung jedoch nicht verloren gehen sollte, wird am Ende jeder kritischen Operation auf eine solche zwischenzeitlich eingegangene Anforderung geprüft und gegebenenfalls die ↑Verdrängung (`pause`) nachgezogen.

RSX 11 Familie von (Mehrplatz-/Mehrbenutzer-) Echtzeitbetriebssystemen für ↑Rechner der PDP-11 Familie (DEC), erste Installation 1972 (↑PDP 11/40). Abkürzung ursprünglich für „*Real-Time System Executive*“, später für „*Resource Sharing Executive*“. Bewährte Prinzipien von RSX-11M (1973, u.a. ↑AST) wurden später für ↑VMS übernommen, das unter derselben Leitung (David Cutler) entstand.

run to completion (dt.) laufen bis zur Fertigstellung. Bezeichnung für eine ↑Ablaufsteuerung, die sicherstellt, dass ein ↑Prozess die mit ihm verbundene ↑Aufgabe zügig erfüllen kann, ohne durch ↑Verdrängung unterbrochen zu werden. Nach erfolgter ↑Einlastung gibt der Prozess den ↑Prozessor erst wieder ab wenn er:

1. entweder durch ↑logische Synchronisation mit einem anderen (ggf. externen) Prozess das Ergebnis einer benötigten Zuarbeit abwarten und dazu ein diesbezügliches ↑konsumierbares Betriebsmittel erwerben (*acquire*) muss, oder
2. die Aufgabe komplett erfüllt hat.

Der erste Punkt impliziert eine ↑Ablaufplanung, die nur den zum Fortschritt des logisch synchronisierten (d.h., kausal abhängigen) Prozesses ihren Teil jeweils beitragenden anderen Prozessen bevorzugt den/einen Prozessor zuteilt. Dieser Aspekt wird jedoch nicht immer als zwingend erforderliches Merkmal angesehen, mit der Folge, dass der betreffende Prozess, trotz ↑kooperative Planung im Grundsatz, durch kausal nicht zusammenhängende Prozesse unerwünschte Verzögerung erfahren kann. Zur Minimierung der ↑Bearbeitungszeit des logisch synchronisierten Prozesses spielt jedoch der kausale Zusammenhang mit anderen Prozessen eine entscheidende Rolle. Dies bedeutet dann allerdings auch ↑Vorwissen zu den Daten- und gegebenenfalls auch Zeitabhängigkeiten zumindest jener Gruppe von Prozessen, für die diese Art von Ablaufsteuerung gelten soll.

run-time system (dt.) ↑Laufzeitsystem.

Rundlauf Merkmal einer ↑Ablaufplanung. bei der jeder einzelne ↑Prozess im stetigen Wechsel mit anderen Prozessen solange die ↑CPU zugeteilt bekommt, bis er seine ↑Aufgabe erfüllt hat. Dabei wird der Wechsel durch das ↑Betriebssystem erzwungen (↑*preemption*), als Folge der ↑Unterbrechung des gegenwärtig auf der CPU stattfindenden Prozesses. Der unterbrochene Prozess wird daraufhin neu eingeplant.

Rundruf Ruf, der an jede ↑Entität innerhalb einer bestimmten Gruppe geht (in Anlehnung an den Duden).

safety (dt.) ↑Betriebssicherheit.

Satellitenrechner Rechenanlage, die für eine andere (für gewöhnlich größere) Rechenanlage vor- und nachbereitende Aufgaben wahrnimmt. Eine solche Anlage vermittelt einem ↑Hauptrechner den Zugang zur ↑Peripherie, an ihr sind verschiedene Ein-/Ausgabegeräte unterschiedlicher Art und Geschwindigkeit angeschlossen. Eingaben werden entgegengenommen und samt verarbeitende Programme zu einem ↑Stapel zusammengefasst, der dann über ein schnelles ↑Peripheriegerät (Band-/Wechselplattenlaufwerk, Steuergerät zur Hochgeschwindigkeitskommunikation) zu gegebener Zeit zum Hauptrechner übertragen wird. Über ein solches Gerät werden ebenfalls die vom Hauptrechner erzeugten Ausgaben entgegengenommen und den zugehörigen Ausgabegeräten zugestellt.

schedule (dt.) ↑Ablaufplan.

scheduler (dt.) ↑Planer.

scheduling (dt.) ↑Ablaufplanung, ↑Planung des zeitlichen Ablaufs.

scheduling criteria (dt.) ↑Einplanungskriterium.

scheduling latency (dt.) ↑Einplanungslatenz.

Schlafzustand Systemzustand, der die einem ↑Prozessor bereitgestellte Energie, ihm angelegte Leistung beschreibt. Je tiefer dieser Zustand, desto geringer der Energieverbrauch beziehungsweise die Leistungsaufnahme des Prozessors, umso länger dauert seine Aufwachphase und umso umfassender sind die Maßnahmen in einem ↑Betriebssystem zur Aufrechterhaltung von Hardwarekontext (insb. ↑CPU, ↑Zwischenspeicher und ↑Hauptspeicher). Der Grad der Abstufung ist prozessorabhängig, heutige (2016) Prozessoren beinhalten eine bis fünf Stufen. Das Betriebssystem bringt den Prozessor zum Schlafen, sobald für ihn ↑Leerlauf festgestellt wurde. Der Prozessor weckt wieder auf, wenn er eine ↑Unterbrechungsanforderung erhält.

Schlange (en.) ↑queue. Bezeichnung für eine dynamische Datenstruktur zur gemeinhin vorübergehenden Aufbewahrung von ↑Objekten, deren Anzahl über die Zeit verschieden und gegebenenfalls auch nicht im Voraus bestimmbar ist. Mit dieser Datenstruktur ist eine typische Art und Weise der Reihung verbunden, bei der ein aufzubewahrendes Objekt ans Ende (*tail*) der Schlange angefügt (*enqueue*) und ein aufbewahrtes Objekt vom Anfang (*head*) der Schlange entfernt (*dequeue*) wird. Nachfolgend ein Beispiel in ↑C, das die Einreihung des Endelements mit konstantem und das Entfernen des Kopfelements mit begrenztem Aufwand durchführt:

```
typedef struct chain {
    struct chain *link;
} chain_t;

typedef struct queue {
    chain_t head;
    chain_t *tail;
} queue_t;
```

Um den konstanten Einreihungsaufwand zu erreichen verwendet diese Implementierung einen Zeiger (`tail`) auf die Stelle, an der das jeweils nächste Element eingehängt werden kann. Initial ist diese Stelle der Kopfzeiger (`head`). Dadurch wird beim Einfügen des ersten Elements in eine leere Liste implizit der Kopfzeiger mit initialisiert. Als Folge entfällt die diesbezügliche Fallunterscheidung, die sonst einen nur noch nach oben begrenzten Berechnungsaufwand mit sich bringt.

Beim Entfernen des letzten Elements ist, wie sonst üblich, dem Endzeiger einen definierten Wert zu geben, der im gegebenen Beispiel, anstatt mit Null, nunmehr mit der ↑Adresse des Kopfzeigers initialisiert wird. Die betreffenden Operationen gestalten sich wie folgt:

```
void enqueue(queue_t *list, chain_t *item) {
    item->link = 0;            /* element becomes new tail, mark the stop */
    list->tail->link = item;            /* add element at current linkage */
    list->tail = item;                        /* advance linkage pointer */
}

chain_t *dequeue(queue_t *list) {
    chain_t *item = list->head.link;       /* fetch stored element, if any */
    if (item && (list->head.link = item->link) == 0) /* last one fetched? */
        list->tail = &list->head;            /* yes, reset linkage pointer */
    return item;                            /* deliver fetched element or 0 */
}
```

Dieses Prinzip entspricht der auch als ↑FCFS oder ↑FIFO bezeichneten Vorgehensweise zur Bearbeitung anstehender ↑Aufgaben oder ↑Daten.

Schreibmarke Bezeichnung der Bildschirmmarke für die aktuelle Bearbeitungsposition.

Schutz Vorrichtung, die eine Gefährdung von einem ↑Prozess abhält oder einen Schaden in einem ↑Rechensystem abwehrt (in Anlehnung an den Duden). Die hierzu im Grundsatz erforderlichen Maßnahmen haben einerseits einen räumlichen und zeitlichen Aspekt der Isolation von Prozessen und sind andererseits bestimmt durch die jeweilige ↑Betriebsart des Rechensystems: sie sind im Allgemeinen nur typisch für ↑Mehrprogrammbetrieb und bei ↑Simultanverarbeitung. In räumlicher Hinsicht ist (bei Mehrprogrammbetrieb) die Integrität der einem Prozess eigenen Anteile von ↑Text und ↑Daten sicherzustellen. Dies wird dadurch erreicht, indem entweder die jeweilige ↑Adresse eines solchen Anteils von keinem anderen Prozess in Erfahrung gebracht werden oder kein Prozess aus seinem ↑Prozessadressraum ausbrechen kann. Ersterer Ansatz geht von einem ↑Einadressraummodell aus, wohingegen letzterer Ansatz ein ↑Mehradressraummodell zur Grundlage hat. Beide Modelle sind zentraler Bestandteil der ↑Schutzdomäne von einem Prozess.
In zeitlicher Hinsicht ist (zur Simultanverarbeitung) zusätzlich sicherzustellen, dass kein Prozess die ↑CPU monopolisieren und damit ununterbrochen nur noch für sich selbst belegen kann. Dies stellt die Anforderung an die ↑Prozesseinplanung, ein ↑Zeitteilverfahren zur Grundlage zu nehmen. Ist als Betriebsart ein Mischbetrieb vorgesehen, bei dem unter anderem der ↑Echtzeitbetrieb von bestimmten Prozessen unterstützt werden soll, muss diese Gruppe von Prozessen weitestgehend vor störenden Einflüssen (↑Interferenz) in ihrem Ablauf geschützt werden. Dies betrifft vor allem Verfahren zur ↑Virtualisierung, wobei hier die besondere Schwierigkeit besteht, dass das zugrunde liegende Steuerprogramm (↑VMM) für gewöhnlich keine Kenntnis über die Prozesse hat, die eine ↑virtuelle Maschine ermöglicht: weder eine reale noch eine virtuelle Maschine kennt die Bedeutung von dem ↑Programm, das sie ausführt; ihr ist damit auch nicht bewusst, dass dieses Programm ein Betriebssystem sein könnte und möglicherweise ↑Ablaufplanung nach besonderen benutzerorientierten, zeitlichen Kriterien durchsetzen muss. Gilt zudem umgekehrt, dass kein Prozess in Erfahrung bringen kann, ob er auf einer realen oder virtuellen Maschine stattfindet, ist die Durchsetzung solcher zeitlichen Kriterien bei gleichzeitig (auf derselben realen Maschine) laufenden virtuellen Maschinen nicht mehr gewährleistet: das zur Virtualisierung einer realen Maschine notwendige ↑Multiplexverfahren wird in aller Regel einen Konfliktherd zu der jeweils auf einer virtuellen Maschine laufenden ↑Prozesseinplanung bilden.

Schutzdomäne Gebiet, auf dem für einen ↑Prozess ein bestimmter ↑Schutz gewährleistet ist. Jedes diesem Gebiet zugeordnete ↑Objekt ist dem Prozess (↑Subjekt) in bestimmter Weise zugänglich. Für gewöhnlich ist damit eine Menge von Objekten definiert, auf die ein Prozess zugreifen kann und mit jedem Objekt darin ist eine Menge von Rechten verbunden, die das ↑Zugriffsrecht des Prozesses festlegt. Prozesse können solche Domänen wechseln, jedoch nur unter Kontrolle der (realen/virtuellen) Maschine, auf die er stattfindet. Typisches Beispiel für solch einen Domänenwechsel ist der ↑Systemaufruf, bei dem der Prozess (a) seinen ↑Pro-

zessor in den privilegierten ↑Arbeitsmodus bringt und (b) in einen anderen oder erweiterten ↑Adressraum, je nach ↑Mehradressraummodell, wechselt.
Zusätzlich zu dem jeweiligen ↑Prozessadressraum wird — bei einem ↑UNIX-artigen ↑Betriebssystem — dieses Gebiet durch die jeweilige ↑UID und ↑GID eines Prozesses abgesteckt. Dazu erhält jedes vom Betriebssystem verwaltete und Prozessen zugängliche Objekt ein {UID, GID}-Paar bei seiner Erzeugung zugewiesen. Dieses Paar nimmt für gewöhnlich die Werte der entsprechenden Kennungen des Prozesses an, der die Objekterzeugung effektiv ausgelöst (z.B. mit `creat(2)` eine ↑Datei angelegt) hat. So lässt sich für jede gegebene {UID, GID}-Kombination eine Menge von Objekten erstellen, auf die ein Prozess zugreifen kann. Alle Prozesse mit der gleichen Kombination von {UID, GID}-Werten besitzen Zugriff auf exakt die gleiche Menge von Objekten. Diese Mengen (d.h., Domänen) sind verschieden für Prozesse mit unterschiedlichen {UID, GID}-Kombinationen.

Schutzfehler Zugriffsfehler in Bezug auf das ↑Schutzsystem einer (realen/virtuellen) Maschine. Betrifft die Verletzung von ↑Speicherschutz, aber auch von Rechten, die ein ↑privilegierter Befehl von einem ↑Prozess erfordert. Stellt eine ↑Ausnahmesituation dar.

Schutzgatter ↑Einfriedung durch eine unveränderliche ↑Gatteradresse. Die Adresse ist „fest verdrahtet“ (*hard-wired*) in der Hardware und bestimmt Position wie auch Größe der geschützten Zone für das ↑Betriebssystem im ↑Hauptspeicher. Nach dem ↑Urladen des Betriebssystems ist der unbelegte Bereich in dieser Zone entweder ↑interner Verschnitt oder als ↑dynamischer Speicher für das Betriebssystem nutzbar. Die Gatteradresse ist dem ↑Binder bekannt, der damit dann die ↑Relokation von dem jeweiligen ↑Maschinenprogramm durchführt, bevor es vom ↑Lader des Betriebssystems in die ungeschützte Zone gebracht wird.
Dieses aus den Anfängen der Betriebssystementwicklung stammende Schutzkonzept findet genau genommen auch heute (2016) noch in ↑Linux und ↑Windows Verwendung. Hier ist der ↑Adressbereich $A = [0, 2^N - 1]$, den eine ↑CPU mit ↑Adressbreite N in logischer Hinsicht abdecken kann, ebenfalls zweigeteilt, wenn nämlich ein ↑logischer Adressraum oder ein ↑virtueller Adressraum durch das Betriebssystem bereitgestellt werden soll. Für $N = 32$ (4 GiB ↑Adressraum), beispielsweise, wird dem Maschinenprogramm eine Zone (vorderer Adressbereich) zugeteilt, die entweder 50% (Windows) oder 75% (Windows /3GB-Schalter, Linux) dieses Adressbereichs entspricht. Der übrige Adressbereich, der letztlich die geschützte Zone (hinterer Adressbereich) für das Betriebssystem ausmacht, ist einem Maschinenprogramm unter Linux/Windows nicht zugänglich. Andererseits ist Linux/Windows die ungeschützte Zone sehr wohl zugänglich. Dieser 50% beziehungsweise 75% Schnitt in dem Adressbereich definiert letztlich eine Gatteradresse in einem logischen/virtuellen Adressraum.

Schutzgatterregister ↑Einfriedung durch eine veränderliche ↑Gatteradresse. Die Adresse ist in einem speziellen ↑Prozessorregister der ↑CPU gespeichert. Der Registerinhalt bestimmt Position wie auch Größe der geschützten Zone für das ↑Betriebssystem im ↑Hauptspeicher. Nach dem ↑Urladen des Betriebssystems markiert dessen Anfangs- oder Endadresse im Hauptspeicher die Gatteradresse, je nachdem, ob die geschützte Zone den hinteren oder vorderen ↑Adressbereich ausmacht. Diese Adresse wird sodann in das ↑Register geschrieben, bevor das Betriebssystem dem ↑Maschinenprogramm in der ungeschützten Zone die Kontrolle übergibt. Das Beschreiben dieses Registers ist eine privilegierte Operation, ihre Ausführung heraus aus der ungeschützten Zone abgefangen (↑*trap*). Diese Operation ist nur dem Betriebssystem erlaubt (↑*operating mode*), da das Maschinenprogramm sonst die geschützte Zone auf den eigenen Adressbereich ausdehnen kann. Während der Ausführung von einem Maschinenprogramm ist die Gatteradresse fest. Bevor das nächste Maschinenprogramm geladen wird und zur Ausführung kommt, kann das Betriebssystem die Gatteradresse verändern, um sich mehr oder weniger Platz im Hauptspeicher zu geben. Da in dem Ansatz die Gatteradresse zur ↑Bindezeit eines Maschinenprogramms als undefiniert gilt, muss im Betriebssystem ein ↑verschiebender Lader tätig sein, um die Maschinenprogramme in die ungeschützte Zone des

Hauptspeichers zu bringen. Dieser Lader nimmt die zur ↑Ladezeit gültige Gatteradresse als ↑Relokationskonstante und richtet das Maschinenprogramm entsprechend aus.

Schutzring (en.) ↑*protection ring*. Bezeichnung für eine bestimmte Art von ↑Schutzdomäne einer ↑Entität — beziehungsweise von einem ↑Subjekt oder ↑Objekt, je nachdem, ob eine aktive oder passive Haltung eingenommen wird. Der Ring bildet einen Mechanismus, um eine bestimmte Menge solcher Entitäten vor möglichen Auswirkungen von Fehlern (↑*safety*) oder böswilligem Verhalten (↑*security*) anderer Prozesse zu bewahren. Der damit ermöglichte ↑Schutz wird auch als ↑Ringschutz bezeichnet.

Schutzsystem Gesamtheit von technischen Maßnahmen, um ↑Schutz zu gewährleisten.

Schutzverletzung ↑Aktion, die einen ↑Schutzfehler (insb. Verletzung von ↑Speicherschutz) verursacht.

schwache Konsistenz Modell der ↑Speicherkonsistenz, für das eine von einem ↑Prozessor explizit einzurichtende Absperrung (↑*memory barrier*) zur ↑Synchronisation laufender Operationen auf dem ↑Arbeitsspeicher die Grundlage bildet. Die Absperrung bewirkt, dass alle ausstehenden Speicheroperationen aller Prozessoren zum Abschluss gebracht und neue Speicheroperationen erst nach erfolgter Synchronisation wieder ausgegeben werden.

Schwebezustand Bezeichnung für den Zustand der Unklarheit, der Unsicherheit, der Unentschiedenheit über einen bestimmten ↑Prozesszustand (in Anlehnung an den Duden). Je nach ↑Betriebsart können für einen ↑Prozess bestimmte Zwischenzustände definiert sein, die Einfluss auf seine ↑Einplanung beziehungsweise ↑Einlastung ausüben.
Ermöglicht das ↑Betriebssystem die ↑Umlagerung der Inhalte von ↑Speicherbereichen, kann der Zustand des betreffenden Prozesses zusätzlich zu bereit oder blockiert jeweils auch schwebend (*pending*) sein. Dies ist dann der Fall, wenn sämtliche von einem ↑Prozessadressraum abgedeckten Bereiche im ↑Hauptspeicher komplett im ↑Hintergrundspeicher ausgelagert sind. Ein solcher Prozess kann zwar weiterhin eingeplant, aber nicht eingelastet werden: letzteres erfordert zunächst die Einlagerung seiner Hauptspeicherbereiche. Demgegenüber kann ein Prozess, der schwebend blockiert ist, sehr wohl in den Zustand schwebend bereit überführt werden, ohne dass er dazu erst eingelagert werden müsste. Diese Zustandsüberführung hat nämlich nur zur Folge, den (nicht umgelagerten) ↑Prozesskontrollblock des betreffenden Prozesses auf die ↑Bereitliste zu setzen.
Eine weitere Quelle von Zwischenzuständen ist die ↑mitlaufende Planung einerseits und die gleichzeitig dazu stattfindende Bereitstellung eines Prozesses andererseits, wobei letzteres entweder indirekt im Rahmen einer ↑Unterbrechungsbehandlung geschieht oder unterbrechungsfrei direkt von einem anderen ↑Prozessor oder ↑Rechenkern aus erfolgt. Vor solch einem Hintergrund können sich folgende Situationen ergeben:

- Ein Prozess, der zum weiteren Fortschritt erst den Eintritt eines bestimmten ↑Ereignisses erwarten muss (↑logische Synchronisation), wird in den Zustand blockiert übergehen. Diese ↑Aktion führt der Prozess selbständig durch, bevor er den Prozessor abgibt, das heißt, während er noch im Zustand laufend ist. Prozesse können demnach laufend blockiert sein.
- Ist in diesem Moment kein anderer Prozess im Zustand bereit, die Bereitliste also leer, kann der Prozess die Rolle als ↑Leerlaufprozess übernehmen. Anstatt den Prozessor abzugeben wird der Prozess untätig. Prozesse können demnach laufend blockiert und untätig sein.
- Ein als blockiert ausgewiesener Prozess kann in den Zustand bereit übergehen und damit auf die Bereitliste gelangen, nämlich sobald das von ihm erwartete Ereignis eintritt. Das kann insbesondere geschehen, obwohl es der Prozess noch nicht geschafft hat, den Prozessor abzugeben. Prozesse können demnach laufend bereit und, im Falle der Rolle als Leerlaufprozess, dabei auch noch untätig sein.

Entsprechend bietet es sich an, den Prozesszustand als eine *Menge* von Statusattributen und nicht als Aufzählung von Zahlenwerten zu kodieren. Ist diese Menge sodann kompakt als *Bitleiste* gespeichert, sind vergleichsweise komplexe Mengenoperationen (↑*read-modify-write*) notwendig, nämlich zur Bildung der Vereinigungs- oder Schnittmenge, um einen bestimmten Zwischenzustand zu definieren (Disjunktion, `or`) beziehungsweise aufzuheben (Konjunktion, `and`). Bei Speicherung als *Zeichenfeld* (z.B., `char[4]`) entarten die Mengenoperationen jedoch zu einfachen Setz- und Rücksetzbefehlen (Zuweisung, `mov`). Letzteres ist gerade bei ↑Parallelverarbeitung besonders vorteilhaft, da die dann erforderlichen atomaren Mengenoperationen allein mit Wahrung der ↑Speicherkohärenz durch die ↑CPU gegeben sein können.

schwergewichtiger Prozess Bezeichnung für einen ↑Prozess, der als ↑Systemkernfaden allein im eigenen und durch ↑Speicherschutz isolierten ↑Adressraum stattfindet. Auf ↑Multics zurückgehendes Verständnis einer ↑Prozessinkarnation, nämlich der Gleichsetzung von ↑Prozess und physisch abgeschottetem Adressraum.

scratchpad memory (dt.) ↑Notizblockspeicher.

scripting language (dt.) ↑Skriptsprache.

security (dt.) ↑Angriffssicherheit.

segfault Abkürzung für (en.) ↑*segmentation fault*.

Segment Unterteilung im ↑Prozessadressraum, wobei der ↑Adressbereich dieser Unterteilung auf den ↑Speicher eines Rechensystems abgebildet ist, genauer: dem ↑Vordergrundspeicher und ↑Hintergrundspeicher. In diesem Bereich liegt Programmtext oder -daten, auch bezeichnet als ↑Textsegment und ↑Datensegment. Ein solcher Adressraum kann mehrere dieser Bereiche umfassen, die sich nicht überlappen und von fester oder variabler Größe sind. Beispiele für Segmente sind grobkörnige Bereiche wie ganze Dateien oder der gesamte Text- oder Datenbereich von einem ↑Programm oder feinkörnige Gebilde wie ein einzelnes ↑Unterprogramm, Datenstrukturen, Objekte oder Variablen.

segment fault (dt.) ↑Segmentfehler.

segment selector (dt.) ↑Segmentwähler.

Segmentadressierungseinheit Funktionsbaustein einer ↑MMU, der eine ↑logische Adresse in eine ↑reale Adresse umsetzt. Definitionsbereich der logischen Adresse ist ein ↑segmentierter Adressraum. Die Abbildung dieses Adressraums geschieht durch eine ↑Segmenttabelle, der Bildbereich stellt sich als ↑realer Adressraum dar.

segmentation (dt.) ↑Segmentierung.

segmentation fault (dt.) ↑Speicherzugriffsfehler, ↑Schutzverletzung.

segmentation unit (dt.) ↑Segmentadressierungseinheit.

Segmentdeskriptor Bezeichnung für einen ↑Deskriptor von einem ↑Segment. Datenstruktur einer ↑MMU (↑*segmentation unit*), um eine ↑logische Adresse in eine ↑reale Adresse umzusetzen — mehr dazu aber in SP2.

segmented paging (dt.) ↑segmentierte Seitenadressierung.

Segmentfehler Fehlgriff auf ein ↑Segment, verursacht einen ↑Bindefehler und hat ↑dynamisches Binden zur Folge. Zu dem Segment besteht eine ↑unaufgelöste Referenz, die im Moment der Verwendung eine ↑Ausnahme erhebt (↑*link trap*).

segmentierte Seitenadressierung Art der ↑Segmentierung von dem globalen ↑Adressraum, wobei ein bestimmter ↑seitennummerierter Adressbereich als Teilmenge durch ein ↑Segment erfasst wird: ein ↑seitennummerierter Adressraum wird segmentiert. Je nach ↑MMU kann diese Organisation implizieren, dass jedes Segment dann die ↑Seite als kleinstes Strukturelement haben muss. Aber ebenso ist es möglich, Segmente wie gehabt als Vielfaches von einem ↑Byte beibehalten zu können. Fällt ↑interner Verschnitt für die letzte Seite eines Segments an, dann liegt bei der byteweisen Segmentierung der anfallende Seitenrest außerhalb des Segments. Zugriffe darauf können als illegal erkannt und abgefangen werden (*segmentation violation*). Darüberhinaus ist solch ein Seitenrest gegebenenfalls nutzbar für ein anderes Segment, dessen erste Seite dann der diesen Rest aufweisenden Seite (am Ende eines anderen Segments) entspricht. Bei geeigneter MMU (↑i386, jedoch nur für Segmente von max. 64 KiB Größe, d.h., für den 16-Bit Schutzmodus) ist durch diese Technik interner wie auch ↑externer Verschnitt vermeidbar. Im Falle von ↑Speichervirtualisierung, ist jedes Segment zudem als ↑seitennummerierter virtueller Speicher ausgebildet.

segmentierter Adressraum Bezeichnung für einen ↑Adressraum, der zweidimensional ausgelegt ist: die erste Dimension benennt das jeweilige ↑Segment und die zweite Dimension benennt das ↑Speicherwort innerhalb dieses Segments. Entsprechend besteht eine ↑Adresse in solch einem Adressraum aus zwei Komponenten, nämlich der ↑Segmentname und der Nummer $[0, N-1]$ des Speicherworts in dem Segment von N Worten. Dabei kann ein solches Segment als ↑seitennummerierter Adressbereich ausgelegt sein, vorausgesetzt die zugrundeliegende ↑MMU, die zur Abbildung des zweidimensionalen Adressraums in jedem Fall erforderlich ist, unterstützt eine derartige Organisation (↑*segmented paging*).

Segmentierung Zerlegung eines komplexen Ganzen in einzelne Abschnitte (in Anlehnung an den Duden). Jeder Abschnitt bildet ein eigenständiges ↑Segment, das ↑Text oder ↑Daten enthält. Das komplexe Ganze entspricht einem ↑Adressraum.

Segmentname Bezeichnung von einem ↑Segment, typischerweise eine Nummer.

Segmentregister Vorkehrung mit der ein ↑Segment im ↑Hauptspeicher festgelegt wird. Spezielle ↑Prozessorregister, die als Paar die Lage (↑reale Adresse) und die Länge (Quantität einer ↑Speicherzelle) des Segments definieren: nämlich das Basisregister (*base register*) und das Längenregister (*limit register*). Im Gegensatz zu ↑Grenzregister sind die Registerinhalte selbst für einen stattfindenden ↑Prozess (d.h., ↑Prozesszustand laufend) veränderlich und es ist auch kein ↑verschiebender Lader erforderlich, um ein als Segment ausgebildetes ↑Maschinenprogramm in den ↑Hauptspeicher zu bringen. Der entscheidende Grund dafür ist, dass jede vom Prozess generierte effektive ↑Adresse eine ↑logische Adresse darstellt, die mit Hilfe des Basisregisters zur ↑Laufzeit vom ↑Prozessor erst noch auf eine ↑reale Adresse abgebildet wird: $A_r = A_l + \mathit{base\ register}$. Der Basisregisterinhalt gilt als ↑Relokationskonstante für einen einzelnen ↑Abruf- und Ausführungszyklus des Prozessors. Diese ↑Relokation einer logischen Adresse zur Laufzeit geschieht jedoch nur, wenn die betreffende Adresse gültig ist, das heißt, eine Speicherzelle in dem Segment referenziert. Vor jeder Relokation muss für die logische Adresse A_l gelten: $0 \leq A_l < \mathit{limit\ register}$. Ist der Adresswert größer oder gleich dem Inhalt des Längenregisters, wird ein Zugriff über diese Adresse abgefangen (↑*segmentation fault*). Jedes solcher Segmente ist damit ein ↑logischer Adressraum mit Wertebereich $[0, N-1]$, wobei N die Länge des jeweiligen Segments angibt.
Zur Erzeugung von dem ↑Lademodul für ein Maschinenprogramm weist der ↑Binder jedem ↑Symbol dieses Programms eine zur Basis 0 relative Adresse zu. Um das Maschinenprogramm zur Ausführung zu bringen, fordert der ↑Lader die ↑Speicherzuteilung an und erhält im Erfolgsfall eine ↑Ladeadresse, die gleichfalls Basisadresse des Segments dieses Programms ist. An diese Ladeadresse wird das Maschinenprogramm in den Hauptspeicher platziert. Für die daraufhin (vom Lader) eingerichtete ↑Prozessinkarnation werden im ↑Prozesskontrollblock Basisadresse und Länge des entsprechenden Segments als Attribute verbucht. Bei einem ↑Prozesswechsel werden diese Attribute in das Basis- und Längenregister geschrieben.

Da dieser Schreibzugriff beim Prozesswechsel als privilegierte Operation im ↑Betriebssystem erfolgt (↑*privileged mode*), müssen im Falle eines durch ein eigenes Segment geschütztes Betriebssystem für den System- und Benutzermodus jeweils eigene Basis-/Längenregister vorhanden sein (↑Arbeitsmodus). Beim Prozesswechsel werden die entsprechenden Segmentattribute (Basis, Länge) in die dem Benutzermodus zugeordneten Basis-/Längenregister geschrieben, die aber erst nach Verlassen des Systemmodus zur Wirkung kommen. Grundsätzlich werden mit jedem Wechsel vom Benutzer- zum Systemmodus und umgekehrt die dem jeweiligen Modus zugeordneten Basis-/Längenregister im Prozessor wirksam. Ist das Betriebssystem dagegen nicht im eigenen Segment isoliert, sind die Basis-/Längenregister nur einfach ausgelegt. In dem Fall ist die Überprüfung der logischen Adresse (*limit check*) im Systemmodus entweder wirkungslos oder abgeschaltet, die Basis-/Längenregister kommen dann nur im Benutzermodus zur Wirkung.

Segmenttabelle Tafel einer ↑MMU, durch die ein ↑segmentierter Adressraum definiert wird. Jeder Eintrag darin ist ein ↑Segmentdeskriptor — mehr dazu aber in SP2.

Segmentwähler (en.) ↑*segment selector*. Spezielles ↑Prozessorregister bei ↑x86, das unter anderem einen ↑Segmentnamen sowie, ab Modell ↑i286, die mit dem betreffenden ↑Segment verbundene Privilegstufe (↑DPL) speichert.

Seite ↑Speicherstück fester Größe, die immer eine Zweierpotenz der Größe von einem ↑Speicherwort ist. Die effektive Größe hängt ab von verschiedenen Faktoren: Umfang der ↑Seitentabelle, verfügbarer Platz im ↑Übersetzungspuffer, ↑interner Verschnitt und der ↑Zugriffszeit auf den ↑Hintergrundspeicher.

Seiten-Kachel-Tabelle ↑Seitentabelle.

Seitenabrufer ↑Programm zur ↑Seitenumlagerung. Intern im ↑Betriebssystem (Systemebene) oder extern dazu oberhalb (Benutzerebene), gegebenenfalls sogar in jedem ↑Maschinenprogramm, angesiedelte Funktion, mit deren Hilfe ↑virtueller Speicher durchgesetzt wird. Abgerufen werden Programmteile aus dem ↑Hintergrundspeicher (Einlagerung) oder dem ↑Vordergrundspeicher (Auslagerung), jeweils im Moment des Zugriffs durch einen ↑Prozess oder vorausschauend. Einheit dafür ist eine ↑Seite, die einzeln oder als Teil einer Folge transferiert wird.

Seitenadressierungseinheit Funktionsbaustein einer ↑MMU, der eine ↑logische Adresse in eine ↑reale Adresse umsetzt. Definitionsbereich der logischen Adresse ist ein ↑seitennummerierter Adressraum. Die Abbildung dieses Adressraums geschieht tabellenbasiert (↑*page table*), der Bildbereich stellt sich als ↑realer Adressraum dar.

Seitendeskriptor Bezeichnung für den ↑Deskriptor einer ↑Seite. Datenstruktur einer ↑MMU (↑*paging unit*), um eine ↑logische Adresse in eine ↑reale Adresse umzusetzen. Der Deskriptor kennzeichnet eine Seite einerseits durch ihren ↑Seitenrahmen und andererseits durch Attribute, die die mit der Seite erlaubten Operationen definieren sowie ihren Systemstatus festhalten. Die erlaubten Operationen haben einen engen Bezug zum ↑Abruf- und Ausführungszyklus der ↑CPU: ausführen (*execute*) erlaubt den Abruf von einem ↑Maschinenbefehl, die Seite enhält ↑Text; lesen (*read*) erlaubt das Auslesen von einem ↑Speicherwort, die Seite enhält Text oder ↑Daten; schreiben (*write*) erlaubt die Veränderung des Inhalts eines Speicherworts, die Seite enhält Daten. Als Statusinformation wird für gewöhnlich festgehalten, ob die Seite benutzt (↑*used bit*) oder beschrieben (↑*dirty bit*) wurde und ob sie auf dem Seitenrahmen liegt (↑*present bit*). Letzteres Statusbit unterstützt die ↑Ladestrategie, es wird vom ↑Betriebssystem verwaltet. Die anderen beiden Bits unterstützen die ↑Ersetzungsstrategie, sie werden von der MMU gesetzt („klebrige Bits“) und vom Betriebssystem gelöscht. Seitenrahmen im realen Adressraum sind durchnummeriert, ebenso wie Seiten im logischen Adressraum. Die Nummer des Seitenrahmens, in dem eine Seite im Hauptspeicher (als Teil des realen Adressraums) „eingespannt“ ist, steht im Deskriptor zu dieser Seite. Diese Nummer multipliziert mit der Seitengröße gibt die Basisadresse der auf den realen Adressraum

abgebildeten Seite. Zu diese Basisadresse wird die Nummer von dem ↑Oktett hinzugefügt, die in den niederwertigen Bits der (abzubildenden) logischen Adresse kodiert ist, um die reale Adresse des an dieser Stelle in der Seite liegenden Objekts zu erhalten (↑Adressabbildung).

Seitenfehler Art von Zugriffsfehler. Grundlage bildet ↑seitennummerierter virtueller Speicher. Im Moment des Zugriffs auf ein ↑Speicherwort im virtuellen Speicher stellt der ↑Prozessor fest, dass die ↑Seite, die dieses Wort umfasst, nicht im ↑Hauptspeicher eingelagert ist. Der Prozessor stellt daraufhin eine ↑Ausnahmesituation fest, die vom Betriebssystem (↑Seitenabrufer) behandelt wird und zur ↑Seitenumlagerung führt: die Seite, auf der das referenzierte Wort steht, wird eingelagert.

Seitenflattern Phänomen andauernder ↑Seitenumlagerung. Verursacht ein ↑Prozess einen ↑Seitenfehler, ist in dem Moment der ↑Hauptspeicher komplett belegt, nämlich kein ↑Seitenrahmen zur Zeit frei, und soll der Prozess für seinen Fortschritt nicht von der Seitenrahmenrückgabe anderer Prozesse abhängig sein, muss das ↑Betriebssystem zur Einlagerung der angeforderten ↑Seite Platz durch Auslagerung einer Seite (desselben oder eines anderen Prozesses) schaffen: es sorgt für die ↑Verdrängung einer Seite aus ihrem Seitenrahmen. Die eben erst verdrängte/ausgelagerte Seite wird (ggf. nach erfolgtem ↑Prozesswechsel) jedoch sofort wieder von ihrem Prozess referenziert, woraufhin erneut ein Seitenfehler auftritt. Ist der Hauptspeicher in dem Moment immer noch komplett belegt, wiederholt sich der Vorgang der Seitenverdrängung: eine eingelagerte Seite (desselben oder eines anderen Prozesses) wird ausgelagert, um die unlängst verdrängte Seite wieder einzulagern, die dann wiederum verdrängt wird, um erneut ausgelagert zu werden und so weiter. Die verdrängte Seite flattert ständig zwischen ↑Vordergrundspeicher und ↑Hintergrundspeicher hin und her, solange kein einziger Seitenrahmen frei ist. Der Prozess der flatternden Seite kommt nur noch extrem langsam voran, da das Betriebssystem viel mehr Zeit mit Ein-/Ausgabe für die diesen Prozess betreffende und andauernde Seitenumlagerung beansprucht. Erst wenn (ggf. unbeteiligte) Prozesse Seitenrahmen an das Betriebssystem zurückgeben, kann das Flattern einer solcher Seite ein Ende finden und der Prozess wieder mit voller Leistung voranschreiten. Das Phänomen kann auftreten, sobald nur eine einzige ↑Betriebsseite eines Prozesses ausgelagert und damit seine ↑Arbeitsmenge auseinandergerissen wird. Es betrifft zumeist auch mehr als einen Prozess und sorgt für starken Leistungseinbruch im gesamten ↑Rechensystem.

seitennummerierte Segmentierung Art der ↑Segmentierung von einem ↑Adressraum, in dem ein ↑Segment als ↑seitennummerierter Adressbereich aufgestellt ist. Ein ↑Segmentdeskriptor beschreibt dabei die für das Segment benötigte ↑Seitentabelle (insb. Anfangsadresse und Länge). Je nach ↑MMU wird die globale ↑Segmenttabelle ebenfalls als Segment erfasst (↑Wurzelsegment). Kommt zusätzlich ↑seitennummerierter virtueller Speicher zum Einsatz, können damit für sehr große Segmente wie auch Segment- beziehungsweise Seitentabellen nur die jeweils benötigten Abschnitte im ↑Hauptspeicher gehalten werden — mehr aber dazu in SP2.

seitennummerierter Adressbereich ↑Adressbereich, dessen Strukturelement eine ↑Seite bildet. Die Größe dieses Bereichs ist ganzzahlige Vielfache der Größe einer Seite, wobei die Seitengröße durch die zugrundeliegende ↑MMU vorgegeben ist.

seitennummerierter Adressraum Bezeichnung für einen ↑Adressraum, der eindimensional ausgelegt ist: die eine Dimension benennt das ↑Speicherwort innerhalb dieses Adressraums. Wesentliches Strukturelement dieses Adressraums ist jedoch die ↑Seite, was bedeutet, dass der komplette Adressraum als einzelner ↑seitennummerierter Adressbereich erscheint. Die innere Gliederung dieses Adressraums ist durch die von einer ↑MMU vorgegebene Seitengröße bestimmt.

seitennummerierter virtueller Speicher Bezeichnung für einen als ↑virtueller Speicher implementierten ↑Arbeitsspeicher, dessen Strukturelement und kleinste Verwaltungseinheit die ↑Seite darstellt. Der gesamte Speicherbereich ist linear in Form solcher Seiten organisiert.

Seitenrahmen Abschnitt fester Größe im ↑Hauptspeicher (↑realer Adressraum) zur Aufnahme von exakt einer ↑Seite: die effektive Rahmengröße ist definiert durch die Seitengröße. Die Nummer eines solchen Rahmens bildet die ↑reale Adresse einer für gewöhnlich nur durch eine ↑logische Adresse oder ↑virtuelle Adresse erreichbaren Seite. Im Hintergrund steht ein ↑seitennummerierter Adressraum, dessen einzelne Seiten auf korrespondierende Seitenrahmen abzubilden sind. Die hierfür nötige Adressumsetzung leistet eine ↑MMU, die dazu vom ↑Betriebssystem vorher entsprechend programmiert werden muss (↑Seitentabelle).

Seitentabelle Tafel einer ↑MMU, durch die ein ↑seitennummerierter Adressraum definiert wird. Jeder Eintrag darin ist ein ↑Seitendeskriptor. Die Tafel liegt im ↑Arbeitsspeicher, ihre (logische/virtuelle) ↑Adresse darin bestimmt in aller Regel heute (2016) das ↑Betriebssystem und wird in einem Adressregister der MMU vermerkt. Für die Festlegung der Tafelgröße gibt es je nach MMU einen statischen oder dynamischen Ansatz. Im statischen Fall ist die Tafelgröße bestimmt durch die von der MMU vorgegebenen Größe einer ↑Seite einerseits und der ↑Adressbreite der CPU andererseits, woraus sich pro Adressraum eine Seitenanzahl 2^P wie folgt ableitet: $2^P = 2^{A-O}$, mit A gleich der Adressbreite und $O = log_2(sizeof(page))$ gleich der Bitanzahl zur Kodierung der Nummer $o = [0, 2^O - 1]$ von einem ↑Oktett innerhalb der Seite $p = [0, 2^P - 1]$. Diese innere Gliederung der Adresse eines seitennummerierten Adressraums mit einem ↑Adressbereich $[0, 2^A - 1]$ gibt die MMU vor, um aus einer solchen Adresse die Seitennummer p als Indexwert zu extrahieren und darüber für die Adressabbildung auf den der Seite p zugeordneten Deskriptor der Tafel/Tabelle zuzugreifen. Da in diesem Fall die MMU keine Überprüfung des Indexwerts vornimmt, muss die Tabelle mit 2^P Deskriptoren gefüllt sein — wobei aber nicht jeder Deskriptor eine gültige ↑Adressabbildung beschreibt. Um für große Werte von P den Speicherbedarf für die Seitentabelle in annehmbaren Größen zu halten, unterstützt die MMU typischerweise eine mehrstufige Abbildung (x86). Dafür wird die Seitennummer p weiter in gleich große Bitleisten untergliedert, wobei jede Leiste einen kleineren Indexwert für eine Seitentabelle einer bestimmten Stufe repräsentiert. Im dynamischen Fall implementiert die MMU, zusätzlich zum Adressregister, noch ein Grenzregister, dessen Inhalt den letzten gültigen Tabelleneintrag definiert. In logischer Hinsicht bildet die Seitentabelle damit ein ↑Segment, für das ein ↑seitennummerierter Adressbereich definiert ist. Unterstützt die MMU ↑segmentierte Seitenadressierung, könnte darüber die Seitentabelle für jeden einzelnen ↑Prozessadressraum entsprechend eingegrenzt werden: Adress- und Grenzregister wären dann Attribute von einem ↑Segmentdeskriptor, der Lage und Größe der Seitentabelle beschreibt.

Seitenumlagerung Funktion in einem ↑Betriebssystem, durch die eine bei der Ausführung von einem ↑Programm referenzierte, aber nicht im ↑Hauptspeicher (Vordergrund) vorliegende ↑Seite vom ↑Hintergrundspeicher automatisch eingelagert wird. Gegebenenfalls wird dazu, wenn nämlich noch Platz für die Einlagerung zu schaffen ist, eine im Hauptspeicher vorliegende Seite vorher auf den Hintergrundspeicher ausgelagert. Grundlage dafür bildet ↑seitennummerierter virtueller Speicher. Für den anfallenden Transfer aller betroffenen Seiten im Vorder- und Hintergrundspeicher sorgt das Betriebssystem.

Seitenvorabruf ↑Seitenumlagerung, die vorausschauend funktioniert und nicht erst im Moment des Zugriffs auf eine ↑Speicherstelle. Das ↑Betriebssystem hat exaktes oder heuristisches Wissen darüber, welche ↑Seite als nächste bei der Ausführung von einem ↑Programm referenziert wird.

Sekundärspeicher Klassifikation für den ↑Ablagespeicher; „zweitrangiger Speicher“, der zwar über eine große Kapazität verfügt, dafür jedoch auch eine hohe ↑Zugriffszeit mit sich bringt.

Selbstvirtualisierung Form der ↑Virtualisierung, die die eigene (reale) Maschine betrifft. Dabei stellt die jeweils bereitgestellte ↑virtuelle Maschine ein vollständiges Abbild der realen Maschine dar, dass heißt, das ↑Programmiermodell der virtuellen Maschine ist mit dem der realen Maschine identisch. Für einen im ↑Maschinenprogramm residierenden ↑Prozess

ist es damit auch nicht feststellbar, ob er auf einer realen oder virtuellen Maschine stattfindet. Typische Ausprägungen dieser Virtualisierungsart sind die ↑Vollvirtualisierung und ↑Paravirtualisierung.

semantische Lücke Bedeutungsbezogener Unterschied zwischen den Beschreibungen von zwei Sprachebenen in einem mehrschichtig organisierten ↑Rechensystem. Ursprünglich begründet in der Diskrepanz zwischen den komplexen Operationen der Konstrukte einer höherer Programmiersprache und den einfachen Operationen (↑*instruction set*) einer ↑CPU.

Semaphor Mittel zur ↑Koordination unterschiedlicher Art (↑binärer Semaphor, ↑allgemeiner Semaphor); ein spezieller Datentyp zum Zwecke der ↑Synchronisation, auf dem im Wesentlichen die beiden Operationen ↑P und ↑V definiert sind. Mit P synchronisiert sich ein ↑Prozess auf ein bestimmtes ↑Ereignis, dessen Auftreten durch V entweder von ihm selbst (↑unilaterale Synchronisation) oder von einem anderen (uni- und ↑multilaterale Synchronisation) Prozess angezeigt wird. Da beide Operationen insbesondere auch von verschiedenen Prozessen benutzt werden, müssen sie einer ↑Nebenläufigkeitssteuerung unterliegen: P und V gehören zur Klasse der ↑Elementaroperation, deren Durchführung jedoch auch bei ↑Parallelverarbeitung ein konsistentes Ergebnis liefern muss — das bedeutet allerdings nicht, sie jeweils als ↑atomare Operation auslegen zu müssen.
Zentrale Bedeutung für den Fortschritt von einem Prozess in dem Zusammenhang hat die jeweils in P und V formulierte ↑Ablaufsteuerung. Diese kann einen Prozess in P blockieren lassen, bis er durch ein V wieder deblockiert wird; sie stellt sich für einen binären Semaphor etwas anders dar als für einen allgemeinen Semaphor. Beiden gemeinsam ist aber, dass die in P blockierten Prozesse eine ↑Warteschlange bilden, die durch V abgebaut wird. Diese Reihe von wartenden Prozessen ist entweder als dynamische Datenstruktur ein Attribut des Semaphordatentyps, damit pro ↑Exemplar vorhanden und wird auch nach eigenen Kriterien verwaltet, oder sie wird im Moment der (durch V veranlassten) Deblockierung eines Prozesses durch den ↑Planer nach seinen Kriterien berechnet. Erstere Variante ist anfällig für ↑Interferenz mit dem Planer, da die Bedienungsverfahren der Warteschlange und der ↑Bereitliste in aller Regel verschieden sind. Letztere Variante ist anfällig für Varianzen in der zur Deblockierung anfallenden ↑Latenzzeit, da die ↑Prozesstabelle erst nach Einträgen abgesucht werden muss, die mit bestimmten Exemplaren des Semaphordatentyps verknüpft sind.
Wichtiges Merkmal — weder Fehler noch Unzulänglichkeit, wie gelegentlich kolportiert — beider Operationen (P und V) darüberhinaus ist, dass sie von jedem Prozess gleichberechtigt genutzt werden können. Anderenfalls gestaltet sich die Koordinierung von Prozessen in nicht wenigen Fällen schwierig (z.B. Hoare-/Hansen-Typ von ↑Monitor) oder gar unmöglich (z.B. Erzeuger-Verbraucher-Beziehung, Planer als ↑kritischer Abschnitt). Gleichwohl haben sich spezielle Semaphorvarianten herausgebildet, wie etwa ↑privater Semaphor und ↑Mutex, durch die bestimmte Muster der Synchronisation unterstützt werden und die helfen, Programmierfehler zu vermeiden oder zu erkennen.

semaphore (dt.) ↑Semaphor; Winker, Signalmast, Formsignal; (gr.-nlat.) Zeichenträger.

sensitiver Befehl Bezeichnung für einen ↑Maschinenbefehl, dessen direkte Ausführung durch eine ↑virtuelle Maschine nicht tolerierbar ist. Ein solcher Befehl gilt als *störungsempfindlich*, wenn er den Zustand der ↑Systemsteuerung, reservierter ↑Prozessorregister oder einer reservierten ↑Speicherstelle ändert/abfragt, das ↑Schutzsystem für Programme, die privilegiert ablaufen müssen, referenziert oder Ein-/Ausgabe tätigt.

Separator Vorrichtung, um einzelne Bestandteile in der mehrwortig ausgeführten Bezeichnung einer ↑Entität zu trennen. Ein spezieller Trenntext, der einzelne oder eine Gruppe von Worten separiert. Dabei wird der jeweils separierte Abschnitt als ↑Name aufgefasst, der sich auf einen bestimmten (vor dem Trenntext bezeichneten) ↑Namenskontext bezieht. Beispiele solcher Trenntexte sind: `>` (*greater-than*, ↑Multics), `/` (*slash*, ↑UNIX) oder \ (*backslash*, ↑Windows) — in der Reihenfolge ihrer historischen Entwicklung.

Sequentialisierung Schaffung einer bestimmten Reihung in einer ↑Aktionsfolge entlang einer ↑Kausalordnung.

sequentielle Konsistenz Modell der ↑Speicherkonsistenz, nach dem jeder ↑Prozessor jede Operation auf den ↑Arbeitsspeicher immer in der durch das (nichtsequentielle) ↑Programm spezifizierten Reihenfolge durchführt.

sequentielles Programm Bezeichnung für ein ↑Programm, das ausschließlich Konstrukte zur Formulierung sequentieller Abläufe verwendet. Diese Konstrukte sind in der Programmiersprache enthalten, in der das Programm formuliert ist. Jedoch ist zu beachten, dass ein und derselbe Programmablauf auf einer Abstraktionsebene (höhere) sequentiell und einer anderen (tiefere) parallel sein kann: nämlich wenn auf höherer Ebene ein ↑Unterprogramm einer tieferen Ebene aufgerufen wird und das Unterprogramm ein ↑nichtsequentielles Programm darstellt.

service time (dt.) ↑Bedienzeit.

session (dt.) ↑Sitzung.

shared code (dt.) gemeinsam genutztes ↑Programm oder ↑Unterprogramm. ↑Text und möglicherweise auch ↑Daten liegen in einem von mehr als einen ↑Prozess zugleich zugreifbaren Bereich im ↑Arbeitsspeicher (↑*shared memory*). Gilt für diesen Programmbereich keine ↑Ablaufinvarianz oder besteht unter den Prozessen zwar eine Kausalordnung, deren Erhaltung durch die ↑Prozesseinplanung jedoch nicht zugesichert werden kann, ist zur Vorbeugung eventueller Inkonsistenzen ↑Synchronisation unter den Prozessen zu erzielen.

shared data (dt.) gemeinsam genutzte ↑Daten. Die Daten liegen in wenigstens einem ↑Speicherwort, auf das von mehr als einem ↑Prozess zugleich zugegriffen werden kann (↑*shared memory*). Um im Falle der gleichzeitig möglichen Schreibzugriffe (Lesen-Schreiben oder Schreiben-Schreiben) einem inkonsistenten Datenbestand vorzubeugen, ist ↑Synchronisation unter den Prozessen zu erzielen.

shared library (dt.) ↑Gemeinschaftsbibliothek.

shared memory (dt.) ↑gemeinsamer Speicher.

shared-memory processor (dt.) ↑speichergekoppelter Multiprozessor.

shell (dt.) Außenhaut, Randzone, Ummantelung. Mit ↑Multics (um 1964) eingeführte Bezeichnung für den über eine ↑Dialogstation genutzten ↑Kommandointerpreter zur Interaktion mit dem ↑Betriebssystem. Vorreiter dafür war `RUNCOM` (um 1963) von ↑CTSS, ein ↑Programm zur Verarbeitung von in ↑Skriptsprache formulierter Kommandos nebst Parametersubstitution.

short-term scheduling (dt.) ↑kurzfristige Einplanung.

Signal Bezeichnung für ein Zeichen mit einer bestimmten Bedeutung, der Träger einer Information (in Anlehnung an den Duden). Gemeinhin erklärt auch als ↑Nachricht mit „Nulllänge“, die jedoch nicht inhaltslos ist.

Simultanverarbeitung Fähigkeit von einem ↑Betriebssystem zur ↑Parallelverarbeitung im ↑Mehrprogrammbetrieb. Manifestiert sich vor allem in speziellen Verfahren zur ↑Prozesseinplanung und durch ↑partielle Virtualisierung der ↑CPU.

single-stream batch monitor (dt.) ↑Einzelstromstapelmonitor.

single-user mode (dt.) ↑Einbenutzerbetrieb.

Sitzung Abschnitt im Zeitverlauf zwischen ↑Anmeldung und ↑Abmeldung, in dem das ↑Rechensystem bestimmte Anforderungen für eine Person oder für einen externen ↑Prozess nachgeht. Eine solche Anforderung kann einen einzelnen ↑Auftrag, einen ↑Dienst oder eine beliebige Abfolge von beiden beinhalten.

SJF Abkürzung für (en.) *shortest job first*. Strategie zur ↑Planung der Bearbeitung von ↑Aufgaben durch ein ↑Rechensystem in einem *Prioritätsverfahren*, das die Aufgaben nach der Bearbeitungszeit sortiert und dann der Aufgabe, deren Bearbeitung vermeintlich am schnellsten geschehen kann, den Vorzug gibt: je kürzer die Bearbeitungsdauer einer Aufgabe, desto höher ihre ↑Priorität, wobei der Prioritätswert einen Zeitwert darstellt. Dabei entspricht die Aufgabe einem ↑Auftrag (*job*).
In dieser Grundidee stimmt das Verfahren mit ↑SPN überein — jedoch ist das der Planung zugrunde liegende Kriterium nicht die Dauer eines jeweiligen ↑Rechenstoßes eines ↑Prozesses, sondern ein kompletter ↑Stapelauftrag. Der entscheidende Punkt liegt darin, dass die Aufgaben für gewöhnlich nicht interaktiv bearbeitet werden (↑*batch mode*). Dabei bildet dann jede dieser Aufgaben einen ↑Stapel von ↑Teilaufgaben, die dem ↑Rechensystem im Ganzen zur Bearbeitung übergeben werden (↑*batch job*). Das Planungskriterium ist die für eine Aufgabe zu veranschlagende totale ↑Laufzeit. Darin enthalten sind alle ↑Wartezeiten, die ein zur Bewältigung der Aufgabe angesetzter Prozess (gegebenenfalls in Kooperation mit anderen Prozessen) erfahren wird.
Ein weiterer wichtiger Unterschied zu SPN ist die ↑vorlaufende Planung. Die dafür im Voraus bekannt zu gebenden Bearbeitungszeiten der Aufgaben werden damit nicht im normalen Betrieb ermittelt. Die Bestimmung einer solchen Bearbeitungszeit kann beispielsweise durch die in ↑Kommandosprache formulierte explizite Angabe einer ↑Frist (*time limit*) im ↑Stapelauftrag erfolgen, deren Quantifizierung vielleicht nur auf Erfahrungswissen zurückgreift. Ein anderer Ansatz wäre ein *Probelauf* des Stapelauftrags unter realistischen Bedingungen, das heißt, die Durchführung eines ↑Produktionsbetriebs. Auf Basis der so ermittelten Laufzeitparameter kann durch eine anschließende ↑Zeitreihenanalyse die voraussichtlich optimale Reihenfolge der Aufgabenbearbeitung bestimmt werden.

SJN Abkürzung für (en.) *shortest job next*. Synonym zu ↑SJF.

Skriptsprache Programmiersprache zur Formulierung von einem zumeist kurzen und kompakten ↑Programm, das für gewöhnlich direkt durch einen ↑Interpreter zur Ausführung kommt (↑CSIM). Ein Hauptaspekt einer solchen Sprache liegt in der formalen Beschreibung von Anweisungsfolgen, um auf bestehende und in dem jeweils gegebenen ↑Rechensystem verfügbare Programme zugreifen und gegebenenfalls auch zu einen umfassenden Programmkomplex verknüpfen zu können. Ein weiterer, häufiger Einsatz besteht im Musterbau (*prototyping*) insbesondere von Anwendungsprogrammen.

sleep state (dt.) ↑Schlafzustand.

SLIH Abkürzung für (en.) *second-level interrupt handler*. Bezeichnung für den ↑Unterbrechungshandhaber zweiter Stufe. Dieser ↑Handhaber wird indirekt durch eine ↑Unterbrechungsanforderung gestartet und beginnt seine Ausführung auf der durch das ↑Betriebssystem bestimmten ↑Unterbrechungsprioritätsebene. Für gewöhnlich ist dies die Prioritätsebene 0, das heißt, alle Unterbrechungsanforderungen sind zugelassen (*interrupts enabled*). Da ein solcher Handhaber als ↑Fortsetzung der entsprechenden ↑Unterbrechungsbehandlung erster Ebene (↑FLIH) fungiert und letztere durch die Hardware definiert auf Prioritätsebene l_{flih}, $l_{flih} \geq 0$ stattfand, ist die ihm zugebilligte Prioritätsebene des Betriebssystems l_{slih}, $0 \leq l_{slih} \leq l_{flih}$ explizit einzunehmen. Dies bedeutet zweierlei:

1. Sicherstellen, dass sämtliche Unterbrechungsbehandlungen der ersten Ebene abgeschlossen sind, das heißt, der durch solche Behandlungen aufgebaute ↑Laufzeitstapel wieder abgebaut wurde. Hier sind insbesondere ↑kaskadierte Unterbrechungen in Erwägung zu ziehen. Da die Behandlung von einem ↑IRQ oder einem ↑NMI jederzeit durch einen

(weiteren) NMI unterbrochen werden kann, ist auch im Falle einer ↑CPU, die nur über einen einstufigen IRQ verfügt (wie im Falle von ↑x86), gegebenenfalls eine mögliche Kaskadierung zu berücksichtigen.

2. Falls nicht bereits geschehen, die CPU oder den ↑PIC dazu veranlassen, nur die Unterbrechungsanforderungen ab Prioritätsebene l_{slih}, normalerweise $l_{slih} = 0$, zuzulassen. Da diese ↑Aktion aus einem Handhaber erster Stufe heraus erfolgt, muss dieser für die Quittierung der von ihm behandelten Unterbrechungsanforderung gesorgt haben. Anderenfalls besteht (vor allem bei ↑Pegelsteuerung) die Gefahr des sofortigen ↑Wiedereintritts in dieselbe Unterbrechungsbehandlung, mit der möglichen Folge einer unkontrollierbaren und gegebenenfalls ↑Panik verursachenden ↑Rekursion.

Diese beiden Schritte bewerkstelligt ein spezieller ↑Ausnahmezuteiler, der damit letztlich eine kontrollierte ↑Sequentialisierung von Handhaberausführungen betreibt. Jede dieser Ausführungen wird allerdings erst veranlasst, wenn von der durch eine ↑Unterbrechung betretenen ↑Systemebene zurück zur ↑Benutzerebene gewechselt werden sollte (↑AST).
Die durch den Handhaber erfolgende ↑Unterbrechungsbehandlung geschieht asynchron zum auf Benutzerebene unterbrochenen ↑Prozess. Je nach ↑Operationsprinzip des Betriebssystems und der damit verbundenen Art der ↑Synchronisation von Prozessen auf Systemebene, läuft die Unterbrechungsbehandlung der zweiten Ebene ebenso wie die der ersten Ebene asynchron ab oder sie erfolgt, anders als die der ersten Ebene, synchron (↑Fortsetzungssperre, ↑nichtblockierende Synchronisation) zu bestimmten ↑Aktionen im Betriebssystem. In jedem Fall geht seine Ausführung immer im Namen des gerade stattfindenden, aber unterbrochenen Prozesses vor. Da dem Handhaber damit kein eigenständiger und eindeutiger Ablaufkontext (d.h., keine eigene ↑Prozessinkarnation) zur Verfügung steht, dürfen im Zuge seiner Ausführung weder direkt noch indirekt Operationen zur Wirkung kommen, die den unterbrochenen Prozess in die ↑blockierende Synchronisation zwingen. Sehr wohl aber kann letzterer für die ↑unilaterale Synchronisation in die Rolle als *Erzeugerprozess* gebracht werden, um nämlich das ↑Ereignis herbeizuführen, auf dessen Eintritt ein anderer Prozess gegebenenfalls wartet: ein signalisierender, ↑allgemeiner Semaphor kann benutzt werden, solange für die Operation dazu (↑V) nirgends ein ↑kritischer Abschnitt passiert werden muss.
Der Wirkungskreis des Handhabers ist nur bedingt eingeschränkt: er erstreckt sich sowohl auf das die Unterbrechung verursachende Gerät der ↑Peripherie als auch auf das Betriebssystem. Die ↑Aufgabe des Handhabers besteht in erster Linie darin, gepufferte ↑Daten zwischen Prozess und Gerät zu transferieren (↑Ein-/Ausgabe), den Prozessen die von den Geräten verursachten ↑Ereignisse mitzuteilen, Prozesse dadurch gegebenenfalls bereitzustellen und von den Prozessen Aufträge zum Absetzen eines ↑Ein-/Ausgabestoßes entgegenzunehmen. Vor allem aber ist sein Zweck, ↑Unterbrechungslatenz zu minimieren.

SMP Abkürzung für (en.) ↑*symmetric multiprocessing*, (en.) ↑*symmetric multiprocessor* und (en.) ↑*shared-memory processor* — wobei letzterer einen ↑Multiprozessor oder ↑Mehrkernprozessor bezeichnet, der nicht zwingend symmetrisch aufgebaut sein muss, sondern insbesondere auch eine asymmetrische Auslegung haben kann (↑AMP).

Softwareschicht Sammlung von ↑Text und ↑Daten (zusammengefasst als ein ↑Programm oder eine Menge davon) auf einer bestimmten Ebene in einem System, das eine ↑hierarchische Struktur aufzeigt. Dabei müssen nicht alle Ebenen dieses Systems in Software vorliegen, das Gesamtsystem kann einen Komplex aus Soft-, Firm- und Hardware bilden. Programme, die in einer Schicht zusammengefasst sind, teilen sich nicht zwingend dasselbe Wissen über die dortigen Datenstrukturen, ↑Modul und Schicht sind zwei voneinander unabhängige Konzepte: Eine Schicht kann ein oder mehrere Module enthalten, ein Modul kann sich über eine oder mehrere Schichten erstrecken. So definiert vor allem die umfassende hierarchische Struktur, welche Programme zusammen eine Schicht ausmachen.

source module (dt.) ↑Quellmodul.

SP Abkürzung für (en.) ↑*stack pointer*; Kürzel der Lehrveranstaltung ↑Systemprogrammierung.

special-purpose operating system (dt.) ↑Spezialbetriebssystem.

Speicher Vorrichtung in einem ↑Rechensystem zur Aufbewahrung von Informationen auf einem ↑Speichermedium.

speicherabgebildete Ein-/Ausgabe Art von Ein-/Ausgabe, bei der die ↑Ein-/Ausgaberegister von einem ↑Peripheriegerät über eine normale ↑Adresse zugänglich sind. Jeder ↑Maschinenbefehl, der in Abhängigkeit von der ↑Adressierungsart wenigstens einen eine ↑Speicherstelle adressierenden Operanden hat, ist zur Durchführung von Ein-/Ausgabevorgängen geeignet: Eingabe entspricht Lesen und Ausgabe entspricht Schreiben, jeweils in Bezug auf eine Speicherstelle.

Speicherbandbreite Maß der Transferrate von ↑Daten zwischen ↑Hauptspeicher und ↑CPU. Im Allgemeinen das Produkt aus ↑Wortbreite und ↑Taktfrequenz.

Speicherbarriere ↑Maschinenbefehl, der eine bestimmte Ordnungsbedingung für die Speicheroperationen einer ↑CPU durchsetzt. Für gewöhnlich besagt diese Bedingung, dass Operationen, die vor der Barriere ausgegeben wurden, garantiert ausgeführt werden vor Operationen, die nach der Barriere ausgegeben werden. Beispiele eines solchen Maschinenbefehls sind `sync` (↑PowerPC) und `mfence` (↑x86).

Speicherbereich Abschnitt bestimmter Länge im ↑Arbeitsspeicher. Der Bereich kann statisch oder dynamisch angelegt worden sein, also vor oder zur ↑Laufzeit der diesen Bereich benutzenden Entität (↑Prozessinkarnation). Im statischen Fall bestimmen Programmierpersonal, ↑Kompilierer, ↑Assemblierer, ↑Binder oder ↑Lader (d.h., ↑Betriebssystem) den Bereich. Dabei ist jeder Verfahrensschritt in der Lage Lokalität oder ↑Adresse des Bereichs vorzugeben, wohingegen nur die ersten drei genannten die Bereichslänge festlegen. Die Länge ist typischerweise ganzahlige Vielfache der Größe von einem ↑Speicherwort. Im dynamischen Fall bestimmen ein ↑Prozess selbst oder das Betriebssystem den Bereich, wobei der Prozess in aller Regel nur die Länge vorgibt und das Betriebssystem die (reale, logische, virtuelle) ↑Adresse für einen mindestens so großen Abschnitt im Rahmen der ↑Speicherzuteilung ermittelt und dem Prozess zuweist.

Speicherbestückung Ausstattung von ↑RAM oder ↑ROM (inkl. der verschiedenen Arten davon) in einem ↑Rechner (↑*main board*, ↑*motherboard*), wobei der Rechnerhersteller Art und Umfang der jeweiligen Ausrüstung festlegt. Grundlage für diese Festlegung bildet dabei immer auch ein (durch die ↑CPU definierter) ↑realer Adressraum, indem nämlich jedem dieser Speicherbausteine ein bestimmter ↑Adressbereich zugewiesen wird. Jede in diesen Adressbereichen fallende ↑reale Adresse, die zur/bei Ausführung von einem ↑Maschinenbefehl von der CPU auf den ↑Adressbus ausgegeben wird, selektiert einen vorhandenen Speicherbaustein. Reale Adressen, die außerhalb dieser Bereiche liegen, bedeuten einen schwerwiegenden Fehler (↑*bus error*) und werden abgefangen (↑*trap*).

Speicherdirektzugriff Methode des Transfers von ↑Daten zwischen ↑Peripheriegerät und ↑Hauptspeicher ohne Hilfestellung durch die ↑CPU. Im Gegensatz zu ↑PIO beauftragt die CPU eine spezielle Steuereinheit (↑DMA *controller*), um die Daten transferiert zu bekommen. Die damit verbundene Ein-/Ausgabe in Bezug auf einen bestimmten ↑Speicherbereich läuft ohne Kontrolle der CPU ab. Folglich kann die CPU in der Zeit andere Berechnungen durchführen: Ein-/Ausgabe und Berechnungen überlappen sich. Zur Durchführung des Datentransfers durch die Steuereinheit sind verschiedene Arbeitsweisen gebräuchlich: ↑Stoßbetrieb, ↑Taktentzug, ↑Transparentbetrieb. Je nach Steuereinheit ist eine einzelne Transfereinheit das ↑Byte (d.h., ↑Oktett), ↑Wort (32-Bit) oder Halbwort (16-Bit). Darüberhinaus kann der Zugriff der Steuereinheit auf den Hauptspeicher über eine ↑reale Adresse, ↑logische Adresse oder ↑virtuelle Adresse erfolgen. Erwartet die Steuereinheit eine reale Adresse, muss diese das ↑Betriebssystem vor Absetzen des Transferauftrags gegebenenfalls zunächst aus einer vorgegebenen logischen/virtuellen Adresse herleiten, nämlich wenn ein ↑logischer Adressraum

oder ↑virtueller Adressraum Quelle oder Ziel des Transfervorgangs ist. Kann die Steuereinheit jedoch mit einer logischen/virtuellen Adresse umgehen, fordert sie selbst bei der ↑MMU die Umwandlung in die korrespondierende reale Adresse an. In beiden Fällen muss aber das Betriebssystem dafür sorgen, dass der Speicherbereich für den Datentransfer reserviert ist, das heißt, der Bereich muss zeitweilig von ↑Überlagerung oder ↑Umlagerung beziehungsweise ↑Seitenumlagerung ausgenommen sein. Das Ende des Transfers wird dem Betriebssystem für gewöhnlich durch eine ↑Unterbrechungsanforderung mitgeteilt.

speichergekoppelter Multiprozessor ↑Parallelrechner, dessen Prozessoren einen gemeinsamen ↑Hauptspeicher besitzen und diesen mitbenutzen können (↑*shared memory*). Je nach Anzahl und Art der gekoppelten ↑Rechner, jeder davon gegebenenfalls auch nur ein einzelner ↑Rechenkern, ist der Hauptspeicher mehr oder weniger gleichförmig ausgelegt und in zeitlicher Hinsicht einheitlich zugänglich (*uniform*). Für gewöhnlich ist bei größeren Systemen der Hauptspeicher nur noch mit unterschiedlicher ↑Latenz zugänglich, gerade auch bezogen auf die Distanz der Strecke „hinter“ dem ↑Zwischenspeicher: nicht jeder ↑Prozessor ist gleich weit entfernt von einer gegebenenfalls gemeinsam und zugleich adressierten ↑Speicherzelle. Damit wird bei einem Fehlzugriff auf den Zwischenspeicher (↑*cache miss*) die Einlagerung der entsprechenden ↑Zwischenspeicherzeile unterschiedlich lange dauern können. So ist wenigstens die ↑Zugriffszeit auf dieselbe Speicherzelle von der Lokalität des Kerns abhängig, von der aus ein ↑Prozess den Zugriff ausübt (*non-uniform*). Darüber hinaus ist, in Abhängigkeit von der Konstruktionskomplexität des jeweiligen Systems, nicht zwingend sichergestellt, dass der gleichzeitige Zugriff auf dieselbe Speicherzelle von verschiedenen Lokalitäten aus auch immer für alle betreffenden Prozesse denselben Wert liefert (↑*cache coherence*). Auch wenn in solch einem ↑Rechensystem den Prozessen ein einheitlicher ↑Adressraum geboten wird, um direkt auf den gemeinsamen Hauptspeicher zugreifen zu können, bedeutet dies längst nicht, dass damit ein ↑nichtsequentielles Programm leicht von der Hand geht.

Speichergrundfläche (en.) ↑*memory footprint*. Bezeichnung für eine ↑Prozessgröße, die die Ausdehnung von einem ↑Prozess, genauer der entsprechenden ↑Prozessinkarnation, im ↑Hauptspeicher oder im ↑Arbeitsspeicher quantifiziert. Der Unterschied zwischen den beiden Betrachtungsweisen liegt in der Art des als Bezugssystem genommenen ↑Prozessadressraums, das heißt, einerseits ↑realer Adressraum beziehungsweise ↑logischer Adressraum oder andererseits ↑virtueller Adressraum. Ersterer Fall definiert eine bestimmte Grundfläche nur im ↑Vordergrundspeicher, wohingegen letzterer Fall diese Fläche um Bereiche im ↑Hintergrundspeicher erweitert. In beiden Fällen umfasst die Ausdehnungsfläche jedoch jeden einzelnen ↑Speicherbereich, der dem Prozess zu einem Zeitpunkt statisch oder dynamisch zugeordnet ist. Gemeinhin bezieht sich diese Prozessgröße aber auf den Hauptspeicher.
Neben der einer Prozessinkarnation jeweils zugeschriebenen Grundflächengröße (im Hauptspeicher) ist gerade auch die des ↑Betriebssystems ein bedeutender Faktor, um nämlich die ↑Gemeinkosten (in Bezug auf Hauptspeicher) für eine bestimmte ↑Betriebsart zu quantifizieren. Für ein ↑Spezialbetriebssystem können diese Kosten weniger als ein Kilobyte betragen, wohingegen ein ↑Universalbetriebssystem leicht mehrere Megabytes mit Beschlag belegt.

Speicherhierarchie Gesamtheit der in einer bestimmten Rangordnung stehenden ↑Speicher in einem ↑Rechensystem. Die ↑hierarchische Struktur ist durch eine Relation zwischen Speicherpaaren definiert, die von oben nach unten eine Zunahme an Speicherkapazität und, als Konsequenz daraus, ↑Zugriffszeit beschreibt. Typisch ist eine sechsstufige Hierarchie von ↑Registerspeicher, ↑Zwischenspeicher, ↑Notizblockspeicher, ↑Hauptspeicher/↑Arbeitsspeicher, ↑Ablagespeicher und ↑Archivspeicher. Die obersten vier Stufen (Register- bis Haupt-/Arbeitsspeicher) umfassen den ↑Primärspeicher, auch ↑Vordergrundspeicher. Demgegenüber bilden Ablagespeicher den ↑Sekundärspeicher und Archivspeicher den ↑Tertiärspeicher, zusammen auch der ↑Hintergrundspeicher. In dieser Anordnung erstreckt sich ↑virtueller Speicher über den gesamten Haupt-/Arbeitsspeicher und einem Teil vom Ablagespeicher (↑*swap area*). Das ↑Betriebssystem verwaltet die unteren drei Stufen (Haupt-/Arbeits-, Ablage- und Archivspeicher), der Notizblockspeicher wird zumeist direkt vom ↑Maschinenprogramm beherrscht, der

Zwischenspeicher ist unter Kontrolle der ↑CPU und Registerspeicher wird vom ↑Kompilierer oder, im Falle der Programmierung in ↑Assembliersprache, vom Maschinenprogramm selbst zugeteilt.

Speicherkachel Einheit fester Größe im ↑Hauptspeicher (↑realer Adressraum, ↑Kachel). Die Größe ist immer ganzzahlige Vielfache der Größe von einem ↑Speicherwort.

Speicherkohärenz Grad und Art der zusammenhängenden (kohärenten) Ausführung gleichzeitiger Operationen auf den ↑Arbeitsspeicher, wobei die Operationen dieselbe ↑Speicherzelle referenzieren (Unterschied zur ↑Speicherkonsistenz) und jede Operation von einer anderen ↑CPU (in einem ↑Multiprozessor) oder einem anderen ↑Rechenkern (in einem ↑Mehrkernprozessor) ausgegeben wird. Die mit diesen Operationen verbundenen gleichzeitigen Zugriffe auf die Speicherzelle verlaufen kohärent, wenn jede einzelne ↑Aktion dazu denselben Wert liefert. Dies ist ein typisches Merkmal von einem ↑Zwischenspeicher in solchen Systemen.
Für gewöhnlich bestimmt die Konstruktionskomplexität des allen Prozessoren gemeinsamen Arbeitsspeichers (↑*shared memory*) den Grad/die Art der jeweils zugesicherten Kohärenz. So ist für einfachere Systeme mit einer eher geringen, im Zehnerbereich liegenden Prozessor-/Kernanzahl Kohärenz zumeist durch die Hardware sichergestellt. Steigt jedoch die Anzahl stark an, sichert die Hardware gegebenenfalls nur noch für bestimmte Gebiete (↑*tile*) kohärente Zugriffe auf gespeicherte ↑Daten zu, wenn überhaupt. In solchen Fällen ist ein ↑nichtsequentielles Programm gefordert, das in seiner eigenen Berechnungsvorschrift die kohärente Sicht, die ein ↑nichtsequentieller Prozess sodann auf die betreffenden Speicherzellen haben soll, zusichert.

Speicherkonsistenz Grad und Art der relativen Ordnung von Operationen auf den ↑Arbeitsspeicher, wobei die Operationen nicht dieselbe ↑Speicherzelle referenzieren (Unterschied zur ↑Speicherkohärenz). Je nach Grad/Art der *Konsistenz* wird zwischen verschiedenen Modellen unterschieden, die gängigen Varianten sind (in der Reihenfolge abnehmender Stärke der Konsistenz): ↑strikte Konsistenz, ↑sequentielle Konsistenz, ↑Prozessorkonsistenz, ↑schwache Konsistenz und ↑Freigabekonsistenz.
Grundlage bildet ↑gemeinsamer Speicher, der real oder virtuell zur Verfügung steht. Letzteres meint einen ↑Arbeitsspeicher, der sich physisch über mehrere lokale ↑Hauptspeicher in einem ↑Multiprozessor oder ↑Mehrkernprozessor erstreckt, in logischer Hinsicht aber als ein gemeinsamer ↑Speicherbereich in Erscheinung tritt (↑DSM). Demgegenüber ist mit real ein Arbeitsspeicher gemeint, dessen Hauptspeicheranteil physisch allen Prozessoren gemeinsamen ist. In beiden Fällen kann für gewöhnlich nur noch eine abgeschwächte Form der Konsistenz bei gleichzeitigen Speicherzugriffen gewährleistet werden.

Speichermarke Marke, die von der ↑MMU nur gesetzt, aber nicht wieder gelöscht wird („klebriges Bit“). Erst das ↑Betriebssystem löscht ein solche Marke, beispielsweise beim Durchlauf einer bestimmten ↑Ersetzungsstrategie für eine ↑Seite.

Speichermedium Träger für Informationen, Datenträger. Die Informationen sind fotografisch (Film), mechanisch (Lochkarte/-streifen), magnetisch (Band, Platte), optisch (CD, DVD) oder elektronisch (Halbleiter) gespeichert.

Speicherpyramide Art der Darstellung der in einem ↑Rechensystem typischerweise existierenden ↑Speicherhierarchie, eine Pyramide stilisierend, zumeist gezeichnet als gleichschenkliges Dreieck in der Ebene. Die Höhe des Dreiecks reflektiert die Anzahl der übereinander liegenden Systeme oder Sorten von ↑Speicher. In Bezug auf diese Speicher nehmen ↑Zugriffszeit und Kapazität von der Spitze bis zur Basis des Dreiecks zu.

Speicherschutz Vorrichtung, die zum ↑Schutz gegen unautorisierte Zugriffe auf den ↑Arbeitsspeicher konstruiert ist. Technische Grundlage ist zumeist eine ↑MMU oder ↑MPU, die vom ↑Betriebssystem entsprechend seines Schutzmodells so programmiert wird, dass ein ↑Prozess nur auf die ihm jeweils zugewiesenen Speicherbereiche zugreifen kann. Neben solchen

hardwarebasierten Techniken gibt es auch softwarebasierte Ansätze, die ↑Typsicherheit von Programmabläufen voraussetzen. In solch einem Fall stellt der ↑Kompilierer sicher, dass das von ihm generierte Programm zur ↑Laufzeit (logisch) keine unautorisierten Speicherzugriffe hervorrufen kann.

Speicherstelle Ort, lokalisierbarer Bereich, in einem ↑Speicher.

Speicherstück Einheit bildender Teil (↑Speicherbereich) eines Ganzen (↑Arbeitsspeicher). Jede Einheit ist gleich groß (↑Speicherwort, ↑Seite) oder von verschiedener Größe (↑Segment von Speicherwörtern oder Seiten).

Speichervirtualisierung Funktion von einem ↑Betriebssystem, um ↑Hauptspeicher entsprechend seiner Anlage einem ↑Prozess scheinbar komplett und exklusiv zur Verfügung zu stellen. Die Maßnahme schafft die Illusion nicht nur von einem Hauptspeichermonopol, sondern auch von einer Hauptspeicherkapazität, die der wirklich vorhandenen um Größenordnungen übersteigen kann: bei einer ↑CPU mit 64-Bit ↑Wortbreite kann der ↑Adressraum eines Prozesses einen ↑Adressbereich von $[0, 2^{64} - 1]$ abdecken — bei einer Hauptspeichergröße der Winzigkeit von vielleicht gerade einmal 8 GiB (d.h., 2^{33} ↑Byte). Grundlage dafür ist ein ↑virtueller Adressraum für den Prozess und ↑virtueller Speicher. Letzteres benötigt vor allem eine ↑Ladestrategie und eine ↑Ersetzungsstrategie, um den Hauptspeicher im Wesentlichen nur noch als ↑Zwischenspeicher zwischen CPU und ↑Umlagerungsbereich erscheinen zu lassen.

Speicherwort Organisationseinheit in einem ↑Speicher (genauer: ↑Hauptspeicher), wobei jeder dieser Einheit eine ↑Adresse zugeordnet ist. Größe und Struktur einer solchen Einheit sind einem ↑Maschinenwort entsprechend ausgelegt.

Speicherzelle Bezeichnung für die kleinste adressierbare Einheit im ↑Arbeitsspeicher. Heute (2016) typischerweise ein ↑Byte (↑Oktett) in einem ↑Speicherwort: der Arbeitsspeicher ist wortorientiert, seine Größe ist ein ganzzahlig Vielfaches der ↑Wortbreite, aber er ist byteweise adressierbar. Für gewöhnlich folgt die ↑Adresse eines Speicherworts jedoch einer gewissen Linienführung (↑*alignment*), um der ↑CPU einen schnellen Zugriff auf den ↑Hauptspeicher zuzusichern. Eine Wortadresse orientiert sich daher, wie schon die Kapazität des Arbeitsspeichers, an der jeweiligen Wortbreite, ihr Wert ist also Vielfaches von 1 (byteorientiert, gerade/ungerade) beziehungsweise 2, 4 oder 8 (wortorientiert, gerade).

Speicherzugriffsfehler Bezeichnung einer ↑Schutzverletzung durch einen ↑Prozess beim Zugriff mit einer ↑Adresse, die außerhalb von dem für sie definierten ↑Adressbereich liegt. Der Adressbereich beschreibt die für ein ↑Segment gültigen Adressen. Jede Adresse, die sich auf eine ↑Speicherstelle außerhalb des Segments bezieht, wird im Zugriffsfall eine ↑Ausnahme herbeiführen (↑*segmentation fault*). Typischerweise werden solche Segmente durch eine ↑MMU oder ↑MPU abgesichert, die der ↑CPU im Falle eines Zugriffsfehler signalisiert, den in Ausführung befindlichen ↑Maschinenbefehl abzufangen (↑*trap*).

Speicherzuteilung Allokation von einem ↑Speicherbereich, Zuweisung des Bereichs als ↑wiederverwendbares Betriebsmittel an eine ↑Prozessinkarnation. Je nach Art der Anfrage durch einen ↑Prozess läuft der Vorgang auf verschiedenen Ebenen ab. Setzt der Prozess dazu einen ↑Systemaufruf (`mmap(2)`, `brk(2)`) ab, so versucht das ↑Betriebssystem den allozierten Speicherbereich im ↑Adressraum des Prozesses verfügbar zu machen. Ist für den Prozess ein ↑logischer Adressraum/↑virtueller Adressraum definiert, muss dazu ein passender und noch unbenutzter ↑Adressbereich (d.h., ein ↑Loch) darin gefunden werden. Im Falle des virtuellen Adressraums reicht es bereits, den eigentlichen Speicherbereich nur im ↑Umlagerungsbereich zu verbuchen. Wirklicher Platz im ↑Hauptspeicher wird dann erst bei Bedarf (↑Ladestrategie) geschaffen, nämlich wenn der Zugriff darauf erfolgt. Dagegen gelingt im Falle des logischen Adressraums die Zuteilung nur, wenn entsprechend der Größe des Speicherbereichs ein oder mehrere Abschnitte direkt im Hauptspeicher verfügbar gemacht werden konnten: im logischen Adressraum ist der Speicherbereich zusammenhängend, nicht aber zwingend ebenso

im Hauptspeicher (↑realer Adressraum). Ist für den Prozess gar nur der reale Adressraum definiert, muss der Speicherbereich auch dort in einem Stück verfügbar sein. Oberhalb des Betriebssystems, auf der Ebene vom ↑Maschinenprogramm, ruft der Prozess für gewöhnlich ein ↑Unterprogramm (`malloc(3)`) auf, um ↑Haldenspeicher zugeteilt zu bekommen. Dieses Unterprogramm ist typischerweise Bestandteil von dem mit einer Programmiersprache verbundenem ↑Laufzeitsystem. Der Unterprogrammaufruf kann in einen Systemaufruf münden, wenn nämlich in der ↑Halde kein Loch (d.h., freier Abschnitt) entsprechend der Größe des angeforderten Speicherbereichs vorhanden ist. Ein solcher Systemaufruf (z.B. `sbrk(2)`) schafft dann einen freien Abschnitt, mit dem ganz oder teilweise die Speicheranforderung bedient wird. In beiden Fällen sorgt eine bestimmte ↑Platzierungsstrategie für die Verortung des angeforderten Speicherbereichs, wobei die beiden betrachteten Ebenen durchaus auch verschiedene Strategien verfolgen können.

Sperrfreiheit (en.) ↑*lock-freedom*. Fortschrittsgarantie für ↑nichtblockierende Synchronisation. Diese Garantie besagt, dass irgendein ↑Prozess eine Operation in einer endlichen Anzahl von Schritten vollenden kann, ohne Rücksicht auf die relativen Geschwindigkeiten der anderen Prozesse. Eine im Vergleich zu ↑Wartefreiheit schwächere Zusicherung, die zwar einem System (von Prozessen), nicht jedoch jedem einzelnen Prozess darin Fortschritt garantiert: gerät ein Prozess in eine ↑Konkurrenzsituation, birgt dies die Gefahr von ↑Verhungern.
Der Vorteil dieser Form von nichtblockierender Synchronisation liegt in der einfacheren Umsetzung, als dies im wartefreien Fall getan ist. Gilt für das ↑Betriebssystem keine ↑Echtzeitbedingung, ist sperrfreie nichtblockierende Synchronisation für gewöhnlich unproblematisch und liefert in vielen Beispielen eine praktikable Lösung.

Spezialbetriebssystem Bezeichnung für ein ↑Betriebssystem, das nur speziell und für gewöhnlich sehr eingeschränkt eingesetzt werden kann und damit nur einzelnen Anwendungszwecken dient; Gegenteil von ↑Universalbetriebssystem. Die Funktionen eines solchen Betriebssystems sind typischerweise in Hinblick auf spezielle Bedürfnisse einer bestimmten Klasse von Anwendungen optimiert. Dies betrifft insbesondere auch den Umfang der von dem Betriebssystem zu leistenden Aufgaben: nur die Funktion, die unbedingt erforderlich ist, wird durch das Betriebssystem auch bereitgestellt beziehungsweise ist im Betriebssystem überhaupt realisiert. Schon allein damit ist ein solches Betriebssystem in räumlicher Hinsicht (z.B. Platzbedarf im ↑Hauptspeicher) für gewöhnlich schlanker als ein Universalbetriebssystem. Auch in zeitlichen Belangen (z.B. ↑Latenzzeit oder ↑Laufzeit) zeigt sich zumeist ein anderes Bild: das Zeitverhalten eines solchen Betriebssystems ist oftmals nicht nur besser, sondern auch vorhersagbar. So ist ein typischer Anwendungsfall insbesondere der ↑Echtzeitbetrieb.

spinlock (dt.) ↑Umlaufsperre.

SPN Abkürzung für (en.) *shortest process next*. Bezeichnung für eine Strategie zur ↑Planung der Bearbeitung von ↑Aufgaben durch einen ↑Prozessor in einem *Vorrangverfahren*, das die Aufgaben nach zunehmender ↑Bedienzeit sortiert und dann der Aufgabe, deren Bearbeitung am schnellsten geschehen kann, den Vorzug gibt: je kürzer die Bearbeitungsdauer einer Aufgabe durch einen ↑Prozess, desto höher seine ↑Priorität, wobei der Prioritätswert einen Zeitwert darstellt.
Im Unterschied zu ↑SJN zielt diese Strategie gemeinhin auf die Förderung von Interaktion in einem ↑Rechensystem ab (↑*conversational mode*). Die dazu erforderliche ↑mitlaufende Planung setzt die Bearbeitungszeit einer Aufgabe der Dauer des einzelnen ↑Rechenstoßes eines Prozesses gleich. Das Verfahren steht vornehmlich im Zusammenhang mit der Planung interaktiver Prozesse, deren meist kurze Rechenstöße häufig nur dazu dienen, ↑Ein-/Ausgabe anzustoßen und damit jeweils einen Dialog mit einem „externen“ Prozess zu führen.
Mit ↑SJN gemeinsames Merkmal ist die ↑probabilistische Planung, da die jeweiligen Bearbeitungszeiten für gewöhnlich nur abgeschätzt werden können (↑*time series analysis*). Darüber hinaus ist das Verfahren ein Beispiel für die nicht ↑präemptive Planung, da ein eintreffender Prozess, dessen abgeschätzte Ausführungszeit kürzer ist als die des gegenwärtig auf dem

Prozessor stattfindenden Prozesses, niemals zum Prozessorentzug führt. Wie bei ↑FCFS, ↑RR oder ↑VRR auch, werden eintreffende Prozesse lediglich der ↑Einplanung zugeführt. Die ↑Einlastung eines Prozesses erfolgt immer erst dann, wenn der gegenwärtig stattfindende Prozess die Bearbeitung seiner (Teil-) Aufgabe beendet.

Exkurs Dreh- und Angelpunkt zur Abschätzung der Länge des zukünftigen Rechenstoßes eines Prozesses ist eine fortlaufend stattfindende Zeitmessung zur Bestimmung der Dauer eines jeden Rechenstoßes, den der Prozess bisher verrichtet hat. Im Fokus steht dabei die Operation zum ↑Prozesswechsel. Diese Operation beendet den Rechenstoß des Prozesses, der den ↑Prozessor abgibt und startet den Rechenstoß des Prozesses, der den Prozessor zugeteilt erhalten hat. Der Umschaltvorgang muss lediglich um Anweisungen zur Zeitnahme ergänzt werden, nämlich um die *Start-* und *Endzeit* eines Rechenstoßes aufzuzeichnen. Aus beiden gemessenen Zeiten lässt sich durch Differenzbildung die wirkliche Dauer eines Rechenstoßes sodann sehr einfach ermitteln:

$$t_{burst} = |p_{end} - p_{start}|.$$

Allerdings umfasst dieser Wert sämtliche Verzögerungszeiten, die der Prozess während seines jüngsten Rechenstoßes erfahren hat. Dies sind insbesondere Zeiten, die mit jeder einzelnen ↑Unterbrechungsbehandlung zu Buche schlagen. Diese Zeiten sind pro Prozess zu akkumulieren und die resultierende Summe ist von der festgestellten Zeitdifferenz abzuziehen:

$$t_{burst} = |p_{end} - p_{start}| - t_{delay}.$$

Dies ergibt die *normalisierte Rechenstoßlänge* eines Prozesses, in die durch eventuelle ↑Unterbrechungsanforderungen entstehende ↑Gemeinkosten nicht einfließen. In ↑C sei die entsprechende Funktion wie folgt formuliert:

```
time_t burst(burst_t *this) {                       /* normalised CPU burst */
    return llabs(this->close - this->start) - this->delay;
}
```

Dabei definiert `close` den Zeitpunkt (Endzeit), zu dem der Rechenstoß abgeschlossen und `start` den Zeitpunkt (Startzeit), zu dem der Rechenstoß gestartet wurde. Attribut `delay` definiert die aufgelaufenen Verzögerungszeiten seit dem letzten Prozesswechsel. Festgelegt wird dieser Wert im zentralen ↑Ausnahmezuteiler, der dazu beispielsweise nach folgendem Muster aufgebaut ist:

```
__attribute__ ((interrupt)) void train(train_t *this) {
    time_t start;                                  /* user-level offset time */

    if (stage(this, PROVE|USER))      /* user-level CPU-burst interrupted? */
        start = time();                          /* yes, remember offset time */

    operate(this);                                   /* proceed with any job */

    if (stage(this, PROVE|USER)) {          /* resume user-level CPU burst? */
        process_t *task = being(ONESELF);    /* yes, point to my descriptor */
        task->time.delay += llabs(time() - start);  /* and record lag time */
    }
}
```

Der Zuteiler prüft, auf welcher Ebene (`stage`) der unterbrochene Prozess gegenwärtig tätig ist. Im Falle der ↑Benutzerebene (`PROVE|USER`) werden Start- und Endzeit der verursachten Verzögerung genommen und im ↑Prozesskontrollblock des betreffenden Prozesses als Verzögerungszeit aufsummiert (einfachheitshalber wird hier ein möglicher ↑arithmetischer Überlauf außen vor gelassen). Dazwischen läuft (`operate`) der Zuteiler für eine beliebige Aufgabe, die dem eigentlichen Zweck der vorliegenden Unterbrechung entspricht. Zudem wird im gegebenen Beispiel zur Bestimmung der Ebene des unterbrochenen Prozesses angenommen, dass die ↑CPU in ihrem ↑Unterbrechungszyklus einen verwendbaren Hinweis in

dem auf den ↑Laufzeitstapel gesicherten ↑Prozessorstatus (`this`) hinterlässt. Im Falle von ↑x86 (ab Modell ↑i286) liefert beispielsweise der gesicherte ↑Segmentwähler den entsprechenden Hinweis (↑DPL). Steht dagegen keine Hardwarehilfe zur Verfügung, lässt sich durch Zählen der Handhaberverschachtelung diese Ebene feststellen (vgl. ↑AST).
Die im Ausnahmezuteiler akkumulierte Verzögerungszeit eines Prozesses ist in der Operation zum Prozesswechsel auf Null zu setzen. Diese Operation beendet den Rechenstoß des Prozesses, der den ↑Prozessor abgibt und startet den Rechenstoß des Prozesses, der den Prozessor zugeteilt erhalten hat. Demzufolge kann für den seinen Rechenstoß aufnehmenden Prozess noch keine Verzögerungszeit aufgelaufen sein. Nachfolgend ein Beispiel für diese Operation (vgl. S. 112):

```
void seize(process_t *task) {                          /* dispatch process */
    process_t *last = being(ONESELF);      /* access process control block */
    state(&last->mood, CLEAR|RUNNING);       /* cancel running state, only */
    last->time.close = time();                          /* end of CPU burst */
    last->flow.crux = shift(task);        /* switch process, save context */
    task->time.start = time();                        /* begin of CPU burst */
    task->time.delay = 0;                             /* no time delay, yet */
    state(&last->mood, FLUSH|RUNNING);       /* define running state, only */
}
```

Wenn ein Prozess seine Tätigkeit aufnimmt, wird die Startzeit (`time.start`) in seinem Prozesskontrollblock vermerkt und auch die Verzögerungszeit (`time.delay`) zurückgesetzt. Beendet der Prozess diese Tätigkeit, um kurz danach (hier durch einen ↑Koroutinenwechsel) die Kontrolle über den Prozessor abzugeben (`shift`), wird die Endzeit (`time.close`) genommen. Die Differenz aus beiden Zeitwerten ergibt die wirkliche Länge des letzten Rechenstoßes eines jeweiligen Prozesses, sie wird aber erst zur Prognose der Länge des zukünftigen Rechenstoßes gebildet (s. nächsten Absatz). Unter der Annahme des ↑Lokalitätsprinzips wird die Länge des nächsten (unbekannten, abzuschätzenden) Rechenstoßes eines Prozesses nach seiner Wiederaufnahme mit einer gewissen Wahrscheinlichkeit der Länge seines letzten (bekannten) Rechenstoßes entsprechen.
Um nun die Länge des nächsten Rechenstoßes eines Prozesses zu prognostizieren, wäre ein naheliegender Ansatz, die gemessenen Längen zu akkumulieren, um aus der damit berechneten prozessspezifischen totalen Rechenzeit das *arithmetische Mittel* zu bilden und als Schätzwert heranzuziehen. Nachfolgende Funktion skizziert die grundlegenden Berechnungsschritte:

```
time_t guess(process_t *task) {                  /* weighted burst average */
    time_t burst = burst(&task->time);                        /* fix length */
    task->time.total += burst;                          /* settle this burst */
    task->time.count += 1;                            /* book the calculation */
    return decay(task->time.total / task->time.count, burst); /* estimate */
}
```

Für einen Prozess die totale Rechenzeit sowie die Anzahl der durchgeführten Akkumulationen festzuhalten, ermöglicht es, ihn in Bezug auf seine durchschnittliche Rechenstoßlänge zu charakterisieren: kleinere Werte stufen ihn als interaktiv oder ein-/ausgabeintensiv ein, größere Werte beschreiben ihn als rechenintensiv. Allerdings bleibt auf Basis allein des arithmetischen Mittels die Vorhersage der als nächstes zu erwartenden Rechenstoßlänge recht ungenau. Dies ist auch dann immer noch der Fall, wenn, wie hier skizziert, dieser Mittelwert (durch `decay`, s.u.) mit der zuletzt gemessenen Rechenstoßlänge (`burst`) gewichtet verrechnet wird, um das Ergebnis dann als neuen Schätzwert für den Prozess festzuhalten.
Schwachstelle der gezeigten Lösung ist die effektiv fehlende Rückkopplung der Schätzung in Bezug auf das gegenwärtige Prozessverhalten. Hierzu muss der jeweils bestimmte Schätzwert in die „Prozessgeschichte“ eingehen, nämlich vermerkt und in die nächste Schätzung wieder eingespeist werden. Eine Umsetzung zeigt nachfolgendes Beispiel, wobei hier die nunmehr als Option zu betrachtende Buchführung zur totalen Rechenzeit und Charakterisierung eines

Prozesses weggelassen wurde:

```
time_t guess(process_t *task) {                /* weighted burst estimation */
    return decay(task->time.guess, burst(&task->time));
}
```

In diesem Beispiel — bei Verwendung dieser Funktion im Verlauf der Bereitstellung eines Prozesses (`ready`, s.u.) und dabei zur Aktualisierung des prozessspezifischen Schätzwerts (`time.guess`) — ergibt sich die erwartete Länge des nächsten Rechenstoßes eines Prozesses als ↑exponentielle Glättung des Durchschnittswerts von der vorangegangenen Schätzung sowie der aktuellen Messung, wobei diese beiden Werte einer Wichtung unterzogen werden. Die Glättung sorgt dafür, dass die Bedeutung eines zurückliegenden Rechenstoßes mit zunehmendem „Alter“ abnimmt und des aktuellen Rechenstoßes hoch bleibt. Erreicht wird dies durch nachfolgend skizzierte Filterfunktion (*decay filter*), bei der die Stärke der Glättung durch einen *Wichtungsfaktor* einstellbar ist:

```
time_t decay(time_t guess, time_t burst) { /* exponential weighted average */
    extern float trend();                  /* weighting factor 0 <= f <= 1 */
    return (guess * (1 - trend())) + (burst * trend());
}
```

Hat der Wichtungsfaktor (`trend`) den Wert 1, wird die bisherige Entwicklung der Rechenstöße komplett ausgeblendet. Für andere Werte ($0 \leq$ `trend` < 1) dominieren die bisher vollbrachten Rechenstöße eines Prozesses mehr oder weniger stark. Diesen Wichtungsfaktor zu bestimmen und eine diesbezügliche Wichtungsfunktion herzuleiten, die dem wirklichen ↑Programmablauf sehr nahe kommt, ist der eigentliche Knackpunkt des Verfahrens
Da die Zeitmessungen erst beim Prozesswechsel (`seize`) geschehen, ist die Abschätzung der Länge des nächsten Rechenstoßes des gerade stattfindenden Prozesses vor dem Wechsel zwecklos: die Endzeit seines Rechenstoßes wurde noch nicht festgehalten, das heißt, der gegenwärtige Rechenstoß des Prozesses ist noch nicht abgeschlossen. Geschieht der Umschaltvorgang (`seize`), weil der Prozess auf ein ↑Ereignis warten muss und daher in den ↑Prozesszustand blockiert übergegangen ist, wird die ↑Einplanung immer auf Veranlassung eines anderen Prozesses, den ↑Leerlaufprozess eingeschlossen, erfolgen: also nach dem Prozesswechsel. In dem Fall liegen Start- und Endzeit des letzten Rechenstoßes des einzuplanenden Prozesses vor, womit die Abschätzung des nächsten Rechenstoßes dieses Prozesses im Moment seiner erneuten Bereitstellung möglich ist.
Wie eingangs erwähnt arbeitet das Verfahren nicht präemptiv, womit es insbesondere auch nicht auf ↑Zeitscheiben basiert: die für ↑RR benötigte Funktion (`check`, S. 126) zum regelmäßigen Prozessorentzug entfällt, womit die in diesem Fall bedingt erfolgende erneute Bereitstellung des Prozesses durch sich selbst eben auch nicht geschehen kann und daher die zur Aufnahme auf die ↑Bereitliste nötige Rechenstoßlängenabschätzung noch vor dem Prozesswechsel nicht in Frage kommt. Anders verhält es sich jedoch, wenn der Prozess von sich aus pausiert (`pause`, S. 47). Die Annahme, ein solches Vorgehen ist wirkungslos und bringt nichts außer Zeitverschwendung, da der pausierende Prozess ja nach wie vor der kürzeste ist, muss allerdings nicht stimmen.
Nicht präemptiv zu arbeiten bedeutet nicht, eine Prozessbereitstellung im Verlauf einer ↑Unterbrechungsbehandlung sei nicht möglich. Diese Arbeitsweise bedeutet lediglich, dass es aus einen solchen Verlauf heraus nicht zur ↑Einlastung eines zeitnah bereitgestellten Prozesses kommt. Folglich kann das Verfahren sehr wohl Prozesse einplanen, deren Rechenstoßlängen kürzer sind als die des im Zustand laufend befindlichen Prozesses. Würde letzterer nun pausieren wollen, könnte demnach das Szenario eintreten, dass sich der stattfindende Prozess zugunsten eines kürzeren Prozesses selbst bereitstellt und dazu noch vor Prozessorabgabe die erwartete Länge seines nächsten Rechenstoßes bekannt sein müsste.
Die einfachste Lösung dazu wäre, einem Prozess keine Möglichkeit zum Pausieren zu eröffnen, indem etwa dem ↑Maschinenprogramm dazu kein (und auf `pause` hinauslaufender) ↑Systemaufruf zur Verfügung steht — ein nicht ganz untypischer Ansatz. Alternativ kann immer

dann, wenn die Endzeit des aktuellen Rechenstoßes noch nicht bekannt ist, der zuletzt für den betreffenden Prozess zur Bereitstellung herangezogene Schätzwert erneut verwendet werden. Diese Lösung zeigt nachfolgendes Beispiel (vgl. S. 46):

```
void ready(process_t *task) {                /* schedule according to SPN */
    if (task != being(ONESELF))                              /* not myself? */
        task->time.guess = guess(task);     /* yes, estimate burst length */
    state(&task->mood, READY);                /* set process ready to run */
    ticket(&task->line, task->time.guess);             /* define sort key */
    enlist(labor(), &task->line);      /* sort descriptor into ready list */
}
```

Der Schätzwert (`time.guess`) wird nicht neu berechnet und damit auch nicht aktualisiert, wenn der gegenwärtige Prozess (`ONESELF`) sich selbst (`task`) bereitstellen wollte. Damit fließt die aus aktuellem Prozessverhalten resultierende Rechenstoßlänge für einen Prozess, der pausieren und somit sich selbst auf die Bereitliste setzen müsste, nicht in die Berechnung der zu erwartenden Länge seines nächsten Rechenstoßes ein. Nur wenn die Bereitstellung durch einen anderen Prozess erfolgt, kann die Länge des jüngsten Rechenstoßes des jeweils bereitzustellenden Prozesses auch Berücksichtigung finden. Bei der Einplanung nicht immer und vor allem auf aktuelle Rechenstoßlängen zurückgreifen zu können, ist der eigentliche Nachteil dieser Lösung. Denn gerade die Berücksichtigung aktueller Werte ist für die möglichst genaue Vorhersage zukünftigen Prozessverhaltens besonders wichtig.
Eine andere Lösung besteht darin, die Zeitnahme für einen Prozess vorzuziehen. Der späteste Zeitpunkt, um die Rechenstoßlänge für einen pausierenden Prozess noch zu bestimmen, ist eine Zeitnahme an der Schnittstelle zum Planer, also noch vor dem eigentlichen Prozesswechsel durch den ↑Umschalter. In dem Fall geht der Zeitaufwand für alle Berechnungen, die durch die Einplanung selbst durchzuführen sind, nicht in die Rechenstoßlänge eines Prozesses ein. Gleiches gilt für die Durchführung der Zeitnahme bereits an der Schnittstelle zum ↑Betriebssystem (d.h., bei jeder ↑Ausnahme), wie es etwa zur Bestimmung von ↑Benutzerzeit und ↑Systemzeit gemeinhin praktiziert wird. In beiden Fällen kann es jedoch zu einer Verzerrung der Rechenstoßlängenabschätzung kommen, da nicht für jeden Prozess der Zeitaufwand innerhalb des Betriebssystems beziehungsweise Planers gleich ist. Die im Hauptausführungsstrang eines Prozesses innerhalb des Betriebssystems stattfindenden ↑Aktionen sind insbesondere durch die beim ↑Systemaufruf übergebenen Parameterwerte bestimmt. Diese Aktionen finden synchron zum Prozess selbst statt, sie sind überwiegend prozessbezogen. Durch eventuelle ↑Unterbrechungen hervorgerufene Nebenausführungsstränge sind demgegenüber gemeinhin nicht prozessbezogen, sondern beziehen sich auf die ↑Peripherie und generieren bestimmte (von Prozessen ggf. erwartete) ↑Ereignisse. Die dabei anfallenden Zeitaufwände (d.h., Verzögerungszeiten) fließen daher, wie oben bereits skiziert (`train`), auch nicht in die Berechnung der Rechenstoßlänge eines Prozesses ein.
Die Zeitnahme beim Prozesswechsel umgeht die eben geschilderte Problematik und ermöglicht zudem eine weitere Abstufung der Systemzeit. Wenn diese Lösung jedoch auch für pausierende Prozesse funktionieren soll, sind diese in einen bestimmten ↑Schwebezustand zu versetzen, der es anderen Prozessen ermöglicht, die noch ausstehende Bereitstellung und damit Einsortierung in die Bereitliste zu vollenden. Dieser Schwebezustand beginnt mit dem Wunsch zu pausieren, erstreckt sich über den Wechsel hinweg zum nächsten Prozess und endet damit, wenn letzterer seinen Vorgänger (nämlich der pausierte Prozess) auf die Bereitliste setzt. Indem der Nachfolgeprozess die Bereitstellung des pausierten Prozesses vornimmt, ist durch den erfolgten Prozesswechsel die Zeitnahme erfolgt, wodurch schließlich die Abschätzung der Rechenstoßlänge des Vorgängerprozesses möglich wird und somit der pausierte Prozess noch seinen Platz auf der Bereitliste finden kann.

spool (dt.) Spule. Abkürzung für (en.) „*simultaneous peripheral operations online*“.

spool area (dt.) ↑Spulbereich.

spooler (dt.) Spulmaschine, Druckpuffer.

spooling (dt.) ↑Spulen.

Sprungmarke (en.) ↑*jump label*. Bezeichnung für die ↑symbolische Adresse von einem ↑Maschinenbefehl, der als Ziel einer Programmverzweigung Verwendung finden kann.

Spulbereich Pufferplatz im ↑Hintergrundspeicher, um zwischengepufferte ↑Daten zur Eingabe an einen (schnellen) ↑Prozess oder Ausgabe an ein (langsames) ↑Peripheriegerät wieder „abspulen“ zu können. Ursprüngliches ↑Speichermedium hierfür ist das Magnetband (↑*spool*) und es kommen Magnetbandspulgeräte zur Ein-/Ausgabe zum Einsatz — woraus sich auch der Begriff „↑*spooling*“ ableitet. Moderne Ansätze nutzen für diese Vorgehendsweise jede Form von Fest- oder Wechselplatten als Speichermedium.

Spulen ↑Stapelverarbeitung von Ein-/Ausgabeaufträgen. Die ein-/auszugebenden ↑Daten gelangen zunächst in einen Pufferbereich (↑*spool area*). Ein ↑Dämon (↑*spooler*) entnimmt diesem die Ein-/Ausgabeaufträge und leitet sie nacheinander an das jeweilige ↑Peripheriegerät weiter. Für gewöhnlich werden die Aufträge nach gewissen Kriterien sortiert auf einer Warteschlange gehalten (↑Ablaufplanung), bis die zur Abwicklung des Auftrags benötigte ↑Entität mit dem nächsten Auftrag bestückt werden kann (↑Einlastung). Typischer Anwendungsfall für solch eine Vorgehensweise zur Ausgabe ist das Drucken (`lpr(1)`, `lpd(8)`), wobei die Entität einem ↑Peripheriegerät („externer Prozess“ Drucker) entspricht. Für langsame Eingabegeräte und -medien (z.B. ↑Lochkarte) führt der Dämon eine Liste von wartenden Prozessen, die ausgelöst werden, sobald Eingabe für sie bereitsteht. In diesem Szenario stellt der jeweilige (interne) Prozess, damit sein ↑Prozessor, die mit einer weiteren ↑Aufgabe einzulastende Entität dar.

spurious interrupt (dt.) unechte, unberechtigte ↑Unterbrechung.

SRAM Abkürzung für (en.) *static RAM*, (dt.) statischer ↑RAM.

SRTF Abkürzung für (en.) *shorted remaining time first*. Bezeichnung für ↑probabilistische Planung von ↑Prozessen nach zunehmender ↑Rechenstoßrestlänge ....

SSD bkürzung für (en.) *solid state disk/drive*, (dt.) Halbleiterspeicher, wobei Platte/Laufwerk sinnbildlich zu sehen ist: es handelt sich weder um eine Platte, noch um ein Laufwerk für eine Platte.

stack pointer (dt.) ↑Stapelzeiger.

Stammdateisystem Bezeichnung für ein ↑Dateisystem, das auf dem ↑Datenträger liegt, von dem das ↑Betriebssystem urgeladen wird (↑*bootstrap*).

Stapel Stoß übereingelegter gleicher Dinge (in Anlehnung an den Duden). Dem Ursprung nach ist jedes dieser Dinge eine ↑Lochkarte, wobei ein bestimmter Satz davon einen ↑Auftrag beschreibt. Dabei bildet dieser Satz durch übereinanderstapeln von Lochkarten eine wohldefinierte Folge von Anweisungen an ein ↑Rechensystem. Mittlerweile ist die Lochkarte jedoch weniger das Instrument, um auch eine bestimmte Sequenz solcher Anweisungen zu definieren, sondern stellvertretend nur noch der Datenträger (↑Speichermedium) für ein ↑Kommando. So lässt sich ein solcher Anweisungsstoß eben auch als ↑Datei speichern, wobei die einzelnen darin enthaltenen Kommandos dann unverändert weiterhin ihren Zweck erfüllen. Für gewöhnlich ist die Formulierung für einen ↑Stapelauftrag in genau dieser Form heute (2016) üblich, indem nämlich das in ↑Kommandosprache formulierte ↑Programm als Datei der ↑Stapelverarbeitung zugeführt wird. Ein modernes Beispiel dafür liefert ein für einen bestimmten ↑Kommandointerpreter vorgesehenes ↑*shell*-Skript.

Stapelauftrag Aufeinanderfolge von Schritten in einem ↑Stapelprozess, wobei jeder Schritt einem ↑Programm entspricht. Ein ↑Auftrag zur Ausführung einer Reihe von Programmen auf einen bestimmten Satz (↑*batch*) von Eingaben von ↑Daten, anstatt auf nur einer einzelnen Eingabe.

Stapelbetrieb ↑Betriebsart zur ↑Stapelverarbeitung.

Stapelmonitor Steuerprogramm zur ↑Stapelverarbeitung. Das ↑Programm führt jeden eingehenden ↑Stapelauftrag dem ↑Kommandointerpreter zu, überwacht die Ausführung von jeden einzelnen in dem ↑Stapel beschriebenen ↑Auftrag und erstellt suksessive das ↑Konsolenprotokoll. Die den ↑Stapelbetrieb bestimmende Funktionseinheit in einem ↑Betriebssystem, das speziell für diese ↑Betriebsart ausgelegt ist.

Stapelprozess Vorgang der ↑Stapelverarbeitung.

Stapelsegment Bezeichnung von einem ↑Datensegment, das den ↑Stapelspeicher von einem ↑Prozess bildet.

Stapelspeicher ↑Speicher, der als Stapel (dynamischeDatenstruktur) organisiert und nach ↑LIFO betrieben wird.

Stapelverarbeitung Verfahren zur Durchführung der einem ↑Rechensystem übergebenen Aufträge, nämlich um ein ↑Programm nach dem anderen automatisch zur Ausführung zu bringen und vollständig abzuarbeiten, dazu jeweils die benötigten ↑Daten zur Eingabe bereitzustellen und die Ausgabedaten zwischen- oder endzuspeichern beziehungsweise darzustellen oder auszudrucken. Die vollständig beschriebenen Aufträge sind in einem ↑Stapel hinterlegt, sie werden ohne weitere Interaktion mit der beauftragenden externen ↑Entität von einem ↑Stapelmonitor abgearbeitet. Dem dabei von dem Stapelmonitor erstellten ↑Konsolenprotokoll kann nach erfolgter Abarbeitung des Stapels der Verlauf und das jeweilige Ergebnis der durchgeführten Arbeitsschritte entnommen werden. Das bevorzugte Verarbeitungsverfahren für Routineaufgaben, beispielsweise um am Tagesende Zahlungseingänge einzubuchen (Rechnungswesen), am Wochenende Verkaufsstatistiken zu generieren (Einzelhandel), am Monatsende Gehaltsabrechnungen zu erstellen (Personalwesen), am Quartalsende Daten einer Aktiengesellschaft aufzubereiten (Berichtswesen) und am Jahresende den rechnerischen Abschluss eines kaufmännischen Geschäftsjahres vorzulegen (Wirtschaftswesen). Aber auch in technischen Bereichen finden sich viele Anwendungsbeispiele, etwa die Digitalisierung von Umkehr- und Negativfilm sowie die „Entwicklung“ der generierten Rohdaten (Fotobearbeitung), die Bildung einer Fertigungsreihenfolge von Produktionsaufträgen (Sequenzierung), das Erstellen von Kopien gespeicherter Informationen (Datensicherung) beziehungsweise die Restauration der Originaldaten nach einem Systemausfall (Datenwiederherstellung) und so weiter — bis hin zur automatischen Fertigung von einem ↑Maschinenprogramm oder ↑Betriebssystem (↑*build management*).

Stapelzeiger ↑Register der ↑CPU, das ihrem Steuerwerk (*control unit*) das gegenwärtige obere Ende in einem ↑Laufzeitstapel (*stack top*) anzeigt: auch ↑SP genannt. Diese Angabe ist eine ↑Adresse im ↑Arbeitsspeicher, an der die ↑Speicherstelle entweder von dem zuletzt gestapelten oder für das nächste zu stapelnde Datum zu finden ist. Mit jeder Stapeloperation wird der Registerinhalt automatisch um die Größe (Vielfaches von einem ↑Byte) des zu stapelnden Datums verändert, und zwar vor oder nach der Operation — was die beiden Möglichkeiten, worauf genau die im Register enthaltene Adresse verweist, begründet. Je nach CPU ist diese Größe fest (typisch für ↑RISC) oder variabel (typisch für ↑CISC), im letzteren Fall aber nur bis zu einer durch die ↑Wortbreite bestimmten oberen Grenze:

$$1 \leq quantity \leq (width(word) + width(byte) - 1))/width(byte),$$

wobei *quantity* eine Byteanzahl repräsentiert und die Funktion *width* eine Bitanzahl liefert. Darüber hinaus kann bei der Adressierung dieser Speicherstelle die Randbedingung gelten, dass der Adresswert im Register ganzzahlig Vielfaches der Größe (Byteanzahl) des dort zu lesenden/schreibenden Operanden sein muss (↑Ausrichtung). In Abhängigkeit von der durch die CPU definierten Expansionsrichtung des Stapels wird der Registerinhalt bei der Datumsablage (*push*) dekrementiert (*top-down stack*) oder inkrementiert (*bottom-up stack*),

das heißt, der Stapel wächst entweder von hohen zu niedrigen oder von niedrigen zu hohen Adressen. Die inverse Operation der Datumsentnahme (*pop*) wirkt umgekehrt.

start time (dt.) ↑Startzeit.

Startblock ↑Block auf einem ↑Datenträger, in dem der ↑Urlader liegt.

Startzeit (en.) ↑*start time*. Bezeichnung für eine ↑Prozessgröße, die den Zeitpunkt festlegt, zu dem die Durchführung einer ↑Aufgabe tatsächlich beginnt.

starvation (dt.) ↑Verhungern.

statischer Speicher Bezeichnung für einen ↑Speicher, dessen räumliches Fassungsvermögen (Kapazität) unveränderlich ist; Gegenteil von ↑dynamischer Speicher. Typisches Beispiel dafür ist das ↑BSS- oder ↑Datensegment von einem (↑UNIX) ↑Prozess. Beide bilden jeweils eine Zusammenfassung der Platzhalter für die verwendeten Programmvariablen, initialisiert (Daten) oder nicht initialisiert (BSS, jedoch zur ↑Ladezeit mit 0 vorbelegt), und dienen zur ↑Laufzeit der Speicherung von Berechnungsergebnissen. Die Größe von diesen Segmenten wird zur ↑Bindezeit festgelegt und verändert sich zur Laufzeit eines Programms nicht mehr.

Statusregister Behältnis in der ↑CPU, in dem eine Sammlung von Zustandsanzeigen (*status-flag bits*) vermerkt ist. Diese Anzeigen werden als Nebeneffekt bei der Ausführung von einem ↑Maschinenbefehl aktualisiert, etwa bei arithmetisch-logischen Operationen (typisch) oder beim Laden von einem ↑Speicherwort in ein ↑Prozessorregister (untypisch, MOS Technology 6502); sie können aber auch explizit durch spezielle Maschinenbefehle gelöscht oder gesetzt werden. Der jeweilige Zustand wird durch ein ↑Markierungsbit in diesem ↑Register dargestellt, wobei folgende Reihe von Bits typisch ist: Übertrag (*carry*), Überlauf (*overflow*), Vorzeichen (*sign*), Null (*zero*) und Parität (*parity*). Auf Basis dieser Zustandsanzeigen lassen sich in ↑Assembliersprache bedingte Anweisungen beziehungsweise Sprünge formulieren und so Fallunterscheidungen und verschiedene Formen von Schleifen in einem Programm umsetzen.

Steuerbus Verbindungssystem in einem ↑Rechner, über das Kontroll- und Statussignale zwischen ↑CPU, ↑Hauptspeicher und ↑Peripherie übermittelt werden.

Steuerung Vorgang in einem System, bei dem eine oder mehrere Größen als Eingangsgrößen andere Größen als Ausgangsgrößen aufgrund der dem System eigentümlichen Gesetzmäßigkeiten beeinflussen (DIN IEC 60050-351). Unterschied zur ↑Regelung ist der offene Wirkungsweg: Ausgangsgrößen wirken nicht auf Eingangsgrößen zurück, wodurch Störgrößen nicht ausgeglichen werden können. Gleichwohl unterliegt der Vorgang einer bestimmten ↑Echtzeitbedingung, die durch den zu steuernden physikalisch-technischen Prozess vorgegeben ist.

Störleitung Übertragungsweg (Kabel, Draht, Leiterbahn), auf dem das Signal für eine ↑Unterbrechungsanforderung von einem ↑Peripheriegerät zur ↑CPU gesendet wird.

Stoßbetrieb Verfahren zum ↑Speicherdirektzugriff. Die Steuereinheit (↑DMA *controller*) transferiert einen Block von ↑Daten in einem Stück, sobald ihr Zugriff auf den ↑Systembus gewährt wurde. Während des Transfers bleibt der ↑CPU der Zugriff auf den Systembus verwehrt, ihre Leistung wird wesentlich beeinträchtigt.

strikte Konsistenz Modell der ↑Speicherkonsistenz, nach dem jeder ↑Prozessor bei jeder Leseoperation aus dem ↑Arbeitsspeicher immer den Wert erhält, der von einem beliebigen (anderen) Prozessor zuletzt an dieselbe ↑Adresse geschrieben wurde.

Subjekt Individuum (↑Prozess), das ein ↑Zugriffsrecht auf ein ↑Objekt wünscht oder hat.

Superblock Satz von ↑Verwaltungsdaten, die ein ↑Dateisystem charakterisieren (↑UNIX).

superblock (dt.) ↑Superblock.

supervisor (dt.) ↑Hauptsteuerprogramm.

supervisor mode ↑*privileged mode.*

swap area (dt.) ↑Umlagerungsbereich.

swapping (dt.) ↑Umlagerung.

Symbol Sinnbild einer ↑Adresse, deren symbolhafte Bezeichnung.

symbolic link (dt.) ↑symbolische Verknüpfung.

symbolische Adresse ↑Adresse, die als ↑Symbol für ein Strukturmerkmal eines Programms (d.h., ein Sprungziel oder der Platzhalter einer Variablen oder Konstanten) steht und zeichenhaftig (mnemonisch) dargestellt ist. Das Symbol ist durch einen ↑Binder in eine Nummer aufzulösen, bevor ein direkter Zugriff über die damit verknüpfte Adresse möglich ist: eine ↑Bindung ist herzustellen. Diese Nummer ist ein Element im ↑Adressraum von dem ↑Prozess, der durch eben dieses Programm bestimmt ist und den Zugriff bewirkt. Dabei kann die Auflösung vor oder zur ↑Ladezeit beziehungsweise sogar erst zur ↑Laufzeit des Programms stattfinden, die Bindung ist damit statisch oder dynamisch.

symbolische Verknüpfung Paarung von ↑Dateiname und ↑Indexknoten durch eine ↑symbolische Adresse. Im Unterschied zur ↑Verknüpfung mit einer ↑Indexknotennummer wird ein ↑Pfadname genutzt, um die Beziehung zu einer ↑Datei zu erstellen. Der Pfadname ist selbst als spezielle Datei gespeichert, was durch ein Attribut im Indexknoten kenntlich gemacht ist. Bei der ↑Namensauflösung wird der zum Dateinamen gesuchte endgültige Indexknoten sodann über den gespeicherten Pfadnamen ermittelt. Damit lassen sich beliebige Verknüpfungen erstellen, insbesondere solche, die die sonst azyklische Struktur von dem ↑Dateibaum durchbrechen (indem ein ↑Verzeichnis adressiert wird) oder auf eine Datei in einem anderen (an einen ↑Befestigungspunkt eingehängtes) ↑Dateisystem verweisen.

symbolischer Maschinenkode Zeichenhaftige, mnemonische Darstellung von ↑Maschinenkode, indem jedem Befehl seine Bedeutung durch eine Merkhilfe als ↑Mnemon gegeben wird. Beispielsweise steht `ADD EAX` für die Addition mit dem 32-Bit breiten Akkumulator von einem ↑x86-kompatiblen ↑Prozessor.

Symboltabelle Datenstruktur, die jedem ↑Symbol in einem ↑Programm Attribute zuordnet, die dann vom ↑Binder ausgewertet werden. Die Attribute geben Hinweise beispielsweise auf Typ, Größe, Gültigkeitsbereich (lokal/global) und Lage (absolut/relativ) des mit dem Symbol bezeichneten Abschnitts von ↑Text/↑Daten eines Programms.

symmetric multiprocessing (dt.) ↑symmetrische Simultanverarbeitung.

symmetric multiprocessor (dt.) symmetrischer ↑Multiprozessor, auch ↑symmetrisches Multiprozessorsystem.

symmetric scheduling (dt.) ↑symmetrische Planung.

symmetrische Planung (en.) ↑*symmetric scheduling.* Modell der ↑Ablaufplanung für ein ↑symmetrisches Multiprozessorsystem. Anders als ↑asymmetrische Planung kann grundsätzlich jeder ↑Prozess auf jeden ↑Prozessor des Systems stattfinden, da ↑Homogenität der Hardware das grundlegende Prinzip bildet. Technisch basiert diese Form der Ablaufplanung auf eine allen Prozessoren gemeinsame ↑Bereitliste, das heißt, auf die diese Liste implementierende Datenstruktur wird von verschiedenen Prozessoren aus zugegriffen. Jeder dieser Prozessoren wird zur Ausführung des ↑Planers herangezogen, wodurch die damit erfolgende ↑Einplanung auch implizit jedem dieser Prozessoren zu Gute kommt und einen Prozess zur ↑Einlastung bereit stellt. Steht ein ↑Prozesswechsel an, wird sich (bildlich gesprochen) jeder Prozessor selbst um einen anderen Prozess bemühen. Demnach wird ein Prozessor nur dann in den

↑Leerlauf eintreten müssen, wenn insgesamt weniger ↑Aufgaben zur Bearbeitung anstehen als Prozessoren in dem System verfügbar sind.
Problematisch an diesem Modell ist die vergleichsweise intensive Kopplung gleichzeitiger Prozesse bei der Einplanung (befüllen/umsortieren der Bereitliste) und Einlastung (leeren der Bereitliste): alle Operationen, die den Zustand der Bereitliste veränderen können, unterliegen der ↑Synchronisation. Dies erzeugt ↑Interferenz wann immer Prozesse von verschiedenen Prozessoren ausgehend andere Prozesse bereitstellen (↑*scheduling*) oder abfertigen (↑*dispatching*) und beschränkt sich nicht nur auf die Momente der ↑Arbeitsteilung oder des ↑Arbeitsentzugs, wie etwa bei der asymmetrischen Planung. Letztere Formen der Arbeitsversorgung helfen zudem nur, um Imbalancen bei mehreren Bereitlisten auszugleichen und bleiben für eine allen gemeinsame Bereitliste ohne Bedeutung. Eine Abschwächung von Interferenzen bei konkurrierenden Listenoperationen ist hier nur noch durch Synchronisationsverfahren erreichbar, die möglichst wenig Störung auf die betroffenen Prozesse ausüben.

symmetrische Simultanverarbeitung (en.) ↑*symmetric multiprocessing*. ↑Betriebsart für zwei oder mehr gleiche (identische) Recheneinheiten, eng gekoppelt über ein gemeinsames Verbindungssystem. Typischerweise ist ein solches System von Recheneinheiten durch einen homogenen ↑Multiprozessor oder ↑Mehrkernprozessor repräsentiert. Auf jeder einzelnen Recheneinheit kann wenigstens ein ↑Prozess stattfinden, echt simultan mit Prozessen anderer Recheneinheiten. Zudem könnten einzelne oder alle Recheneinheiten einem ↑Multiplexverfahren unterzogen sein und somit jeweils mehr als einen Prozess pseudo-simultan ermöglichen.

symmetrisches Multiprozessorsystem (en.) ↑*symmetric multiprocessor* (↑SMP). Bezeichnung für ein ↑Rechensystem, das aus wenigstens einen ↑Multiprozessor oder ↑Mehrkernprozessor aufgebaut ist und dessen Recheneinheiten symmetrisch ausgelegt sind. Die Symmetrie ist in der ↑Homogenität der Hardware begründet, wodurch insbesondere ein für alle Prozessoren einheitlicher ↑Befehlssatz definiert und damit die Grundlage für ↑symmetrische Simultanverarbeitung gegeben ist: die Prozesse können auf jeden beliebigen Prozessor des Systems stattfinden (↑*symmetric scheduling*).

synchrone Ausnahme Bezeichnung für eine ↑Ausnahme, die von dem ↑Prozess selbst verursacht wurde. Bleiben das ↑Programm, das den Prozess in statischer Hinsicht bestimmt, und die Eingabedaten, die den Prozess in dynamischer Hinsicht bestimmen, unverändert, wird der Prozess immer wieder an derselben Stelle (↑Adresse in seinem ↑Adressraum) in dieselbe Falle (↑*trap*) tappen: die ↑Ausnahmesituation ist vorhersehbar und reproduzierbar. Beispiele dafür sind ein ungültiger ↑Maschinenbefehl, eine falsche ↑Adressierungsart, ein ↑Schutzfehler oder auch ein ↑Seitenfehler, nämlich wenn ↑virtueller Speicher eine ↑lokale Ersetzungsstrategie benutzt.

Synchronisation Ergebnis der ↑Synchronisierung. Jeder beteiligte ↑Prozess unterliegt der Koordination der Kooperation und Konkurrenz zwischen seinesgleichen.

Synchronisierung Vorgänge zeitlich aufeinander abstimmen, sie in einer bestimmten Reihenfolge stattfinden lassen. Die Maßnahmen dazu können eine ↑blockierende Synchronisation oder ↑nichtblockierende Synchronisation der betroffenen Vorgänge bewirken.

system level (dt.) ↑Systemebene.

system mode (dt.) Systemmodus: ↑Arbeitsmodus.

system time (dt.) ↑Systemzeit.

system-call dispatcher (dt.) ↑Systemaufrufzuteiler.

system-call stub (dt.) ↑Systemaufrufstumpf.

Systemaufruf Aufruf von einem im ↑Betriebssystem liegenden ↑Unterprogramm, initiiert durch eine in einem ↑Maschinenprogramm kodierte ↑Aktion.

Systemaufrufnummer ↑Operationskode, der den ↑Systemaufruf an ein ↑Betriebssystem identifiziert und normalerweise als fortlaufende Nummer dargestellt ist. Diese Nummer wird vom ↑Systemaufrufzuteiler auf eine interne Funktion des Betriebssystems abgebildet.

Systemaufrufparameter Argument für ein ↑Unterprogramm; hier: aktueller Parameter für einen ↑Systemaufruf und der ↑Systemfunktion, die die mit diesen Aufruf bezweckte Operation im Betriebssystem implementiert.

Systemaufrufstumpf Programmabschnitt, der einen ↑Systemaufruf absetzt und das dazu erforderliche Protokoll zwischen dem umfassenden ↑Maschinenprogramm und dem aufzurufenden Betriebssystem abwickelt. Der Systemaufruf wird kodiert, die ↑Systemaufrufparameter werden zur Übergabe ans Betriebssystem bereitgestellt, die ↑CPU wird angewiesen, die ↑partielle Interpretation des Maschinenprogramms durch das Betriebssystem zu ermöglichen und nach Rückkehr vom Systemaufruf wird auf eine ↑Ausnahmesituation geprüft. Dieser Programmabschnitt entspricht einer ↑Blattprozedur des Maschinenprogramms.

Systemaufrufzuteiler Gegenstück zu einem ↑Systemaufrufstumpf; Programmabschnitt im ↑Betriebssystem, der die ↑Aktionsfolge festlegt, um einen ↑Systemaufruf anzunehmen und abzuwickeln: den Systemaufruf total und das ↑Maschinenprogramm partiell zu interpretieren. Das bedeutet, erstens, den Systemaufruf dekodieren und auf Gültigkeit prüfen. Ein ungültiger Systemaufruf begründet eine ↑Ausnahmesituation. Zweitens, die Weitergabe der ↑Systemaufrufparameter an das aufzurufende ↑Unterprogramm, welches die angeforderte Systemfunktion implementiert. Abhängig vom ↑Operationsprinzip des Betriebssystems erfolgt die Überprüfung der Parameter hier bei ihrer Weitergabe oder später im Moment ihrer Verwendung. Ungültige Parameter begründen eine Ausnahmesituation. Drittens, die durch den Systemaufuf ausgelöste ↑partielle Interpretation des Maschinenprogramms beenden und die ↑CPU anweisen, die Interpretation des Maschinenprogramms wieder aufzunehmen. Dieser Programmabschnitt ist eine ↑Wurzelprozedur des Betriebssystems.

Systembus Gesamtheit der in einem ↑Rechner vorhandenen Sammelschienen, über die ↑CPU, ↑Hauptspeicher und ↑Peripherie gekoppelt sind: nämlich der ↑Adressbus, ↑Datenbus und ↑Steuerbus.

Systemebene (en.) ↑*system level*. Bezeichnung für die Stufe, sowohl im Sinne von Privilegien (↑*privileged mode*) als auch in Bezug den Kontext (↑Betriebssystem), auf der ein ↑Prozess gegenwärtig stattfindet.

Systemfunktion ↑Unterprogramm im ↑Betriebssystem. Ein solches ↑Programm implementiert beispielsweise die durch einen ↑Systemaufruf vom ↑Maschinenprogramm angeforderte Operation. Im Falle von ↑UNIX ist für jede mit `man(1)` unter Abschnitt 2 „*System calls*“ gelistete Funktion wenigstens ein Unterprogramm des Betriebssystems verknüpft. Darüberhinaus enthält ein Betriebssystem Unterprogramme, die nicht als Folge von Systemaufrufen zur Ausführung kommen, also ihren Ursprung in einer ↑Aktion des Maschinenprogramms haben, sondern bei denen der Aufruf originär von der Hardware ausgelöst wird, nämlich bei einer ↑Unterbrechungsanforderung oder allgemein aufgrund einer ↑Ausnahmesituation. Ferner kann ein im Betriebssystem autonom stattfindender ↑Prozess, eventuell sogar mehrere davon, als ↑Dämon Anlass für solche Aufrufe sein.

Systemkernfaden Bezeichnung für einen ↑Faden im ↑Maschinenprogramm, der durch eine eigene ↑Prozessinkarnation im ↑Betriebssystemkern implementiert ist. Sämtliche für solch einen Faden erforderlichen Betriebsmittel sowie für seine Verwaltung nötigen Funktionen stellt das ↑Betriebssystem. Dies betrifft den ↑Laufzeitkontext des Fadens und die Funktionen zur ↑Ablaufplanung, ↑Einlastung und ↑Synchronisation: All diese Funktionen, insbesondere auch der ↑Prozesswechsel, verlaufen unter Kontrolle des Betriebssystemkerns und erfordern daher einen ↑Systemaufruf, wenn innerhalb des Maschinenprogramms vom aktuellen zu einem anderen Faden umgeschaltet werden soll. Damit ist jeder dieser Fäden aus Sichtweise des

Maschinenprogramms vom Bautyp her ein ↑leichtgewichtiger Prozess, da (im Gegensatz zum ↑Anwendungsfaden) zwar vergleichsweise aufwändige Systemaufrufe zu seiner Kontrolle erforderlich sind, jedoch bei Fadenwechsel im selben Maschinenprogramm derselbe gegebenenfalls unter ↑Speicherschutz stehende ↑Adressraum aktiv bleibt. Für das Betriebssystem ist ein solcher Faden ein ↑Objekt erster Klasse, das heißt, der durch den Faden konkretisierte ↑Prozess ist insbesondere auch dem Betriebssystemkern bekannt. Eine vom Maschinenprogramm aus beanspruchte ↑Systemfunktion läuft damit immer im Namen dieses Fadens ab. Wird dieser dann durch die Systemfunktion blockiert (↑Prozesszustand), kann der Betriebssystemkern von selbst zu einen anderen Faden dieser Art im selben Maschinenprogramm umschalten.

Systemkonsole Ausrüstung, mit der die manuelle Steuerung und Überwachung von Vorgängen in einem ↑Rechner möglich ist (↑*operator panel*). Anfänglich ein aus vielen Schaltern und verschiedenen Lampenfeldern bestehendes Gerät, über das neben der Eingabe von Kommandos zum Starten, Laden, Anhalten, Fortsetzen und Herunterfahren des Rechners auch bitweise Inhalte der ↑Prozessorregister und vom ↑Hauptspeicher angezeigt und verändert werden konnten sowie allgemein der Systemzustand abgerufen wurde. Zwischenzeitlich auch ergänzt um ein schreibmaschinenähnliches ↑Peripheriegerät (↑TTY) mit ↑Tabellierpapier oder eine spezielle ↑Dialogstation, worüber der Rechner seine Anweisungen in Form einer ↑Kommandozeile erhielt. Seit der ↑PC Einzug gehalten hat auch Inbegriff für die Kombination aus Datensicht- und -eingabegerät (↑*terminal*), über das der Rechner gesteuert und überwacht wird.

Systemprogrammiersprache Bezeichnung für eine Hochsprache zur ↑Systemprogrammierung. Eine formale Programmiersprache, mit deren Sprachkonstrukte ein ↑Programm zur gezielten Kontrolle der Hardware (↑CPU, ↑Peripherie, ↑Speicher) ausgedrückt werden kann. Zweck einer solchen Programmiersprache ist bewusst nicht die Abstraktion von der Hardware, sondern die problemorientierte und schon auf höherer Abstraktionsebene stattfindende Formulierung von Algorithmen und Datenstrukturen bei direkter Durchgriffsmöglichkeit auf die Hardware (↑ISA) in funktionaler (↑Maschinenbefehl) und nichtfunktionaler (↑Programmiermodell, ↑reale Adresse, ↑Ausrichtung) Hinsicht. Eine ↑Assembliersprache zählt für gewöhnlich nicht in diese Kategorie, da sie ausschließlich — dafür aber allumfassend — lediglich den Durchgriff auf die Hardware ermöglicht. Von den höheren Programmiersprachen ist beispielsweise ↑C oder ↑C++ dieser Kategorie zuzurechnen, obgleich in einigen Fällen (↑Ausnahmebehandlung, ↑Prozesswechsel, ↑Synchronisation) mangels geeigneter Sprachkonstrukte vereinzelt auf Assembliersprache zurückgregriffen werden muss. Eine Hilfskonstruktion (*workaround*) dabei ist oftmals, Assembliersprachenanweisungen entweder inzeilig (*inline*) an den benötigten Stellen im Hochsprachenprogramm zu kodieren oder als getrennt zu übersetzendes ↑Unterprogramm dem Hochsprachenprogramm zum Aufruf beiseitezustellen. Sind die hardwarenahen Operationen ausschließlich durch (in Assembliersprache formulierte, getrennt übersetzte) Unterprogramme zugänglich, handelt es sich bei der Programmiersprache, in der das diese Unterprogramme aufrufen müssende ↑Hauptprogramm formuliert ist, im Allgemeinen nicht um eine Hochsprache zur Systemprogrammierung.

Systemprogrammierung Programmierung von ↑Systemsoftware. Diese Software ist entweder Teil von einem ↑Betriebssystem oder muss in engem Zusammenspiel mit dem Betriebssystem und der Hardware (↑CPU, ↑Peripherie) funktionieren. Zur Formulierung der Systemsoftware kommt für gewöhnlich eine ↑Systemprogrammiersprache zum Einsatz. Gegenstand eines diesbezüglichen Lehrgebiets ist der Aufbau, die Struktur und die Funktionsweise von Software, die als ↑Programm hilft, ein ↑Rechensystem für einen bestimmten Anwendungszweck zu betreiben. Dabei kann dieses Programm für sich allein genommen durchaus ziemlich wirkungslos sein: es kommt möglicherweise erst dann zur Geltung, wenn es als ↑Unterprogramm (Systemsoftware) in ein ↑Hauptprogramm (Anwendungssoftware) eingebunden ist und von dort aufgerufen werden muss. Typisches Beispiel dafür ist eine ↑Programmbibliothek, deren Softwarebestände die Schnittstelle zu einem Betriebssystem darstellen.

Systemsoftware Gesamtheit von in Software vorliegender Programme, die eine ↑Rechenanlage betriebsbereit machen (in Anlehnung an den Duden). Dazu zählt insbesondere das ↑Betriebssystem, aber auch die mit einem Betriebssystem typischerweise ausgelieferte ↑Programmbibliothek ist hier eingeschlossen. Typisches Beispiel für letzteres ist die ↑libc, die für gewöhnlich die Programmierschnittstelle zu einem Betriebssystem (z.B. ↑UNIX) bereitstellt. Allgemein wird darunter all jene Software zusammengefasst, die die Grundlage für die Ausführung von Anwendungsprogrammen legt.

Systemsteuerung Gesamtheit der zur ↑Steuerung der Abläufe in einem ↑Rechensystem erforderlichen technischen Bestandteile in Bezug auf einen bestimmten ↑Arbeitsmodus.

Systemzeit (en.) ↑*system time*. Bezeichnung für die Zeitspanne, die ein ↑Prozess im ↑Betriebssystem (↑*system mode*) verbracht hat. Ein über die gesamte Lebenszeit eines Prozesses akkumulierter Wert. Gegensatz zur ↑Benutzerzeit.

Tabellierpapier Endlosdruckpapier; Papier, das keine Einzelblätter aufweist, sondern eine lange Bahn von (bis zu 2000 Stück) durch Perforation voneinander getrennter Blätter bildet.

Taktentzug Verfahren zum ↑Speicherdirektzugriff. Die Steuereinheit (↑DMA *controller*) transferiert einen Block von ↑Daten in Einzelschritten, für die sie jeweils den ↑Systembus nur kurzzeitig belegt: einen Buszyklus „unbemerkt“ in ihren Besitz bringt (↑*cycle stealing*). Zwischen den Einzelschritten hat die CPU Zugriff auf den Systembus; jedoch wird ihre Leistung während des laufenden Transfers beeinträchtigt, wenn sie auf den Bus zugreifen muss.

Taktfrequenz Geschwindigkeit, mit der ↑Daten in der ↑CPU verarbeitet werden können; auch Taktung oder Taktrate von einem ↑Prozessor. Frequenz, mit der die CPU ihre Steuerbefehle erteilen kann. Angabe in ↑Hertz und zumeist mit einem Präfix versehen, der eine ganzzahlige Vielfache von 10^3 für diese Einheit angibt: Kilo-, Mega-, Gigahertz.

task (dt.) ↑Aufgabe.

tatsächlicher Parameter (en.) ↑*actual parameter*. Argument, das einem ↑Unterprogramm beim Aufruf tatsächlich übergeben wird. Ein dazu korrespondierender ↑formaler Parameter bestimmt den Typ des Wertes, der übergeben werden kann.

TCP Abkürzung für (en.) *Transmission Control Protocol*, seit 1974. Ermöglicht verbindungsorientierte, zuverlässige, gesicherte und geschützte Kommunikation von Nachrichtenströmen variabler Länge oberhalb von ↑IP. Zusatzfunktionen sind (1) Multiplexen der transferierten ↑Daten, (2) Handhabung und Verpackung der Daten als Nachrichten, (3) Transfer dieser Nachrichten, (4) Bereitstellung einer gewissen Dienstgüte und von Zuverlässigkeit sowie (5) Flusskontrolle und Stauvermeidung.

Team Gruppe von Vorgängen an einer gemeinsamen ↑Aufgabe. Logisch entspricht jeder Vorgang einem ↑Prozess, der physisch als ↑Faden im ↑Maschinenprogramm in Erscheinung tritt und im ↑Betriebssystem durch eine ↑Ablaufplanung kontrolliert wird. Wesentliches Merkmal ist die Repräsentation als ↑nichtsequentielles Programm, das die Basis für den gemeinsamen ↑Adressraum der Fäden legt. Unwesentlich ist die Repräsentation des Fadens im Betriebssystem, solange sichergestellt ist, dass ein Faden in einer sich für ihn abzeichnenden Entwicklung zuvorkommend (*präemptiv*) zur Ausführung gelangen kann. Letzteres bedeutet keinesfalls, den Faden als ↑Systemkernfaden implementieren zu müssen. Andere Formen sind ebenso möglich, beispielsweise eine ↑Fortsetzung.

Teilaufgabe Teil einer ↑Aufgabe.

Teilhaberbetrieb Bezeichnung für eine ↑Betriebsart, bei der ein ↑Dialogprozess über mehr als eine ↑Dialogstation den Rechnerbetrieb führt. Für gewöhnlich bietet die Dialogstation einem Menschen den Zugang zum ↑Rechensystem. Im Gegensatz zum ↑Teilnehmerbetrieb erfüllt

der Dialogprozess typischerweise nur einen bestimmten ↑Dienst, jedoch können mehrere solcher Prozesse verschiedener Dienste zugleich bereitstehen. Der Dialogprozess besteht über die Dauer einer ↑Sitzung hinweg, er führt mehrere Sitzungen gegebenenfalls verschiedener Teilhaber nacheinander.

Teilhabersystem Bezeichnung von einem ↑Rechensystem für ↑Teilhaberbetrieb.

Teilinterpretation Synonym zu ↑partielle Interpretation.

Teilnehmerbetrieb Bezeichnung für eine ↑Betriebsart, bei der wenigstens ein ↑Dialogprozess pro ↑Dialogstation den Rechnerbetrieb führt. Für gewöhnlich bietet die Dialogstation einem Menschen den Zugang zum ↑Rechensystem. Im Gegensatz zum ↑Teilhaberbetrieb erfüllt der Dialogprozess keinen bestimmten ↑Dienst, sondern bildet lediglich die „Außenhaut“ (↑*shell*) des Rechensystems, um einen ↑Kommandointerpreter mit Anweisungen zu versorgen. Der Dialogprozess besteht nur für die Dauer einer ↑Sitzung.

Teilnehmersystem Bezeichnung von einem ↑Rechensystem für ↑Teilnehmerbetrieb.

Teilvirtualisierung Synonym zu ↑partielle Virtualisierung.

Termin (en.) ↑*deadline*. Festgelegter absoluter oder relativer Zeitpunkt, bis zu dem oder an dem etwas geschehen soll (in Anlehnung an den Duden).

terminal (dt.) Datenstation, Endgerät; ↑Dialogstation.

Termineinhaltung Vereinbarung eines festgelegten Zeitpunkts, bis zu dem ein Berechnungsergebnis vorliegen soll oder muss. Der Termin ist typischerweise abgestuft klassifiziert wie folgt: weich (schwach), wenn die Terminverletzung tolerierbar ist, mit jeder weiteren Verzögerung das Berechnungsergebnis aber an Wert verliert; fest, wenn die Terminverletzung tolerierbar ist, das Berechnungsergebnis aber keinen Wert mehr hat und die ↑Aufgabe daher abgebrochen werden muss; hart (strikt), wenn die Terminverletzung wegen sonst drohender Gefahr für Leib und Leben oder hohem Risiko für wirtschaftlichen Verlust nicht tolerierbar ist, eine ↑Ausnahmesituation, auf die die Anwendung rechtzeitig reagieren können muss. Charakteristisches Merkmal für ein ↑Maschinenprogramm, das unter ↑Echtzeit ablaufen muss.

Tertiärspeicher Klassifikation für den ↑Archivspeicher; „drittrangiger Speicher“, der zwar über eine enorm große Kapazität verfügt, dafür jedoch auch eine sehr hohe ↑Zugriffszeit mit sich bringt.

Text Fassung von Befehlen im ↑Programm; inhaltlich zusammenhängende Folge von Anweisungen (in Anlehnung an den Duden). Oft auch als Kode (*code*) bezeichnet, der nach Übersetzung des Programms letztlich in Form von ↑Maschinenkode vorliegt und damit die von einem ↑Prozessor auszuführenden Befehle, nicht jedoch die zu verarbeitenden ↑Daten, beschreibt.

text (dt.) ↑Text, Wortlaut von Instruktionen.

Textsegment Bezeichnung für ein ↑Segment, das den ↑Text von einem ↑Programm speichert.

thrashing (dt.) ↑Seitenflattern.

thread (dt.) ↑Faden.

through-put time (dt.) ↑Durchlaufzeit.

tile (dt.) Kachel, Fliese; hier im Sinne von ↑*tile processor*.

tile processor Bezeichnung für einen ↑Mehrkernprozessor, der aus mehreren (norm. identischen, aber auch verschiedenartig strukturierten/aufgebauten) integrierten Kacheln (↑*tile*) besteht, wobei jede dieser Kacheln wenigstens eine oder mehrere Verarbeitungseinheiten (d.h., jeweils

ein ↑Rechenkern), einen ↑Zwischenspeicher und eine Leitungs- oder Speichervermittlungsstelle (*switch*) umfasst. Innerhalb des durch eine solche Kachel abgeschlossenen Gebiets ist für gewöhnlich ↑Speicherkohärenz sichergestellt.

time series (dt.) ↑Zeitreihe.

time series analysis (dt.) ↑Zeitreihenanalyse.

time sharing (dt.) ↑Teilnehmerbetrieb.

time slice (dt.) ↑Zeitscheibe.

time-triggered system (dt.) zeitgesteuertes System. Ein ↑Rechensystem dessen ↑Echtzeitbetrieb jede anstehende ↑Aufgabe zu dem jeweils für sie fest vorgegebenen Zeitpunkt durchführt. Auch bezeichnet als taktgesteuertes System.

timer (dt.) ↑Zeitgeber.

TLB Abkürzung für (en.) *translation lookaside buffer*, (dt.) ↑Übersetzungspuffer.

Trägerleiterplatte Platine, die als Träger für elektronische Bauteile dient und dazu über geeignete mechanische Befestigungen (Sockel) und elektrische Verbindungen verfügt. Typisches Beispiel ist die ↑Hauptplatine in einem ↑Rechner.

transaction mode (dt.) ↑Teilhaberbetrieb.

Transaktion ↑Aktion oder ↑Aktionsfolge, die eine *logische Einheit* bildet und als Ganzes entweder gelingt oder scheitert. Gelingt sie, wurde die Berechnung vollständig korrekt durchgeführt und der damit verbundene Datenbestand im konsistenten Zustand hinterlassen. Scheitert sie, bleibt die Aktion/Aktionsfolge ohne Auswirkung, als wenn sie nie stattgefunden hätte.

transparent mode (dt.) ↑Transparentbetrieb.

Transparentbetrieb Verfahren zum ↑Speicherdirektzugriff. Die Steuereinheit (↑DMA *controller*) transferiert einen Block von ↑Daten in Teilen, nämlich wenn die ↑CPU den ↑Systembus nicht benötigt. Die CPU wird in ihrer Leistung nicht beeinträchtigt.

trap (dt.) Fangstelle, abfangen; ↑Abfangung.

Trommeladresse Bezeichnung für eine ↑Adresse im ↑Trommelspeicher.

Trommelmaschine ↑Rechner, bei dem der ↑Hauptspeicher ausschließlich aus ↑Trommelspeicher besteht. Solche Rechner wurden bis etwa 1970 hergestellt.

Trommelspeicher ↑Peripheriegerät zur elektromagnetischen Aufzeichnung und Wiedergabe von ↑Daten. Die Außenseite eines im Betrieb schnell rotierenden Metallzylinders ist mit ferromagnetischem Material beschichtet, das der eigentlichen Informationsspeicherung dient. Der Zylinder hat mehrere Spuren und pro Spur ist wenigstens ein eigener (oftmals feststehender) Lese-/Schreibkopf vorhanden, was eine mittlere ↑Zugriffszeit (bereits in den 1950er Jahren) um 10 ms ermöglicht. Ursprünglich der ↑Hauptspeicher in einem ↑Rechner (↑*drum machine*), wobei dann der ↑Maschinenbefehl die ↑Trommeladresse des jeweiligen Operanden enthält. Später als Erweiterung für die Zwischenspeicherung von Daten oder Teilen einer ↑Programmbibliothek, bis in die 1980er Jahre auch zur schnellen ↑Umlagerung von ↑Text und Daten (z.B. in ↑UNIX: `/dev/drum`) in Gebrauch (↑*swapping*). Im Falle von Erweiterungsspeicher besorgt ein ↑Gerätetreiber die Ein- oder Ausgabe mittels ↑Ein-/Ausgaberegister, die nebst Daten auch die Trommeladresse enthalten beziehungsweise dem Gerät übermitteln. Der Gerätetreiber ist für gewöhnlich ein ↑Unterprogramm von einem Steuerprogramm zur Organisation und Überwachung des Rechnerbetriebs (↑*resident monitor*).

TTY Abkürzung für (en.) *teletypewriter*, (dt.) ↑Fernschreiber.

Typfehler Folge einer ↑Typverletzung.

Typsicherheit Ausmaß der Verhinderung von ↑Typfehler durch eine Programmiersprache beziehungsweise einen ↑Kompilierer. Die Maßnahmen dazu greifen statisch (bei der ↑Kompilation), dynamisch (zur ↑Laufzeit) oder in Kombination. Je nach Stärke der Typisierung der Programmiersprache wird der Kompilierer mehr oder weniger viel Typkonflikte unbeanstandet durchlassen, die dann jedoch durch generierte Anweisungen zur Typkontrolle zur Laufzeit abgefangen werden und als ↑Ausnahmesituation enden.

Typverletzung Unstimmigkeit bei Operationen auf verschiedenen Datentypen. Beispielsweise eine Ganzzahl (`int`) als Zeiger auf ein Zeichen (`char*`) oder als Universalzeiger (`void*`) behandeln — dadurch eine beliebige ↑Adresse konstruieren und (logisch) unautorisiert auf den ↑Arbeitsspeicher zugreifen können.

UDP Abkürzung für (en.) *User Datagram Protocol*, seit 1980. Ermöglicht verbindungslose, unzuverlässige, ungesicherte und ungeschützte Kommunikation in einem ↑Netzwerk. Das Protokoll verwendet Prüfsummen zur Überprüfung der Integrität transferierter ↑Daten und Anschlussnummern (*port number*), um darüber verschiedene Funktionen im Quell- und Zielknoten eines Datagramms zu adressieren.

Überlagerung Programmteil (↑Text oder ↑Daten), dessen Platz im ↑Hauptspeicher wechselweise über die Zeit von mehreren solcher Teile gemeinsam belegt wird.

Überlagerungsspeicher Bereich in einem peripheren ↑Speicher, in dem ↑Überlagerung eines Programms abgelegt sind.

überlappte Ein-/Ausgabe ↑Betriebsart, bei der Ein-/Ausgabe die damit im Zusammenhang stehende oder eine beliebige andere Berechnung zeitlich überlagert. Dazu setzt ein ↑Prozess zunächst eine Ein-/Ausgabeoperation an das ↑Betriebssystem ab. Im weiteren Verlauf löst diese Operation, effektiv verursacht durch einen ↑Gerätetreiber, einen ↑Ein-/Ausgabestoß bei einem ↑Peripheriegerät aus. Dieser Stoß kann gleichzeitig mit dem jeweils von der ↑CPU generierten ↑Rechenstoß eines Prozesses stattfinden. Wartet der Prozess, der die Ein-/Ausgabe angestoßen hat, nicht auf die Beendigung der dafür ursächlichen Operation, überlagern sich der durch ihn selbst herbeigeführte Rechen- und Ein-/Ausgabestoß. Begeht der Prozess dagegen ↑passives Warten, führt dies für gewöhnlich zum ↑Prozesswechsel und damit dazu, dass sich Rechen- und Ein-/Ausgabestöße, die aus verschiedenen Prozessen resultieren, zeitlich überlagern können. Da die Ein-/Ausgabe unabhängig von dem Prozess stattfindet, der die Operation dazu abgesetzt hat, sendet das betreffende Peripheriegerät eine ↑Unterbrechungsanforderung an die CPU, um dem Betriebssystem die Beendigung der entsprechenden Operation anzuzeigen. Aus demselben Grund erfolgt der Transfer ein- oder auszugebender ↑Daten durch ↑Speicherdirektzugriff.
Allgemein betrachtet tragen diese Maßnahmen wesentlich zur Steigerung von ↑Durchsatz bei, sie bewirken allerdings auch ↑Interferenz: Prozesse werden verzögert oder unterbrochen, die nichts mit der laufenden oder beendeten Ein-/Ausgabeoperation zu tun haben müssen. Wegen der strikt asynchronen Vorgehensweise ist der Zeitpunkt oder die Zeitspanne der Beeinflussung zudem unvorhersehbar für die Prozesse. Je nach Art eines Prozesses ist diese Eigenschaft mehr oder weniger kritisch. Alle Prozesse, für die eine bestimmte ↑Termineinhaltung gilt, sind davon betroffen. Für gewöhnlich kann diese Beeinflussung seitens der ↑Peripherie nicht unterbunden werden, was insbesondere auf ↑DMA zutrifft: auch wenn die Technik selbst einen Prozess nicht unterbricht, trägt sie (mit Ausnahme von ↑Transparentbetrieb) durch abknapsen von ↑Speicherbandbreite jedoch zu seiner Verzögerung bei, wenn er zugleich auf den ↑Hauptspeicher zugreift. Für Prozesse, die eine solche Beeinflussung nicht tolerieren, muss das Betriebssystem Vorkehrungen treffen, die einen vorhersehbaren Verlauf zusichern. Das kann im Extremfall bedeuten, Unterbrechungen oder DMA phasenweise oder für „sensitive Prozesse“ ganz zu unterbinden.

Übersetzer ↑Dienstprogramm, das ein in einer bestimmten Sprache formuliertes ↑Programm in ein semantisch äquivalentes Programm einer anderen Sprache umwandelt. Typisch dafür sind ↑Kompilierer und ↑Assemblierer.

Übersetzung Bezeichnung für die ↑Kompilation oder ↑Assemblierung von einem ↑Programm. In beiden Fällen erfolgt die Umwandlung eines in Quellsprache (z.B. ↑C oder ↑ASM86) vorliegenden Programms in eine bestimmte Zielsprache (z.B. ↑ASM86 oder ↑Maschinensprache).

Übersetzungseinheit Quellmodul für einen ↑Übersetzer, seine Eingabe.

Übersetzungspuffer Funktionseinheit zur schnellen ↑Adressabbildung, integriert in einer ↑MMU oder ↑CPU. Beim Zugriff auf den ↑Arbeitsspeicher ermöglicht diese Einheit mit einem kurzen „Blick zur Seite“ (*look aside*) festzustellen, ob eine von der CPU verwendete ↑logische Adresse direkt in eine ↑reale Adresse übersetzt werden kann. Die für die Übersetzung erforderlichen Hinweise werden entweder von der MMU oder dem ↑Betriebssystem in dem Puffer (↑TLB) zwischengespeichert.

UID Abkürzung für (en.) *user identifier*, (dt.) ↑Benutzerkennung.

Umgebungsrauschen (en.) ↑*ambient noise*. Synonym zu ↑Hintergrundrauschen.

Umlagerung (en.) ↑*swapping*. Übertragung von Speicherinhalten zwischen verschiedenen Ebenen der ↑Speicherhierarchie in einem ↑Rechensystem. Typisch dafür ist etwa die Auslagerung von einem kompletten ↑Prozess vom ↑Vordergrundspeicher in den ↑Hintergrundspeicher beziehungsweise umgekehrt die Einlagerung.

Umlagerungsbereich Bereich im ↑Ablagespeicher, der vom ↑Betriebssystem für die Sicherungskopie (*backup*) des Inhalts von einem ↑Prozessadressraum vorgesehen ist. ↑Hintergrundspeicher, ohne dem ↑virtueller Speicher für einen ↑Prozess nicht funktioniert. Die Größe dieses Bereichs richtet sich nach der maximalen Anzahl, der zu einem Zeitpunkt vom Betriebssystem unterstützten Prozesse und der maximalen Größe des Adressraums eines jeden dieser Prozesse (↑schwergewichtiger Prozess). Aus diesem Bereich werden nur die Anteile von ↑Text und ↑Daten eines Prozesses in den ↑Hauptspeicher kopiert (umgelagert), die für sein effizientes Vorankommen gerade benötigt werden (↑*working set*). Umgekehrt werden bei Verdrängung aus dem Hauptspeicher nur die Anteile in diesen Bereich zurück kopiert (umgelagert), die von dem Prozess seit ihrer Einlagerung zwischenzeitlich verändert worden sind.

Umlagerungsmittel Handhabe zur ↑Umlagerung von ↑Text oder ↑Daten zwischen ↑Vordergrundspeicher und ↑Hintergrundspeicher. Die Text-/Datenbestände sind in den Strukturierungselementen von einem ↑Prozessadressraum zusammengefasst, das heißt, sie liegen entweder in einer ↑Seite oder in einem ↑Segment.

Umlaufsperre Bezeichnung für eine Sperre (*lock*), deren Gültigkeit von einem ↑Prozess durch Kreiseln (*spin*) anhaltend überprüft wird; ↑blockierende Synchronisation. Der Prozess betreibt ↑geschäftiges Warten, bis die Sperre aufgehoben ist. Ein Verfahren zur ↑Synchronisation von Prozessen auf einem ↑Multiprozessor oder ↑Mehrkernprozessor. Dabei sollte darauf geachtet werden, dass jeder der kreiselnden Prozesse auf einen eigenen ↑Prozessor stattfindet und der die Sperre haltende Prozess keine ↑Verdrängung erfährt. Ansonsten drohen erhebliche Leistungseinbußen durch die zusätzliche Verzögerung der Prozesse anderer Prozessoren durch einen einzelnen (von seinem Prozessor verdrängten) Prozess.

Umplanung (en.) ↑*rescheduling*. Abänderung eines existierenden Plans, nach dem ein ↑Auftrag einem ↑Prozessor zur Verarbeitung übergeben werden soll. Typischer Fall ist ein ↑Ablaufplan, der zur ↑Laufzeit an geänderten Randbedingungen angepasst werden muss. Dadurch kann ein bereits eingeplanter ↑Prozess nunmehr früher oder später für die ↑Einlastung der ↑CPU in Frage kommen. Voraussetzung dafür jedoch ist die ↑mitlaufende Planung, um überhaupt auf sich dynamisch ergebende, für gewöhnlich auch unvorhersehbare und Prozesse beeinflussende ↑Ereignisse reagieren zu können. Beispiele dafür sind ↑HRRN, ↑SRTF, ↑VRR und ↑FB

beziehungsweise ↑MLFQ.
Zu beachten ist, dass umplanende Verfahren nicht zwingend auch die ↑Verdrängung der Prozesse von ihrem Prozessor zur Folge haben müssen. Zuallererst steht die Neuordnung der ↑Bereitliste entsprechend aktueller Randbedingungen im Vordergrund. Gekoppelt mit einem Verfahren, das ↑präemptive Planung verfolgt, kann der die Liste anführende Prozess abschließend dann Vorzug vor dem gegenwärtig auf der CPU stattfindenden Prozess genießen und diesen verdrängen.

Umschalter (en.) ↑*dispatcher*. Bezeichnung für ein ↑Programm, das die ↑Systemfunktion zum ↑Prozesswechsel implementiert.

unaufgelöste Referenz Bezeichnung für eine ↑Referenz, deren ↑Adresse noch unbekannt ist. Eine solche Referenz wird beim ↑Binden gelöscht und durch eine ↑Bindung ersetzt.

Unicode Universalzeichensatz (1991), bei dem ein bis vier ↑Byte zur Kodierung eines einzelnen Zeichens verwendet werden. Unterteilt in 17 Ebenen, mit jeweils bis zu 2^{16} Zeichen. Obermenge von ↑ASCII, die die ersten 128 Zeichen entsprechend definiert.

unilaterale Synchronisation Bezeichnung einer ↑Synchronisation, die sich nur in einer bestimmten Rolle blockierend für einen ↑Prozess auswirken kann; auch ↑logische Synchronisation oder ↑Bedingungssynchronisation. Zwischen den Prozessen besteht eine Erzeuger-Verbraucher-Beziehung in Bezug auf ein ↑konsumierbares Betriebsmittel. Der Erzeugerprozess zeigt das ↑Ereignis an, ein Betriebsmittel zur Verfügung gestellt zu haben und muss zur Durchführung dieser Anzeige (Signalisierung) selbst nicht warten. Demgegenüber ist der Verbraucherprozess in seinem Vorankommen von einem solchen Ereignis abhängig. Er wird blockieren, wenn das Ereignis (Signalisierung) noch nicht stattgefunden hat. Solch ein Muster der Kooperation von Prozessen, formuliert als ↑nichtsequentielles Programm, definiert die konsequente ↑Aktionsfolge für einen bestimmten Sachzusammenhang, das daher auch als logische Synchronisation bezeichnet wird. Darüberhinaus ist ein derartiges Ereignis auch Beispiel für die Entkräftung der Wartebedingung eines Verbraucherprozesses, der mit dieser Betrachtungsweise dann der Bedingungssynchronisation unterliegt. Grundlage für dieses Muster der Synchronisation bildet ein ↑allgemeiner Semaphor, der das konsumierbare Betriebsmittel repräsentiert. Die Anzahl der mit ↑V durch den Erzeugerprozess freigesetzten Einheiten diese Betriebsmittels ergibt sich durch den positiven Wert des Semaphors. Mit ↑P wird diese Anzahl durch den Verbraucherprozess reduziert, er konsumiert eine entsprechende Einheit des Betriebsmittels. Der Absolutwert eines negativen Werts des Semaphors gibt die Anzahl von Betriebsmitteleinheiten an, auf deren Bereitstellung Verbraucherprozesse warten.

uninterruptible mode (dt.) ↑unterbrechungsfreier Betrieb.

uniprocessor (dt.) ↑Uniprozessor.

uniprogramming (dt.) ↑Einprogrammbetrieb.

Uniprozessor (en.) ↑*uniprocessor*. Bezeichnung für einen ↑Prozessor, der nur aus einer ↑CPU besteht beziehungsweise einen einzigen ↑Rechenkern enthält. Für die ↑Parallelverarbeitung von ↑Prozessen ist damit die zeitliche Partitionierung des Prozessors (↑partielle Virtualisierung) Voraussetzung. Überhaupt lässt sich ↑Nebenläufigkeit dann nur mit Hilfe von ↑Unterbrechungsanforderungen zum Ausdruck bringen.

Universalbetriebssystem Bezeichnung für ein ↑Betriebssystem, das allgemein und umfassend eingesetzt werden kann und verschiedenen Anwendungszwecken dient; Gegenteil von ↑Spezialbetriebssystem. Trugschluss ist, dass ein solches Betriebssystem allen Anwendungszwecken gleichermaßen gut dienen kann. Vielmehr wird im Allgemeinen auf Kompromisslösungen etwa bei der ↑Betriebsmittelverwaltung zurückgegriffen, die voraussichtlich für viele Anwendungsbereiche akzeptable Leistungen von dem ↑Rechensystem erwarten lassen. Das schließt

ein umfassendes Funktionsangebot ein, auch wenn nicht alle Funktionen von jeder Anwendung zwingend erfordert werden. Typische Beispiele dafür sind etwa ↑virtueller Speicher, ↑Verdrängung, ↑Nachrichtenversenden, ↑Virtualisierung, ↑Schutz, aber auch das ↑Dateisystem oder der ↑Mehrprogrammbetrieb. Daraus resultierende latente, nichtfunktionale Merkmale eines Rechensystems in zeitlicher (z.B. ↑Latenzzeit oder ↑Laufzeit) wie auch räumlicher (z.B. Platzbedarf im ↑Hauptspeicher) Hinsicht können Anwendungen sogar verhindern.

UNIX Mehrplatz-/Mehrbenutzerbetriebssystem. Erste Installation im Jahr 1969 (DEC PDP-7), zunächst programmiert in ↑Assembliersprache, später (1972) umgearbeitet unter Verwendung von ↑C. In der Anfangsphase (1969) unter dem Namen UNICS geführt, als Abkürzung für (en.) „*Uniplexed Information and Computing Service*“ und Wortspiel in Bezug auf ↑Multics. Ursprung vieler Betriebssystementwicklungen, insbesondere ↑Linux und ↑Darwin.

unprivileged mode ↑*user mode*.

unresolved reference (dt.) ↑unaufgelöste Referenz.

unteilbare Anweisung Formulierung einer ↑Aktion, für die ↑Unteilbarkeit gilt. Eine solche Aktion findet scheinbar “zeitlos“ statt. Während die Aktion stattfindet, ändert sich der Zustand von dem ↑Programm, in dem die betreffende Anweisung kodiert ist, anscheinend nicht.

unteilbarer Grundblock Abschnitt in einem ↑Programm, auch Basisblock genannt, der einen linearen und der ↑Unteilbarkeit unterzogenen Kontrollfluss beschreibt. Linearität des Kontrollflusses bedeutet, dass der Block nur einen Einsprungspunkt und einen Ausstieg aufweist.

unteilbares Betriebsmittel Exklusivität beanspruchendes ↑wiederverwendbares Betriebsmittel, dessen Einheiten zu einem Zeitpunkt nur von maximal einen ↑Prozess erworben (*acquire*) und für die Zeitdauer bis zur Freisetzung (*release*) belegt werden dürfen. Typisches Beispiel eines solchen Betriebsmittels ist jede Form von Drucker: für die Dauer eines Druckvorgangs steht das Druckgerät keinem anderen Druckprozess zur Verfügung, ansonsten können unlesbare oder unverständliche Ausdrucke die Folge sein. In diese Kategorie ist aber auch ein einzelnes ↑Speicherwort einzuordnen, wenn nämlich durch ↑Parallelverarbeitung nicht tolerierbare Lese-Schreib-Konflikte für eine an der ↑Adresse dieses Worts liegende und *gemeinsam benutzte Variable* auftreten können. Im übertragenen Sinn passt auch ein ↑kritischer Abschnitt dazu, bei dem letztlich ja die von seinem Programmtext belegten Speicherworte — nicht etwa die von dem Text referenzierten Daten! — zeitweilig der exklusiven Nutzung nur durch einen Prozess unterzogen sind (↑unteilbarer Grundblock).

Unteilbarkeit Umstand, der die Aufsplittung einer ↑Aktion oder ↑Aktionsfolge und Verteilung der Einzelmaßnahmen auf verschiedene Zeitintervalle verhindert. Die (Schrittfolge einer) Aktion beziehungsweise Aktionsfolge findet scheinbar gleichzeitig statt, sie ist atomar, kann sich nicht mit einem weiteren Exemplar ihrer selbst zeitlich überlappen.

Unterbrechung Störung von einem ↑Prozess, einer ↑Aktionsfolge oder ↑Aktion, verursacht durch ein in Bezug auf diese Handlungen externes Ereignis. Das Ereignis findet asynchron zu der Handlung statt, unvorhersagbar und nicht reproduzierbar. Bestenfalls lässt sich noch die ↑Zwischenankunftszeit der Ereignisse abschätzen, womit jedoch nur eine Aussage über die Frequenz und nicht über den Zeitpunkt möglich ist.

Unterbrechungsanforderung Forderung der ↑Peripherie an die ↑CPU, eine ↑Unterbrechungsbehandlung einzuleiten und durchzuführen.

Unterbrechungsbehandlung Vorgang zur Analyse und Bearbeitung der bei Ausführung von einem ↑Maschinenprogramm aufgetretenen Störung (↑Unterbrechung). Handelt es sich um eine berechtigte (echte) Störung, wird diese bearbeitet und die Ausführung des unterbrochenen Programms anschließend wieder aufgenommen. Im Falle einer unberechtigten Störung (↑*spurious interrupt*), wird diese vermerkt. Zuviele unberechtigte Störungen innerhalb kurzer

Zeit deuten auf einen Fehler in der Hardware hin und könnten ↑Panik auslösen. Anderenfalls wird die Programmausführung fortgesetzt.

Unterbrechungsdeskriptortabelle Jargon (Intel) für eine ↑Unterbrechungsvektortabelle, deren Einträge ↑Deskriptoren für die ↑Ausnahmehandhaber sind.

unterbrechungsfreier Betrieb ↑Arbeitsmodus von einem ↑Prozessor (↑CPU, ↑Rechenkern), um ↑Synchronisation herzustellen. In dem Modus lässt der Prozessor eine ↑Unterbrechungsanforderung nicht durch. Der Prozessor handelt für das ↑Betriebssystem (↑*privileged mode*), unterbindet dabei aber gleichzeitig eine eventuelle ↑Unterbrechungsbehandlung. Dadurch wird eine mögliche nebenläufige ↑Aktionsfolge für den in dem Modus handelnden Prozessor ausgeschlossen. Der Prozessor wechselt in diesen Modus entweder implizit, nämlich wenn er in seinem ↑Unterbrechungszyklus eine anhängende Unterbrechungsanforderung erkennt, oder explizit, durch Setzen einer ↑Unterbrechungssperre. Während der Prozessor in dem Modus agiert, ist sein Unterbrechungszyklus faktisch außer Kraft. Der Prozessor verlässt den Modus ebenfalls implizit oder explizit, nämlich beim Rücksprung aus der Unterbrechungsbehandlung (wenn der zu einem früheren Zeitpunkt im Unterbrechungszyklus gesicherte und jetzt wiederhergestellte ↑Prozessorstatus dies bestimmt) oder Aufheben der Unterbrechungssperre. Je nach Art der Zustellung des der Unterbrechungsanforderung zugrunde liegenden Digitalsignals, dessen Frequenz und der Dauer des Modus ist es nach seiner Beendigung möglich, eine solche zwischenzeitlich gestellte Anforderung entweder verpasst zu haben (↑Flankensteuerung) oder nachträglich anzunehmen und zu behandeln (↑Pegelsteuerung).

Unterbrechungshandhaber (en.) ↑*interrupt handler*. Bezeichnung für einen speziellen, ausschließlich zur ↑Unterbrechungsbehandlung ausgelegten und vorgesehenen ↑Handhaber.

Unterbrechungshandhaberstumpf (en.) ↑*interrupt handler stub*. Bezeichnung für einen ↑Unterbrechungshandhaber begrenzter, fester Größe. Ein solcher Handhaber ist typisch für eine ↑CPU der ↑RISC Klasse. Je nach Art der ↑Unterbrechung ist in dem Stumpf die ↑Unterbrechungsbehandlung entweder komplett (↑SLIH) oder anteilig (↑FLIH) möglich. Gegebenenfalls verzweigt dieser Handhaber auch nur über eine in Software implementierte Vektortabelle (↑IVT oder ↑IDT).

Unterbrechungslatenz ↑Latenzzeit ab Zeitpunkt der Wahrnehmung der ↑Unterbrechung in der ↑CPU bis zum Zeitpunkt der Aufnahme der ↑Unterbrechungsbehandlung im Betriebssystem. Wurde keine ↑Unterbrechungssperre angefordert, ist die Verzögerungszeit bestimmt durch die restliche Ausführungszeit von dem ↑Maschinenbefehl, der im Moment der ↑Unterbrechung aktiv war. Wurde jedoch eine Unterbrechungssperre gesetzt, ergibt sich die Verzögerung aus der Restlaufzeit bis zur Aufhebung der Sperre durch das Betriebssystem.

Unterbrechungsmaske (en.) ↑*interrupt mask*. Bezeichnung für eine Vorrichtung in der ↑CPU, um die ↑Unterbrechung eines ↑Programmablaufs zeitweilig nicht zuzulassen. In logischer Hinsicht gibt die Maske eine bestimmte ↑Unterbrechungsprioritätsebene vor, auf der die CPU operiert. Unterbrechungen mit einer Priorität kleiner oder gleich dieser Ebene sind unterbunden. Das Setzen der Maske ist für gewöhnlich eine privilegierte Operation (↑*privileged mode*).
In technischer Hinsicht ist eine solche Maske eine Bitleiste von unterschiedlicher Länge, je nach CPU. Für ↑x86 sowie allen gegenwärtigen 64-Bit Varianten dieser Familie (von Intel oder AMD) besteht die Bitleiste aus genau einem Bit (*interrupt flag*) im ↑Statusregister, womit nur eine Unterbrechungsprioritätsebene definiert ist. Bei Prozessoren der ↑PDP 11 oder ↑m68k Familie umfasst(e) die Bitleiste drei Bits und damit acht Prioritätsebenen.

Unterbrechungsprioritätsebene Attribut des aktuellen Systemzustands, das die Priorität der momentan von der ↑CPU akzeptierten ↑Unterbrechungsanforderung anzeigt. Typischerweise werden Anforderungen mit einer Priorität kleiner oder gleich der Priorität der CPU nicht akzeptiert, diese sind gesperrt. Im Normalbetrieb läuft die CPU auf Prioritätsebene 0 und jede

Unterbrechungsanforderung hat eine Priorität größer als 0. Mit jeder akzeptierten Unterbrechungsanforderung wird die Priorität der CPU auf die Ebene dieser Anforderung gehoben. Bei Rückkehr aus der ↑Unterbrechungsbehandlung kehrt die Priorität der CPU wieder auf die vorige Ebene zurück. Im ↑Unterbrechungszyklus führt die CPU damit implizit eine ↑Unterbrechungssperre für die Prioritätsebene durch, die mit der Unterbrechungsanforderung assoziiert ist. Eine solche Sperre kann jedoch auch explizit gesetzt werden, um der ↑Unterbrechung eines Programmablaufs vorzubeugen (↑kritischer Abschnitt). Darüber hinaus kann diese Sperre im Rahmen der Unterbrechungsbehandlung explizit aufgehoben werden, um noch vor Rückkehr daraus auf Unterbrechungsanforderungen reagieren zu können und damit die ↑Unterbrechungslatenz zu verkürzen. Läuft die Unterbrechung jedoch über ↑Pegelsteuerung und hat der ↑Unterbrechungshandhaber vor Aufhebung der Sperre die Unterbrechung noch nicht am ↑Peripheriegerät bestätigt, gilt die zugehörige Anforderung immer noch als anhängig und es kommt zum indirekt rekursiven Aufruf des Handhabers durch die CPU. Dies kann ↑Panik auslösen, da davon auszugehen ist, dass ein solcher Kreislauf im Unterbrechungshandhaber nicht durchbrochen wird beziehungsweise werden kann.

Unterbrechungssperre Vorkehrung zur Abwehr einer ↑Unterbrechung, stellt ↑Synchronisation her. Implementiert als ↑privilegierter Befehl, der nur lokale Auswirkung auf seinen ↑Prozessor hat und eine an ihn gestellte ↑Unterbrechungsanforderung der ↑Peripherie nicht durchlässt. Dazu muss diese Anforderung am Prozessor maskierbar sein (↑IRQ). Eine nichtmaskierbare Anforderung (↑NMI) dagegen kann nicht gesperrt werden: Gegebenenfalls lässt sich die Peripherie jedoch zeitweilig umprogrammieren, einen NMI nicht anzufordern. Zur Aufhebung der Sperre kommt eine entsprechende inverse Operation zum Einsatz. Bei der Unterbrechungsabwehr besonders zu beachten ist die Art der Zustellung des Digitalsignals (↑Pegelsteuerung, ↑Flankensteuerung).

Unterbrechungsvektor (en.) ↑*interrupt vector*. Bezeichnung für eine definierte ↑Adresse im ↑Hauptspeicher, an der entweder ↑Daten oder ↑Maschinenbefehle zur Auslösung einer ↑Unterbrechungsbehandlung liegen. Gemeinhin sind solche Vektoren in einer Tabelle (↑IVT oder ↑IDT) zusammengefasst.
Enthält der Vektor Daten (typisch für ↑CISC), handelt es sich dabei in der einfachsten Ausfertigung lediglich um die Adresse (↑PC) des von der ↑CPU automatisch im ↑Unterbrechungszyklus zu startenden ↑Unterbrechungshandhabers. Ein treffendes Beispiel dafür ist ↑m68k. Bei einer CPU mit segmentierter Adressierung ist für gewöhnlich zusätzlich noch der ↑Segmentname Teil des Vektors, wie etwa im Falle des ↑x86 bis zum 80286. Ab 80386 ist ein solcher Vektor ein ↑Deskriptor (8 Bytes), durch den insbesondere die Art des ↑Ausnahmehandhabers spezifiziert ist und welchen Tortyp die CPU für seine Aktivierung verwenden soll (↑*interrupt*, ↑*trap* oder ↑*task gate*). Bei Prozessoren der ↑PDP 11 Familie besteht der Vektor aus einem Tupelwert, der bei Annahme der ↑Unterbrechung von der CPU als PC und ↑PSW übernommen wurde.
Enthält der Vektor Maschinenbefehle (typisch für ↑RISC), beginnt die CPU die Ausführung mit der Unterbrechungsbehandlung direkt an dieser Stelle. Ein solcher Vektor enthält lediglich einen ↑Unterbrechungshandhaberstumpf von fester, begrenzter Größe (z.B. 256 Bytes beim ↑PowerPC). Durch diesem Stumpf ist nicht nur der weitere Verlauf der Unterbrechungsbehandlung bestimmt, sondern auch die Art und Weise, wie die jeweilige Behandlung problemspezifisch im ↑Betriebssystem eingebettet und zu aktivieren ist (z.B. als ↑Unterprogramm, ↑Koroutine oder gar in Form einer ↑Prozessinkarnation).

Unterbrechungsvektortabelle (en.) ↑*interrupt vector table*. Vorrichtung in einer ↑CPU zur listenförmigen Zusammenstellung der verfügbaren ↑Unterbrechungsvektoren. Wenn die CPU ihren ↑Unterbrechungszyklus durchführt, liest sie eine (vom ↑Peripheriegerät oder ↑PIC gesendete) Unterbrechungsnummer vom Bus und führt mit dem gelesenen Wert sowie der Tabellenadresse als Operanden eine *indizierte Adressierung* durch, um den der ↑Unterbrechung zugeordneten Vektor zu laden und damit die ↑Unterbrechungsbehandlung zu starten. Die Tabellengröße ist prozessorspezifisch, für ↑x86 sind 256 Einträge festgelegt.

Unterbrechungszyklus Phase im ↑Abruf- und Ausführungszyklus, wird typischerweise am Ende der Befehlsausführung durchlaufen und umfasst folgenden *Vorspann* an Teilschritten, um die ↑Unterbrechungsbehandlung — allgemein: ↑Ausnahmebehandlung, denn für die ↑Abfangung gilt dasselbe — einzuleiten: (1) den ↑Prozessorstatus sichern, mindestens davon die Inhalte von ↑Statusregister und ↑Befehlszähler, (2) den Befehlszähler mit der ↑Adresse vom ↑Unterbrechungshandhaber laden und (3) in den privilegierten ↑Betriebsmodus wechseln, sofern von der ↑CPU implementiert. Der *Nachspann* der Unterbrechungsbehandlung kommt durch einen speziellen ↑Maschinenbefehl in ↑Aktion, der ganz normal im Abruf- und Ausführungszyklus verarbeitet wird und folgenden wesentlichen Schritt macht: (5) den Prozessorstatus wieder herstellen. Hier ist zu beachten, dass die Schritte (2) und (5) jeweils den Befehlszähler verändern und damit für den nächsten Durchlauf vom Abruf- und Ausführungszyklus vorgeben, von welcher Adresse der nächste Maschinenbefehl zur Ausführung abgerufen werden soll. Diese Schritte bewirken letztlich den Hinsprung (2) zur Unterbrechungsbehandlung und den Rücksprung (5) zu dem ↑Prozess, der unterbrochen/abgefangen wurde. Da der Betriebsmodus der CPU in ihrem Statusregister vermerkt ist, wird mit Wiederherstellung vom Prozessorstatus implizit zu dem Betriebsmodus umgeschaltet (5), der zum Zeitpunkt der ↑Unterbrechung aktiv war. Zur Sicherung (1) und Wiederherstellung (5) all dieser Statusdaten findet typischerweise ein ↑Stapelspeicher Verwendung.

Unterprogramm ↑Programm, dessen Aufruf in demselben (Rekursion), einem anderen oder mehrerer anderer Programme kodiert ist. Je nach Programmiersprache und -paradigma technisch verschieden umgesetzt und unterschiedlich bezeichnet: Operation, Abschnitt, Routine, Subroutine, Prozedur, Funktion, Methode oder Makro.

Ureingabe Vorgang zur Inbetriebnahme von einem ↑Rechner, indem der ↑Urlader über eine als Schalttafel ausgelegte ↑Systemkonsole manuell in den ↑Hauptspeicher gebracht wird. Die wesentlichen Schritte dabei sind (am Beispiel ↑PDP 11/40):

1. die Hardware des Rechners betriebsbereit machen:
   (a) den ENABLE/HALT Schalter in die HALT Position bringen, damit die Funktionen der Systemkonsolle freigeben
   (b) sicherstellen, dass der OFF/POWER/LOCK Schalter in der POWER Position steht
2. das ↑Urladeprogramm in den Hauptspeicher bringen:
   (a) die ↑Adresse der ersten zu beschreibenden Stelle (↑Speicherwort) im Hauptspeicher als binär kodierten (16-stelligen) Wert am Schaltregister einstellen
   (b) den eingestellten Wert durch Betätigung der LOAD ADRS Taste in das Busadressregister (BAR) laden
   (c) das als nächstes in den Hauptspeicher zu transferierende Datum als binär kodierten (16-stelligen) Wert am Schaltregister einstellen
   (d) den eingestellten Wert durch Betätigung der DEP Taste in das durch BAR adressierte Speicherwort laden, BAR wird dabei automatisch um 2 erhöht
   (e) die Schritte ab 2c wiederholen, solange das Urladeprrogramm noch nicht vollständig eingegeben worden ist
3. den Rechner hochfahren:
   (a) das Bandlaufwerk anschließen und das Band mit dem gewünschten ↑Absolutlader in das Laufwerk einlegen
   (b) die Schritte 2a und 2b zur Eingabe der Adresse, an der die Ausführung des Urladeprogramms starten soll, erneut durchführen
   (c) den ENABLE/HALT Schalter in die ENABLE Position bringen und mit der START Taste das Urladeprogramm schließlich zur Ausführung bringen
   (d) abwarten, dass das Bandlaufwerk zum Stillstand kommt und der Absolutlader damit seine Aufgabe (das ↑Betriebssystem zu laden) erfüllt hat

(e) den OFF/POWER/LOCK Schalter in die LOCK Position bringen

Typischerweise ist das Urladeprogramm nicht sehr umfangreich: im betrachteten Beispiel umfasst es lediglich etwa 14 Maschinenbefehle beziehungsweise Speicherworte, die nach und nach in der Schleife um Schritt 2c einzugeben sind. Insgesamt ist der Ablauf jedoch sehr spezifisch auf den jeweiligen Rechner ausgelegt und variiert damit durchaus auch stark von Hersteller zu Hersteller. In (vom Rechnerhersteller mitgelieferten) Festwertspeicher kodiert, kann der Vorgang allerdings weitestgehend automatisiert ablaufen.

Urladen Vorgang bei der Inbetriebnahme von einem ↑Rechner, nämlich das ↑Betriebssystem in den ↑Hauptspeicher zu bringen (↑*bootstrap*).

Urladeprogramm Bezeichnung für ein ↑Programm) zum ↑Urladen von einem ↑Betriebssystem, bestimmt die Funktions- und Arbeitsweise von dem ↑Urlader.

Urlader Bezeichnung für einen ↑Lader, der zur Inbetriebnahme von einem ↑Rechner das ↑Betriebssystem in den ↑Hauptspeicher bringt und startet. Der Ladevorgang ist üblicherweise zweistufig organisiert. Zunächst wird ein auf das Betriebssystem zugeschnittener Lader vom gewählten ↑Datenträger heruntergeladen, typischerweise aus dem ersten ↑Block (↑*boot block*), und gestartet. Dieser Lader lädt das Betriebssystem und bietet dazu häufig eine Wahl der ↑Betriebsart an, in der das geladene Betriebssystem den Rechner betreiben soll: im Falle von ↑UNIX etwa Ein- oder Mehrbenutzerbetrieb (*single/multi-user mode*).

USB Abkürzung für (en.) *universal serial bus*, (dt.) vielseitiger serieller Bus.

use case (dt.) ↑Anwendungsfall.

used bit Synonym zu ↑*reference bit*.

user level (dt.) ↑Benutzerebene.

user mode (dt.) Benutzermodus: ↑Arbeitsmodus.

user program (dt.) ↑Anwenderprogramm, ↑Anwendungsprogramm, ↑Benutzerprogramm.

user space (dt.) ↑Benutzerbereich, -gebiet, -gelände.

user thread (dt.) ↑Anwendungsfaden.

user time (dt.) ↑Benutzerzeit.

user-level scheduling (dt.) Planung auf Benutzerebene. Bezeichnung für eine ↑mitlaufende Planung, die eng gekoppelt mit dem ↑Betriebssystem als Funktion direkt im Kontext eines ↑Benutzerprogramms stattfindet. Der ↑Planer als dedizierter Betriebssystemaufsatz ist ein ↑Unterprogramm in dem Benutzerprogramm. Dieser Ansatz gibt jedem Benutzerprogramm die Option zu einem eigenen Planer, der speziell auf die Bedürfnisse der ↑Anwendung zugeschnitten ist und dazu ein anwendungsorientiertes ↑Einplanungskriterium implementiert.
Mit verschiedenen Benutzerprogrammen können damit zugleich verschiedene Planer oberhalb des Betriebssystems tätig sein. Die Herausforderung dabei ist, ↑Simultanverarbeitung verschiedener Benutzerprogramme durch das Betriebssystem möglichst frei von ↑Interferenz mit den verschiedenen Planern zu ermöglichen. Zu beachten ist hier, dass die Planer in den Benutzerprogrammen nicht voneinander wissen und sie nur aktiv werden können, wenn das Betriebssystem die Ausführung ihres Benutzerprogramms veranlasst. Für gewöhnlich betreibt das Betriebssystem die ↑CPU dazu in einem ↑Zeitteilverfahren, das die Planer auf der Benutzerebene im ↑Rundlauf aktiviert.

userland siehe ↑*user space*.

utility (dt.) ↑Dienstprogramm.

V Abkürzung für (hol.) *verhoog*, (dt.) erhöhen; synonym zu (en.) *up*, *signal*, *release*.

VAX Abkürzung für „*virtual address extension*“ und Bezeichnung für eine von DEC entwickelte ↑ISA als Nachfolge zur ↑PDP 11: 32-Bit, ↑CISC, vier Privilegstufen (*kernel*, *executive*, *supervisor*, *user*), 32 ↑Unterbrechungsprioritätsebenen, hardwareunterstützten ↑AST, erstmalig umgesetzt mit der VAX-11/780 (1977) und betrieben durch ↑VMS, später ↑OpenVMS.

Veränderungsbit ↑Speichermarke im ↑Seitendeskriptor, die von der ↑MMU beim Schreibzugriff auf die betreffende ↑Seite gesetzt wird.

Verbindungsregister (en.) ↑*link register*. Bezeichnung für ein ↑Prozessorregister, das die Rücksprung- oder Fortsetzungsadresse des gegenwärtig auf dem ↑Prozessor ausgeführten ↑Unterprogramms enthält. Beim Unterprogrammaufruf wird diese Adresse nicht automatisch auf dem ↑Stapelspeicher gesichert, wodurch für den Hin- und Rücksprung kostspielige Schreib-/Leseoperationen für den ↑Hauptspeicher entfallen. Nur im Falle von geschachtelten Aufrufen kommt es zu solchen Haupstpeicherzugriffen, die dann durch explizit abzusetzende, aber gewöhnliche ↑Maschinenbefehle zum Schreiben/Lesen eines ↑Maschinenworts bewerkstelligt werden. Dieses Register ist typisches Merkmal des ↑Programmiermodells von einem ↑RISC.

Verbund ↑Datentyp, der aus einem oder mehreren (identischen oder verschiedenen) Datentypen zusammengesetzt ist. Die einzelnen Elemente (Attribute) dieses Datentyps sind entweder nacheinander im ↑Speicher angeordnet und durch ihre jeweilige ↑Adresse eindeutig unterscheidbar oder überlagern sich im Speicher und besitzen alle dieselbe Adresse: `struct` beziehungsweise `union` in ↑C/↑C++. Jedes dieser Elemente kann selbst wieder einen solchen Zusammenschluss von Datenstrukturen bilden.

verdeckter Kanal Variante von einem Sicherheitsangriff auf ein ↑Rechensystem, die es ermöglicht, Informationen von einem ↑Prozess abzugreifen und zu einem anderen Prozess zu transferieren, ohne dass beide zur direkten Kommunikation berechtigt wären. Der Kanal ist nicht zum Informationstransfer bestimmt: er benutzt keine der von einem ↑Betriebssystem angebotenen legitimen Mechanismen zum Datentransfer, sondern zweckentfremdet andere legitime und per regulärem ↑Systemaufruf angebotene Mechanismen dafür. Beispiel ist der durch bestimmte Berechnungen eines Prozesses bewirkte Effekt auf die Systemlast, die von einem anderen Prozess gemessen werden kann. Die Lastschwankungen geben Anlass zu Spekulationen über die in dem Moment stattfindenden Vorgänge beziehungsweise werden gezielt zur Informationskodierung genutzt.

Verdrängung (en.) ↑*preemption*. ↑Aktionsfolge, durch die ein ↑wiederverwendbares Betriebsmittel den Besitzer wechselt. Dem ↑Prozess, der dieses ↑Betriebsmittel gerade besitzt, wird es entzogen und einem anderen Prozess zugeteilt, der dann zum neuen Besitzer wird. Typisches Beispiel eines solchen Betriebsmittels ist die ↑CPU oder ein einzelner ↑Rechenkern, wenn nämlich die ↑Prozesseinplanung nach einem ↑Zeitteilverfahren arbeitet.

Verdrängungsgrad Abstufung des Vorhandenseins der Eigenschaft von einem ↑Betriebssystem, ↑Verdrängung frei von ↑Latenz zu gestalten. Benutzt das Betriebssystem etwa eine ↑Unterbrechungssperre zur ↑Synchronisation, verzögert sich die mögliche Verdrängung von dem gerade auf (einem bestimmten ↑Rechenkern) der ↑CPU stattfindenden Prozess wenigstens für die Dauer dieser Sperre. Gleiches gilt im Falle einer ↑Verdrängungssperre, die das Betriebssystem entweder zusätzlich oder alternativ für die Synchronisation gebraucht. Benutzt das Betriebssystem eine ↑Umlaufsperre, um Synchronisation für den ↑Planer in einem ↑Mehrkernprozessor oder ↑Multiprozessor zu erzielen, verzögert sich auch hier die mögliche Verdrängung eines in dem Fall aber fernen Prozesses: beispielsweise um den ↑Rechenkern dieses Prozesses für einen von einem anderen Rechenkern aus deblockierten vorrangigen Prozess zu nutzen. Jede dieser Sperren erzwingt ein Stück ↑nichtsequentielles Programm bestimmter Länge. Diese Länge variiert im Allgemeinen, wie auch die Häufigkeit solcher Stücke verteilt über das gesamte ↑Programm — bestimmt durch das ↑Operationsprinzip des Betriebssystems. Ein

Betriebssystem operiert für gewöhnlich *verdrängend* zwischen solchen Sperren, vorausgesetzt sein Operationsprinzip impliziert keine allumfassende Sperre (↑BKL) und definiert faktisch kein ↑sequentielles Programm. Ist das Betriebssystem dagegen frei von solchen Sperren, operiert es *voll verdrängend* und bietet damit die besten Voraussetzungen für das größtmögliche Maß an ↑Parallelität. Die genaue Abstufung der Intervalle, in der Verdrängung stattfinden kann, lässt sich demnach durch folgendes Verhältnis quantifizieren (*degree of preemption*):

$$dop = \frac{length(nonsequential\ code)}{length(total\ code)}$$

Bezugspunkt für die Bestimmung der jeweiligen Länge ist der einzelne ↑Maschinenbefehl. Nur ein vollverdrängend operierendes Betriebssystem erlaubt ↑Prozesswechsel nach jeden einzelnen Maschinenbefehl, nur in dem Fall ergibt der Quotient eine Eins beziehungsweise liegt ein Grad von 100 Prozent vor. In allen anderen Fällen liegt der Quotientenwert im Bereich $[0, 1[$ und der Grad unter 100 %. Damit würde Parallelität wie auch die Wahrscheinlichkeit von besserer Auslastung und Leistung von dem ↑Rechensystem entsprechend verringert.

Verdrängungspunkt (en.) ↑*preemption point*. Stelle in einem ↑Programm, an der die ↑Verdrängung eines ↑Prozesses, der nämlich durch eben dieses Programm definiert ist, stattfinden kann. Diese Stelle zeichnet sich aus durch eine Anweisung, die die Bereitwilligkeit eines Prozesses zur Abgabe des von ihm vorübergehend kontrollierten ↑Prozessors dem ↑Planer anzeigt. Der kooperativ von dem Prozess ins Spiel gebrachte Planer kann daraufhin eine ↑Umplanung vornehmen, deren Folge die ↑Einlastung des Prozessors durch einen anderen Prozess sein kann.

Verdrängungssperre Vorkehrung zur Abwehr einer ↑Verdrängung, stellt ↑Synchronisation her. Implementiert als ↑Aktionsfolge im ↑Betriebssystem, die einen ↑Prozesswechsel zeitweilig unterbindet. Entweder wird die ↑Ablaufplanung oder die ↑Einlastung von einem ↑Prozess, dessen Auslösung durch ein bestimmtes ↑Ereignis verursacht ist, zurückgestellt. Der Prozesswechsel wird erst verzögert wirksam, nämlich im Moment der Aufhebung der entsprechenden Sperre. Die Technik ist einer ↑Unterbrechungssperre sehr ähnlich, jedoch wird nicht der ↑Arbeitsmodus der ↑CPU geändert, sondern der Modus, in dem das ↑Betriebssystem operiert.

Verhungern (en.) ↑*starvation*. Bezeichnung eines Problems bei der ↑Ablaufplanung oder ↑Synchronisation, wodurch ein ↑Prozess die für sein weiteres Vorankommen erforderliche ↑Ressource fortwährend versagt bleibt. Der Prozess unterliegt keiner ↑Verklemmung, da er nicht blockiert, sondern laufend oder bereit ist (↑Prozesszustand). Er kommt nicht voran, obwohl die von ihm benötigte Ressource zeitweilig frei ist, sie ihm jedoch von einem anderen Prozess immer wieder weggeschnappt wird, bevor er sie für sich beanspruchen kann.
In Bezug auf bestimmte Implementierungsvarianten einer ↑Umlaufsperre wird dieses Problem auch als ↑*after you, after you*-Blockierung bezeichnet.

Verklemmung (en.) *deadlock*. Bezeichnung für die gegenseitige Blockierung gleichzeitig, aber unabhängig voneinander stattfindender ↑Prozesse (in Anlehnung an den Duden) — mehr dazu aber erst in VL 11.

Verklemmungsvermeidung (en.) *deadlock avoidance*. Bezeichnung für *analytische Maßnahmen*, um es nicht zu einer ↑Verklemmung kommen zu lassen — mehr dazu aber erst in VL 11.

Verknüpfung Paarung von ↑Dateiname und ↑Indexknoten durch eine ↑numerische Adresse (↑*inode number*), gespeichert im ↑Verzeichniseintrag für die benannte ↑Datei. Damit kann weder eine Beziehung zu einem ↑Verzeichnis auf demselben noch zu einer ↑Datei auf einem anderen (an einen ↑Befestigungspunkt eingehängtes) ↑Dateisystem hergestellt werden. Die zu assoziierende Datei ist eine ↑Entität desselben Dateisystems.

verschiebender Lader ↑Lader, der beim ↑Laden von einem ↑Maschinenprogramm zugleich für die ↑Relokation von eben diesem ↑Programm sorgt.

Verschmelzung Vereinigung von einem frei werdenden ↑Adressbereich mit einem oder zwei angrenzenden Adressbereichen zu einem einzigen großen Adressbereich, wobei die Länge eines jeden dieser Bereiche bekannt ist. Optionales Merkmal von einem ↑Platzierungsalgorithmus, um den Grad von ↑Fragmentierung vermindern zu können. Ein frei werdender ↑Speicherbereich ist seiner Länge (↑*best fit*/↑*worst fit*) oder ↑Adresse (↑*first fit*/↑*next fit*) nach auf die Liste freier Speicherabschnitte (↑*hole list*) zu platzieren. Dabei wird überprüft, ab dieser Abschnitt direkt zu Abschnitten, die bereits auf der Liste stehen, benachbart ist. Liegt der durch einen ↑Prozess frei werdende Abschnitt ($hole_p$) im ↑Hauptspeicher direkt vor einem bereits auf der Liste stehenden Abschnitt ($hole_l$), gilt also $address(hole_p)+sizeof(hole_p) = address(hole_l)$, oder direkt danach, gilt also $address(hole_l) + sizeof(hole_l) = address(hole_p)$, wird der bereits auf der Liste stehende Abschnitt entsprechend um die Länge des frei werdenden Abschnitts korrigiert: ein größerer freier Abschnitt wird erzeugt und es kommt kein weiterer Listeneintrag hinzu. Liegt der frei werdende Abschnitt genau zwischen zwei bereits auf der Liste stehenden Abschnitte, kommt es zum Lückenschluss: ein noch größerer freier Abschnitt wird erzeugt und es wird ein Listeneintrag gestrichen. Diese Maßnahme ist einfach durchzuführen, wenn die Listeneinträge der Adresse nach sortiert sind. In dem Fall ändert sich nicht die Listenposition des Eintrags, sondern lediglich der im Eintrag verzeichnete Längenwert. Sind die Listeneinträge jedoch der Größe nach sortiert, erzeugt ein Verschmelzungsschritt einen größeren freien Abschnitt, der in einem zweiten Suchlauf wieder einzusortieren ist, um dabei durch einen weiteren möglichen Verschmelzungsschritt zu einem noch größeren Abschnitt zu werden, der schließlich einen dritten Suchlauf zum Einsortieren nach sich zieht.

Verschnitt Abfall, bei der ↑Speicherzuteilung anfallender Rest. Typischerweise gibt ein ↑Prozess die Anzahl von ↑Byte an, die ein ihm zuzuteilender ↑Speicherbereich umfassen soll. Wenn die Speicherzuteilung jedoch nicht byteorientiert arbeitet, weil die Größe ihrer kleinsten Verwaltungseinheit (↑Speicherwort, ↑Seite) ein Vielfaches der Größe eines Byte ausmacht, erhält der Prozess einen größeren Bereich als angefordert zugeteilt. In aller Regel bleibt der überschüssige Anteil in diesem Speicherbereich von dem Prozess ungenutzt (↑interner Verschnitt). Arbeitet die Speicherzuteilung jedoch byteorientiert und teilt sie einem Prozess einen Speicherbereich zu, indem sie ein verfügbares ↑Loch ausreichender Größe verkleinert aber nicht eliminiert, bleibt ein kleineres Loch übrig, das gegebenenfalls für weitere Zuteilungen unbrauchbar ist (↑externer Verschnitt).

verteilter gemeinsamer Speicher ↑Arbeitsspeicher, der innerhalb derselben Ebene der ↑Speicherhierarchie über mehrere ↑Rechner verteilt vorliegt. Typischerweise ist dieser Arbeitsspeicher als auseinanderliegender globaler ↑Hauptspeicher organisiert, dessen Einzelteile die lokalen Hauptspeicher verschiedener Rechner bilden. Zugriffe auf einen entfernten ↑Speicherbereich sind nur über ein ↑Rechnernetz möglich und für gewöhnlich mit abgeschwächter ↑Speicherkonsistenz behaftet.

Mittels ↑Speichervirtualisierung können die verteilten Hauptspeicher in logischer Hinsicht als Einheit durch ein ↑Betriebssystem zur Verfügung gestellt werden. Grundlage dafür ist ein hinreichend großer ↑seitennummerierter Adressraum, über den die im Rechnernetz verfügbaren Hauptspeicher zugänglich gemacht werden können (↑PGAS). Die ↑Umlagerung von Speicherinhalten, ausgelöst durch einen ↑Seitenfehler beim Zugriff auf einen entfernten Hauptspeicher, geschieht sodann nicht zwischen verschiedenen Ebenen (d.h., ↑Vordergrundspeicher und ↑Hintergrundspeicher) der Speicherhierarchie, sondern innerhalb derselben Hauptspeicherebene.

Verwaltungsdaten Bezeichnung für ↑Daten, die zur Bewirtschaftung von ↑Nutzdaten erforderlich sind (↑*overhead*).

Verzeichnis Ordner; nach einem bestimmten System geordnete Aufstellung mehrerer unter einem bestimmten Gesichtspunkt zusammengehörender ↑Daten (in Anlehnung an den Duden). Die

Daten sind in einer ↑Datei erfasst, deren ↑Name in dem Ordner verzeichnet ist. In Bezug auf diesen Namen definiert der Ordner gleichsam einen ↑Namenskontext, der die ↑Bindung zu einer Dateidatenstruktur beschreibt (↑*directory entry*). Ein solcher Ordner ist letztlich selbst eine Datei, die im ↑Dateisystem zur Speicherung eben dieser Bindungen vorgesehen ist. Die genaue Auslegung dieser speziellen Datei gibt das ↑Betriebssystem vor. Im Falle von ↑UNIX sind in jeder solchen Datei zwei vordefinierte Verzeichnitseinträge vorgesehen: ein Eintrag (↑*dot*) definiert die Bindung zur Ordnerdatei selbst und ein zweiter Eintrag (↑*dot dot*) verweist auf den Elterordner. Ansonsten werden keine weiteren Vorgaben gemacht.

Verzeichniseintrag Eintrag in einem ↑Verzeichnis, der die ↑Bindung zu einer namentlich identifizierten Dateidatenstruktur beschreibt (↑*hard link*). Dieser Eintrag bildet die ↑symbolische Adresse einer ↑Datei ab auf die ↑numerische Adresse der dieser Datei im ↑Dateisystem zugeordneten Datenstruktur: wesentliches Merkmal dabei ist das Tupel $(Symbol, Adresse)$, wobei ↑Adresse (im Falle von ↑UNIX) eine ↑Indexknotennummer ist. Die Datenstruktur entspricht dem auf dem ↑Datenträger daselbst gespeicherten ↑Deskriptor (↑inode) für diese Datei.

Virtualisierung Nachbildung einer (realen) Maschine oder Teile davon. Die komplette Nachbildung wird auch als ↑Vollvirtualisierung bezeichnet, ansonsten ist der Begriff ↑Teilvirtualisierung angebracht. Grundsätzlich kann die Nachbildung allein durch einen in Software implementierten ↑Interpreter geleistet werden (↑CSIM). Dies wird für gewöhnlich jedoch nur praktiziert, wenn die nachgebildete Maschine (↑Prozessor) unterschiedlich ist von jenem Prozessor, auf dem dieser Interpreter zur Ausführung kommt. Die ↑JVM ist ein weit verbreitetes Beispiel dafür. In den anderen Fällen findet ein ↑Hypervisor Verwendung, der lediglich die ↑partielle Interpretation bestimmter Prozessorbefehle (↑sensitiver Befehl) vornimmt.

Virtualisierungssystem ↑Interpretersystem zur Nachbildung einer (realen) Maschine, typischerweise realisiert durch einen ↑VMM. Der VMM ist ein für eine ↑Wirtsmaschine bestimmtes Steuerprogramm, er erschafft die Ablaufumgebung für eine ↑virtuelle Maschine. Unterschieden wird zwischen *Typ I* und *Typ II*. Ein Typ I VMM läuft auf der bloßen Hardware (Wirtsmaschine), direkt auf der ↑CPU, wohingegen ein Typ II VMM zusätzlich noch ein ↑Wirtsbetriebssystem benutzt. Zentrale Funktion von einem VMM ist es, die direkte Ausführung eines „störungsempfindlichen Befehls“ (↑sensitiver Befehl) durch die virtuelle Maschine zu verhindern. Ein solcher Befehl wird vom VMM abgefangen (↑*trap*) und speziell behandelt (↑partielle Interpretation). Typisches Beispiel ist die ↑Unterbrechungssperre. Wird diese durch eine ↑Aktion in einem ↑Gastbetriebssystem veranlasst, darf nur eine ↑Unterbrechungsanforderung an die virtuelle Maschine, die eben dieses Gastbetriebssystem ausführt, maskiert werden. In solch einem Fall fängt der VMM den entsprechenden ↑Maschinenbefehl (`cli`, x86) ab und unterbindet die mögliche ↑Unterbrechung einer auf der virtuellen Maschine stattfindenden Aktion. Der Zustand der ↑Wirtsmaschine bleibt diesbezüglich unverändert, das heißt, für sie gilt keine Unterbrechungssperre. Kommt der inverse Befehl (`sti`, x86) zur Ausführung durch die virtuelle Maschine, agiert der VMM dementsprechend.

virtuelle Adresse Bezeichnung der ↑Adresse von einem ↑Speicherwort, dessen Inhalt möglicherweise im ↑Hauptspeicher residiert. Eine ↑logische Adresse, die von der wirklichen Lokalität (↑Vordergrundspeicher oder ↑Hintergrundspeicher) der durch sie bezeichneten ↑Speicherstelle im ↑Arbeitsspeicher abstrahiert. Wird eine solche Adresse von der ↑CPU im ↑Abruf- und Ausführungszyklus appliziert, kann ein Zugriff darüber den Hauptspeicher verfehlen (↑*mainstore miss*). Ein solcher Fehlzugriff führt jedoch lediglich dazu, dass der betreffende ↑Prozess unterbrochen und, nachdem das ↑Betriebssystem sich eingemischt hatte, später wieder aufgenommen wird.

virtuelle Maschine Maschine, die nicht in Wirklichkeit existiert, aber echt (real) erscheint. Der ↑Prozessor dieser Maschine ist entweder ein ↑Interpreter oder ein ↑Übersetzer. Ersterer implementiert ein ↑Virtualisierungssystem bestimmter Art, das ein ↑Programm dieser Maschine direkt ausführen kann, wohingegen letzterer das auszuführende Programm in ein Programm

einer anderen (realen/virtuellen) Maschine transformiert, um es durch diese ausführen zu lassen.

virtueller Adressraum Bezeichnung für einen ↑Adressraum, der vom ↑Betriebssystem überwacht wird; eine bestimmte Menge von Nummern, wobei jede einzelne davon eine ↑virtuelle Adresse repräsentiert. Typischer Anwendungsfall für einen solchen Adressraum ist ↑virtueller Speicher, wodurch ein ↑Programm ablaufen kann, obwohl zur ↑Laufzeit nur Einzelbestandteile von ↑Text und ↑Daten davon im ↑Hauptspeicher vorzuliegen brauchen. Erst virtuelle Adressen erlauben die Abstraktion von der Lokalität von Informationseinheiten in Bezug auf die ↑Speicherhierarchie. Ein ↑logischer Adressraum hat diese Eigenschaft nicht, eine Adresse darin abstrahiert lediglich von der Lokalität solcher Einheiten innerhalb des Hauptspeichers: ein Programm muss hier also immer komplett im Hauptspeicher vorliegen, damit es ausgeführt werden kann. Zur Adressraumüberwachung nutzt das Beriebssystem bei Bedarf den ↑Hauptspeicherfehlzugriff durch einen ↑Prozess als Mechanismus. Dadurch erst kann es in den Prozess (ohne sein Wissen, aber dennoch zeitlich wahrnehmbar für ihn) eingreifen und nicht im ↑Hauptspeicher eingelagerte Text- und Datenbestände automatisch nachladen.

virtueller Speicher ↑Arbeitsspeicher, der „unecht“ vorhanden ist, nicht in Wirklichkeit existiert, aber echt erscheint. Wird durch ein ↑Betriebssystem bereitgestellt und anteilig auf den im ↑Rechensystem verfügbaren ↑Vordergrundspeicher (zur direkten Verarbeitung, ↑Hauptspeicher) und ↑Hintergrundspeicher (zur Zwischenspeicherung, ↑Ablagespeicher) abgebildet.

Vitruv (Marcus Vitruvius Pollio, 80–70 bis etwa 15 v. Chr.) altrömischer Baumeister, Bauingenieur, Architekt, Autor.

VM/370 Einplatz-/Einbenutzerbetriebssystem für eine ↑virtuelle Maschine im ↑Mehrprogrammbetrieb. Erste Installation 1972 (IBM System/370).

VMM Abkürzung für (en.) *virtual machine monitor*, (dt.) Steuerprogramm für eine ↑virtuelle Maschine, zentraler Bestandteil von einem ↑Virtualisierungssystem.

VMS Mehrplatz-/Mehrbenutzerbetriebssystem, Abkürzung für „*Virtual Memory System*“, erste Installation 1977 (DEC VAX-11). Hatte maßgeblichen Einfluss auf die Entwicklung von ↑Windows NT, das ab 1988 unter derselben Leitung (David Cutler) entstand.

volatile register (dt.) ↑flüchtiges Register.

Vollvirtualisierung Form der ↑Selbstvirtualisierung: die ↑Virtualisierung erfolgt ausschließlich durch einen ↑Hypervisor. Im Unterschied zur ↑Paravirtualisierung ist keinem ↑Prozess, weder im ↑Maschinenprogramm noch im ↑Betriebssystem, die Tatsache bekannt, dass die Hardware (↑CPU, ↑Peripherie), auf die er stattfindet, virtualisiert wird. Dies setzt jedoch virtualisierbare Hardware voraus, insbesondere die Fähigkeit der CPU, dass ein ↑sensitiver Befehl abgefangen werden kann (↑*trap*) und dann durch den Hypervisor zur Ausführung kommt (↑partielle Interpretation). Eine Hardware gilt als voll virtualisierbar, wenn ausnahmslos jeder sensitive Befehl derart behandelt werden kann.

volume (dt.) ↑Datenträger.

Vordergrundspeicher ↑Hauptspeicher; auch Primärpeicher. Flüchtiger ↑Speicher, der direkt über Lese-/Schreiboperationen der ↑CPU bedient wird.

Vorhersagbarkeit Grad, in dem eine korrekte Prädiktion (Vorhersage) des Zustands, Verhaltens oder Platz-, Energie- oder Zeitbedarfs von einem ↑Rechensystem möglich ist, in qualitativer oder quantitativer Hinsicht. Diese unterliegt immer bestimmten und vorher zu treffenden Annahmen. Insbesondere für ↑Echtzeit sind dabei folgende Aspekte bedeutsam: die Granularität eines Termins und Laxheit (weich, fest, hart) einer ↑Aufgabe, die Striktheit (weich, fest, hart) eines Termins, Zuverlässigkeitsanforderungen, die Größe des Systems und der Grad an Interaktion (↑Koordination) zwischen den Komponenten, sowie die Charakteristik der Umgebung, in der das System operieren muss.

vorlaufende Planung (en.) ↑*offline scheduling*. Modell der ↑Ablaufplanung, die ört- und zeitlich entkoppelt von den einzuplanenden ↑Prozessen geschieht: sie findet im Voraus statt und erzeugt einen gemeinhin statischen ↑Ablaufplan, der zu einem späteren Zeitpunkt vom ↑Betriebssystem abgearbeitet wird. Allerdings kann dieser Plan auch den Initialzustand einer ↑Bereitliste definieren, die dann durch eine anschließende ↑mitlaufende Planung aktualisiert und fortgeschrieben wird. Der ↑Planer ist nicht Bestandteil des Betriebssystems (örtlich entkoppelt), sondern getrennt davon als ↑Dienstprogramm verwirklicht, das zur Aufstellung des Ablaufplans erst zur Ausführung gebracht werden muss (zeitlich entkoppelt).
Diese Form von ↑Planung ist geboten, wenn die Berechnungskomplexität zur Erstellung des Ablaufplans (für eine gegebene Menge von Prozessen) einen Einsatz im laufenden Betrieb impraktikabel macht. Planung verursacht ↑Gemeinkosten, die in Grenzen zu halten sind, um, entsprechend der jeweils geforderten ↑Betriebsart, die eigentlich zu verwirklichenden Prozesse der ↑Maschinenprogramme beziehungsweise die ↑Peripherie bedienen zu können: ein Betriebssystem ist für gewöhnlich nur ausnahmsweise aktiv, was insbesondere bedeutet, nicht beliebig viel ↑Rechenzeit für sich selbst zu beanspruchen. Darüber hinaus geschieht Planung sehr oft vorlaufend, um jederzeit (möglichst exakte) Aussagen zur ↑Vorhersagbarkeit von Prozessen treffen zu können. Typischer Fall ist die ↑deterministische Planung. In allen Fällen ist zur Erstellung des Plans ↑Vorwissen unabdinglich.

Vorübersetzung Form der ↑Kompilation, um eine für die schnelle ↑Interpretation in Software besser geeignete Repräsentation von einem ↑Programm zu erhalten.

Vorwissen Kenntnis von etwas haben, mit der (lat.) *a priori* zuverlässige Aussagen möglich sind. Im Fokus stehen einerseits ↑Prozessgrößen, insbesondere Zeitpunkte, -intervalle und ↑Fristen. Andererseits kann dieses Wissen bevorzugt auch Beziehungen zwischen ↑Prozessen und ↑Betriebsmitteln beschreiben und damit letztlich auch Prozessabhängigkeiten repräsentieren. In all diesen Fällen sind die Informationen jedoch im Voraus bekannt und fließen als Parameter in die ↑Ablaufplanung ein.
Diese Größen und logischen Verknüpfungen sind dem ↑Planer „von außen“ vorgegeben, sie werden nicht zur ↑Laufzeit der zu planenden Prozesse erfasst. Sie beruhen auf Erfahrungswissen, empirische Untersuchungen, entwurfsbegleitende Modellanalyse, werkzeuggestützte Programmflussanalyse, messbasierte Ansätze (in Bezug auf Testläufe) oder Kombinationen davon. Kenntnis von solchen Informationen zu besitzen, ist obligatorisch für ↑deterministische Planung und optional, aber wünschenswert, für jede andere Form von ↑Planung.

VRR Abkürzung für (en.) *virtual round robin*. Bezeichnung für eine Strategie zur ↑Planung der Bearbeitung von ↑Aufgaben durch einen ↑Prozessor im Reihumverfahren (↑Rundlauf) mit Bevorzugung interaktiver ↑Prozesse. Eine Spezialisierung von ↑RR, mit der dem noch (in abgeschwächter Form) verbleibenden ↑Konvoieffekt vorgebeugt werden kann. Dazu wird neben der ↑Bereitliste eine weitere ↑Warteschlange eingeführt, in die Prozesse gelangen, die:

1. ihren letzten ↑Rechenstoß beendeten, indem sie einen ↑Ein-/Ausgabestoß, dessen Beendigung sie erwarten müssen, ausgelöst haben und
2. durch Beendigung ihrer ↑Ein-/Ausgabe wieder bereitgestellt wurden.

Die Prozesse in dieser zusätzlichen Warteschlange sind also nicht durch Ablauf ihrer ↑Zeitscheibe zur Abgabe des ↑Prozessors gezwungen wurden, sie gelten daher nicht als rechenintensiv (↑*CPU bound*), sondern sind vermutlich ein-/ausgabeintensiv (↑*I/O bound*). Steht erneut die ↑Einlastung an, wird zunächst versucht, aus der Warteschlange ein-/ausgabeintensiver Prozesse die nächste Aufgabe für den Prozessor zu entnehmen. Ist diese *Vorzugsliste* leer, kommt die nächste Aufgabe aus der Bereitliste und damit dann ein als vermutlich rechenintensiv eingestufter Prozess zur Wirkung.
Einziges Moment, durch das die ↑Verdrängung eines stattfindenden Prozesses erreicht wird, ist der mittels einer ↑Unterbrechungsanforderung angezeigte Zeitscheibenablauf. Folge der diesbezüglichen ↑Unterbrechungsbehandlung wird die Verdrängung des unterbrochenen Prozesses sein, wenn wenigstens ein anderer Prozess auf der Vorzugs- oder Bereitliste steht. Die

das Ende einer Ein-/Ausgabe anzeigende Unterbrechungsanforderung führt demgegenüber nicht zur zeitnahen Einlastung des betreffenden Prozesses, sondern lediglich zu seiner ↑Einplanung unter Verwendung der Vorzugsliste. Frühestens mit Ablauf der aktuellen Zeitscheibe wird diesem Prozess dann der Prozessor zugewiesen, nämlich wenn dem Prozess bei der Einplanung die höchste ↑Dringlichkeit zuerkannt wurde.
Die Bearbeitung der Aufgaben in der Bereitliste erfolgt strikt nach RR, ihre Reihungstrategie ist ↑FCFS und benötigt daher eine konstante ↑Laufzeit für das Einsortieren. Demgegenüber sind die Aufgaben in der Vorzugsliste entsprechend der Restdauer der Zeitscheibe des jeweils zugehörigen Prozesses geordnet, womit einsortieren einen linear ansteigenden Laufzeitaufwand mit sich bringt. Für beide Listen gilt jedoch, dass die jeweils nächste Aufgabe am Kopfende (d.h., vorne) und damit in konstantem Aufwand entfernt wird.
Unabhängig davon, welcher der beiden Listen die nächste Aufgabe beziehungsweise der zugehörige Prozess entstammt, der Prozessor wird nur maximal eine Zeitscheibe lang für die Aufgabenbearbeitung belegt: kein Prozess kann daher den Prozessor bewusst oder unbewusst monopolisieren (↑*CPU protection*). Wurde der Prozess der Bereitliste entnommen, erhält er die ganze Zeitscheibe an Rechenzeit zugeteilt. Kam der Prozess allerdings aus der Vorzugsliste, belegt er den Prozessor längstens für die Restdauer seiner „angebrochenen“ Zeitscheibe. Der ↑Zeitgeber wird, im Gegensatz zu RR, also nicht mit einem konstanten Wert für alle Prozesse programmiert, sondern mit einem variablen Wert, der prozessabhängig ist und damit ein Attribut des jeweiligen ↑Prozesskontrollblocks bildet. Grundsätzlich gilt: Wann immer ein Prozess durch Ablauf seiner (vormals ganzen oder zerstückelten) Zeitscheibe eine Unterbrechung erfährt und dadurch zur Prozessorabgabe gezwungen wird, so kommt er zurück auf die Bereitliste.

Exkurs Ausgehend von den Implementierungsskizzen für FCFS und RR sind Änderungen in vergleichsweise geringem Umfang erforderlich, wobei zwei Operationen (`ready`, `elect`) im Fokus stehen. Darüber hinaus sind nunmehr zwei Listen zu führen, deren Einträge zur Einlastung bereite Aufgaben beziehungsweise Prozessen repräsentieren. Ein diesbezüglicher ↑Datentyp sei wie folgt aufgebaut:

```
typedef struct backlog {
    queue_t ready;                /* normal list of ready-to-run processes */
    chain_t favor;               /* priority list of ready-to-run processes */
} backlog_t;
```

Dieser Datentyp ist eine einfache Erweiterung des vorher bereits für FCFS oder RR benutzten Datentyps, der dort jedoch nur die normale Bereitliste (`ready`) als Attribut hat. Hier nun kommt eine Liste von vorzuziehenden Aufgaben beziehungsweise Prozessen hinzu (`favor`), wobei die Listeneinträge nach aufsteigender Restdauer der jeweiligen Zeitscheibe sortiert sind. Die Einträge in dieser Liste sind ↑Exemplare des folgenden Datentyps:

```
typedef struct entry {
    chain_t next;                              /* successor on some waitlist */
    int rank;                                      /* sort key of this entry */
} entry_t;
```

Neben dem Attribut zur Verkettung (`next`) enthält jeder Eintrag einen Sortierschlüssel (`rank`), der letztlich die Position des vorzuziehenden Prozesses in Abhängigkeit von den anderen Prozessen seiner Art in der Liste festlegt. Im gegebenen Fall ist der Sortierschlüssel gleich dem Wert der Restdauer der Zeitscheibe des Prozesses, dessen ↑Prozesskontrollblock (`process_t`) einen solchen Eintrag als Attribut (`line`) umfasst. Im Moment der Beendigung der Ein-/Ausgabe wird ein auf dieses ↑Ereignis wartender Prozess bereitgestellt, indem sein Prozesskontrollblock entsprechend seines Ranks auf die Vorzugsliste kommt:

```
void favor(process_t *task) {          /* schedule according to accuteness */
    state(&task->mood, FLUSH|READY);          /* set process ready to run */
    ticket(&task->line, task->rest);                    /* define sort key */
    enlist(&labor()->favor, &task->line);      /* sort into priority list */
}
```

Prozesse, die der Vorzugsliste entnommen werden, belegen den Prozessor maximal für die Restdauer ihrer Zeitscheibe. Wann immer im Moment des Ablaufs einer Zeitscheibe wenigstens ein bereitstehender Prozess die Prozessorzuteilung erwartet, wird der gegenwärtig stattfindende Prozess zurück auf die normale Bereitliste gesetzt:

```
void ready(process_t *task) {                /* schedule according to FCFS */
    state(&task->mood, READY);                 /* set process ready to run */
    enqueue(&labor()->ready, &task->line);           /* add to ready list */
}
```

Dies ist übrigens ebenfalls die Vorgehensweise für jeden Prozess, der erstmalig (d.h., nach seiner Erzeugung) bereitgestellt wird. Im Gegensatz zur Vorzugsliste belegt jeder Prozess, der der Bereitliste entnommen wird, den Prozessor für maximal eine volle Zeitscheibe. Für die Bestimmung eines zur Einlastung des Prozessors bereitstehenden Prozesses wird sodann wie folgt aus einer der beiden Listen ausgewählt:

```
process_t *elect(backlog_t *list) {         /* choose ready-to-run process */
    chain_t *item =
        chain(&list->favor) ? unbag(&list->favor) : dequeue(&list->ready);
    return forge(item);
}
```

Demnach wird der Prozess mit dem kleinsten Zeitscheibenwert bevorzugt, wenn die Vorzugsliste nicht leer ist (`unbag`). Dies ist immer der ganz oben, vorne, am Kopfende, auf der Vorzugsliste (durch `favor`) verzeichnete Prozess. Ansonsten wird der nächste Prozess, dem dann eine ganze Zeitscheide zuzuteilen ist, dem Kopf der Bereitliste entnommen (`dequeue`) — in beiden Fällen ist aus dem Kettengliedzeiger noch ein Prozesszeiger zu formen (`forge`: vgl. FCFS, S. 48). Die Programmierung des Zeitgebers mit dem prozessabhängigen Zeitscheibenwert geschieht bei der Einlastung, im Gegensatz zu RR, da hier die Zeitscheibenlänge für alle Prozesse identisch ist. Im Fall von RR erfolgt für gewöhnlich höchstens noch die Aktivierung des Zeitgebers, nämlich im Moment des ↑Prozesswechsels durch den ↑Umschalter. Alle anderen Operationen des ↑Planers bleiben im Vergleich zu RR unverändert.

Wagenrücklauf Vorgang der Rückführung des Läufers (↑*cursor*) an den Zeilenanfang (↑CR) einer ↑Dialogstation, häufig auch kombiniert mit einem Zeilenwechsel (↑LF). Insbesondere auch zur Auslösung einer meist durch Betätigung der ↑Eingabetaste abgegebenen Anweisung zur Ausführung einer ↑Kommandozeile. Der Begriff hat seinen Ursprung in der Schreibmaschine beziehungsweise dem ↑Fernschreiber, wo am Zeilenende manuell (durch den Zeilenschalthebel der Schreibmaschine) oder automatisch (durch Tastendruck oder ein Steuerzeichen beim Fernschreiber) der Wagen (*carriage*) mit den Typenhebeln oder dem Kugelkopf auf den Anfang einer neuen Zeile positioniert werden musste.

wait-freedom (dt.) ↑Wartefreiheit.

Wartefreiheit (en.) ↑*wait-freedom*. Fortschrittsgarantie für ↑nichtblockierende Synchronisation. Diese Garantie besagt, dass jeder ↑Prozess jede Operation in einer endlichen Anzahl von Schritten vollenden kann, ohne Rücksicht auf die relativen Geschwindigkeiten der anderen Prozesse. Eine stärkere Zusicherung als ↑Sperrfreiheit, da jedem einzelnen Prozess in einer ↑Konkurrenzsituation Fortschritt garantiert wird.

Warteschlange Konstrukt für aufgelaufene, unerfüllte und zeitweilig zurückgestellte Aufträge. Bildet sich, wenn in einem Zeitintervall mehr Aufträge eintreffen, als in demselben Inter-

vall verarbeitet werden können. Ist ein solcher Auftrag etwa ein ↑Prozess, so bedeutet beispielsweise eine gefüllte ↑Bereitliste, dass zu wenig ↑Betriebsmittel der Art ↑Prozessor zur Verfügung stehen. Hinzufügen weiterer Prozessoren kann zum Abbau der Liste beitragen, oder die gerade verfügbaren Prozessoren werden im ↑Zeitteilverfahren entsprechend einer bestimmten Strategie bedient, bis die Liste geleert werden konnte. Die Strategie ist im ↑Einplanungsalgorithmus verankert.

Warteverhalten Art und Weise, wie ein ↑Prozess dem Eintreffen von einem ↑Ereignis entgegensieht: ↑aktives Warten oder ↑passives Warten. Das Ereignis, auf das der Prozess wartet, definiert einen bestimmten Zustand, der gelten muss, damit der Prozess in seinem ↑Programm weiter voranschreiten kann. Typisches Beispiel ist der von einem Prozess ausgelöste ↑Ein-/Ausgabestoß. Hängt der Prozess von einzugebenden ↑Daten ab, muss er die Beendigung des diesbezüglichen Eingabestoßes abwarten: um Fortschritt erzielen zu können, benötigt der Prozess Eingabedaten, die ihm am Ende dieses Stoßes zur Verfügung stehen und dann durch ein Signal, ein ↑konsumierbares Betriebsmittel, angezeigt werden. Möchte der Prozess Daten ausgeben und ist jedoch der dazu benötigte Ausgabepuffer voll, muss der Prozess die Beendigung des diesbezüglichen Ausgabestoßes abwarten: um Fortschritt erzielen zu können, benötigt der Prozess ein ↑wiederverwendbares Betriebsmittel, den Puffer, der durch den Ausgabestoß geleert wird (↑DMA) und mit Ende dieses Stoßes wieder Platz für weitere auszugebende Daten hat. Aber auch ohne nebenläufigem Ein-/Ausgabestoß — die ↑Nebenläufigkeit bezieht sich auf den jeweiligen Stoß, kausal abhängig ist der Prozess auf das dem Stoßende entsprechenden Ereignis — kann ein Prozess in Wartesituationen kommen, nämlich wenn er allgemein ↑Synchronisation mit anderen Prozessen erzielen muss.

Wartezeit Bezeichnung für eine ↑Prozessgröße, die die Zeitspanne von der Unterbrechung bis zur Fortsetzung einer Operation oder Operationsfolge quantifiziert.

WCET Abkürzung für (en.) *worst-case* ↑*execution time*, größte anzunehmende ↑Ausführungszeit.

Wechseldatenträger ↑Speichermedium, das nicht fest in einem ↑Rechensystem eingebaut ist. Das Medium ist austauschbar, es wird durch ein am Rechensystem angeschlossenes ↑Peripheriegerät zum Speichern und Abrufen (gespeicherter) Informationen zugänglich gemacht. Das ↑Betriebssystem ermöglicht einem ↑Prozess den Zugang zu diesem Gerät per ↑Systemaufruf. Die im Betriebssystem stattfindende Interaktion mit dem Gerät bewerkstelligt ein ↑Gerätetreiber, der speziell auf die Eigenschaften dieses Geräts zugeschnitten ist. Beispiele für solche heute (2016) eingesetzten Medien sind der Lochstreifen, die Lochkarte, das Magnetband, die Magnetplatte, flexible Magnetplatte (*floppy disc*), optische Speicherplatte (↑CD, ↑DVD), Speicherkarte (*flash/memory card*, ↑SSD) oder der Speicherstab (*memory stick*, ↑USB).

wechselseitiger Ausschluss Form von ↑multilaterale Synchronisation, auch Inbegriff für ↑blockierende Synchronisation. Sorgt dafür, dass ein ↑kritischer Abschnitt stets nur von höchstens einen ↑Prozess betreten und durchstreift werden kann. Jeder der beteiligten Prozesse erwartet *aktiv* (↑Umlaufsperre) oder *passiv* (↑binärer Semaphor, ↑Mutex) das ↑Ereignis, dass der von einem Prozess erworbene kritische Abschnitt wieder freigesetzt wird. Ein Prozess wartet aktiv, wenn er durch anhaltendes Abfragen (*polling*) seiner Wartebedingung während seiner Wartezeit beschäftigt bleibt (↑*busy waiting*). Im Gegensatz dazu wartet ein Prozess passiv, wenn er blockiert (↑Prozesszustand) und dem ↑Ereignis, dass der kritische Abschnitt verlassen wird, „schlafend“ entgegensieht.

Wettlaufsituation (en.) ↑*race condition*, ↑*race hazard*. Umstand, bei dem ein ↑nichtsequentielles Programm eine ↑Aktion oder ↑Aktionsfolge mit unbestimmten zeitlichen Verhalten zulässt. Konsequenz daraus können inkorrekte Programmabläufe sein, die Fehlverhalten verursachen. Dem zugrunde liegen asynchrone Ereignisse, die die Programmabläufe nicht nur unvorhersagbar, sondern auch nicht reproduzierbar machen. Zur Vorbeugung des möglichen Fehlverhaltens sind Maßnahmen zur ↑Nebenläufigkeitssteuerung erforderlich.

Wiedereintritt (en.) ↑*re-entrance*. Moment, in dem ein ↑Programm erneut, und zwar verschachtelt, zur Ausführung kommt. Ursache ist eine vorangegangene ↑Ausnahme, erhoben während der Programmausführung, wobei die ↑Ausnahmebehandlung Bestandteil des Programms selbst ist. Im Zuge dieser Behandlung wird ein in seiner Ausführung zuvor unterbrochener Programmabschnitt (einer indirekten ↑Rekursion nicht unähnlich) wiederholt betreten. Zur korrekten Ausführung ist für das einen solchen Abschnitt enthaltene Programm ↑Ablaufinvarianz gefordert.

Wiederholungslauf Erneute ↑Interpretation von einem ↑Maschinenbefehl, bei dessen Verarbeitung zuvor im ↑Abruf- und Ausführungszyklus der ↑CPU eine ↑Ausnahmesituation eingetreten ist. Das ↑Betriebssystem hat die mit dem Maschinenbefehl verbundene ↑Aktion abgefangen (↑*trap*), die erhobene ↑Ausnahme behandelt (↑partielle Interpretation) und hinterlässt für die CPU einen ↑Prozessorstatus, der nach der Beendigung der ↑Ausnahmebehandlung zur Wiederholung der Ausführung des abgefangenen Maschinenbefehls führt (↑*rerun bit*).

wiederverwendbares Betriebsmittel Bezeichnung für ein ↑Betriebsmittel, das nur in begrenzter Anzahl vorhanden ist und von dem es eine feste Anzahl von Einheiten gibt. Jede dieser Einheiten ist entweder verfügbar oder belegt, das heißt, keinem oder einem ↑Prozess zugewiesen. Mit erfolgter Zuweisung hat ein Prozess die von ihm angeforderte Betriebsmitteleinheit erworben (*acquire*). Ist die Mitbenutzung desselben Betriebsmittels ausgeschlossen (↑unteilbares Betriebsmittel), können Einheiten davon zu einem Zeitpunkt nur maximal einem Prozess zugewiesen sein. Ein Prozess kann beliebige von ihm belegte Einheiten von Betriebsmitteln freisetzen (*release*), sofern er seinerseits nicht den Erwerb einer Betriebsmitteleinheit erwartet und demzufolge blockiert ist. Erworbene Einheiten können dem Prozess nicht entzogen werden, sie sind erst wieder verfügbar, nachdem sie durch den Prozess explizit freigesetzt wurden.

Windhundprinzip Bezeichnung für ein Planungsverfahren, das ein ↑wiederverwendbares Betriebsmittel ausschließlich nur in der zeitlichen Reihenfolge des Eintreffens von Bedarfsanmeldungen zuteilt („wer zuerst kommt, mahlt zuerst“). Typisches Beispiel ist ↑FCFS.

Windows Mehrplatz-/Mehrbenutzerbetriebssystem, erste Installation 1985 (Intel 8086), zunächst lediglich als fensterbasierte graphische Benutzeroberfläche zur Erleichterung des Umgangs mit ↑MS-DOS. Erst seit ↑Windows NT mehrplatz-/mehrbenutzerfähig.

Windows NT Mehrplatz-/Mehrbenutzerbetriebssystem, erste Installation 1993 (Intel 80286). Steht für ↑Windows „New Technology“ — oder Halbgeschwister von ↑VMS, dem unter derselben Leitung (David Cutler) entstandenem ↑Betriebssystem: WNT entwickelt sich sich aus VMS, indem positionsweise ein Schritt im Alphabet weitergegangen wird.

Wirtsbetriebssystem ↑Betriebssystem, das ein ↑Gastbetriebssystem bewirtet. Letzteres kann in der Bereitstellung einer eigenen ↑Systemfunktion von den (per ↑Systemaufruf zugänglichen) Systemfunktionen des Wirtssystems profitieren. Darüberhinaus kann mehr als ein Gastsystem von dem Wirtssystem bedient werden, ohne dass ein ↑Prozess des Gastsystems dies in funktionaler Hinsicht wahrnimmt. Eine solche Mehrfachnutzung kann sich *homogen* (gleichartige Gastsysteme) oder *heterogen* (verschiedenartige Gastsysteme) darstellen, ohne dass dies wiederum dem Wirtssystem bewusst ist.

Wirtsmaschine Maschine realer oder virtueller Natur, die eine ↑virtuelle Maschine bewirtet: letztere benutzt (↑Benutzthierarchie) erstere. Dies bedeutet in erster Linie, dass das ↑Programmiermodell der benutzten Maschine ebenfalls für die benutzende Maschine gilt. Typischerweise läuft der ↑VMM als einziges ↑Programm auf dieser Maschine, um eben eine oder mehrere virtuelle Maschinen zu realisieren.

work sharing (dt.) ↑Arbeitsteilung.

work stealing (dt.) ↑Arbeitsentzug.

working set (dt.) ↑Arbeitsmenge.

worst fit (dt.) schlechteste Passung: ↑Platzierungsalgorithmus.

Wort Kurzbezeichnung für ein ↑Maschinenwort oder ↑Speicherwort.

Wortbreite Größe (Bitanzahl) von dem ↑Maschinenwort einer ↑CPU. Typisch waren oder sind 8, 9, 12, 16, 18, 24, 32, 36, 39, 40, 48, 60 und 64 Bits pro Wort (Stand 2016).

Wurzelprozedur Oberprogramm, auch übergeordnetes ↑Programm, das ein ↑Unterprogramm aufruft, aber selbst nicht als Unterprogramm aufgerufen wird.

Wurzelsegment Bezeichnung für ein ↑Segment, das die Ansatzstelle (Wurzel) für die hardwaregestützte ↑Segmentierung von einem ↑Adressraum bildet (↑segmentierter Adressraum). In diesem Segment liegt die globale, jedem ↑Prozessadressraum gemeinsame ↑Segmenttabelle (↑Multics).

Wurzelverzeichnis Bezeichnung für ein ↑Verzeichnis, das die Ansatzstelle (Wurzel) von einem ↑Dateisystem bildet. Von dieser Stelle aus ist jede in dem Dateisystem erfasste ↑Entität erreichbar, was insbesondere auch für das ↑Einhängen eines Dateisystems von Bedeutung ist. In dem Fall nämlich wird der ↑Befestigungspunkt (ein Verzeichnis) in einem Dateisystem zur Wurzel des eingehängten Dateisystems.

x86 Prozessorarchitektur, 16/32-Bit, abwärtskompatibel zum Intel 8086 (1978).

Zeilendrucker ↑Peripheriegerät das alle Zeichen einer Zeile gleichzeitig druckt. Die durch einen zugehörigen ↑Gerätetreiber zur Ausgabe nacheinander mittels ↑Ein-/Ausgaberegister bereitgestellten ↑Daten werden zeilenweise in dem Gerät gepuffert und dann auf ↑Tabellierpapier ausgegeben. Der Gerätetreiber ist für gewöhnlich ein ↑Unterprogramm von einem Steuerprogramm zur Organisation und Überwachung des Rechnerbetriebs (↑*resident monitor*). Falls ↑abgesetzter Betrieb geführt wird, ist das Gerät selbst am ↑Satellitenrechner angeschlossen, ansonsten am ↑Hauptrechner.

Zeilenvorschub Setzen der ↑Schreibmarke auf die nächste Zeile, ohne dabei die aktuelle Position des Läufers (↑*cursor*), die er innerhalb der vorigen Zeile besaß, zu verändern. Ursprüngliche Bezeichnung für das Drehen der Walze bei einer Schreibmaschine, um das Papier eine Zeile vorwärts zu transportieren. Für den ↑Fernschreiber wurde für einen solchen Zeilenwechsel ein spezielles Steuerzeichen (↑LF) eingeführt, das nach wie vor den Zeilenumbruch in einer ↑Dialogstation bewirkt.

Zeitfenster Zeitspanne fester Länge. Ein begrenztes Zeitkontingent für ein bestimmtes ↑Ereignis oder eine Folge festgelegter Geschehnisse.

Zeitgeber ↑Peripheriegerät, mit dem zeitlich gebundene Vorgänge geregelt werden können. Das Gerät sendet eine ↑Unterbrechungsanforderung an die ↑CPU, wenn ein vom ↑Gerätetreiber vorprogrammiertes Zeitintervall abgelaufen ist.

Zeitreihe Bezeichnung für eine Folge von Zeitgrößen, die nach einer bestimmten Gesetzmäßigkeit, in einem bestimmten regelmäßigen Abstand aufeinanderfolgen (in Anlehnung an den Duden). Eine geordnete Aufstellung von Zeitlängen. Typisches Beispiel einer solchen Reihe wäre eine Aufstellung der Stoßlängen (↑Rechenstoß oder ↑Ein-/Ausgabestoß), die ein jeder ↑Prozess innerhalb einer bestimmt Zeitspanne hervorbringt.

Zeitreihenanalyse Ermittlung der Einzelbestandteile einer ↑Zeitreihe mittels empirischer Methoden, mit dem Ziel, durch inferenzstatistische Untersuchung eine Vorhersage über die weitere Entwicklung dieser Folge treffen zu können. Die einzelnen Zeitgrößen beruhen auf Beobachtung oder Erfahrung, sie repräsentieren Werte, die bei der Nutzung von ↑Betriebsmitteln erhoben und pro ↑Prozess erfasst werden.

Typisches Beispiel eines solchen Nutzungswerts ist die jeweilige Stoßlänge eines Prozesses, und zwar sowohl ↑Rechenstoß als auch ↑Ein-/Ausgabestoß. Für gewöhnlich bringt ein Prozess mehrere solcher Stöße abwechselnd hervor. Mit jeder Stoßart wird der Prozess eine eigene Zeitreihe bilden, deren jeweilige Gliederanzahl prozessabhängig ist und daher auch als verschieden beziehungsweise variabel gilt. Zeitreihen mit eher langen Rechenstößen beziehungsweise kurzen Ein-/Ausgabestößen deuten auf rechenintensive Prozesse, die den ↑Prozessor häufig lange für Berechnungen belegen (↑*CPU bound*). Demgegenüber deuten Zeitreihen mit eher langen Ein-/Ausgabestößen an, dass hier ein-/ausgabeintensive Prozesse am Werke sind (↑*I/O bound*). Schließlich lassen Zeitreihen mit eher kurzen Rechenstößen vermuten, vermehrt interaktive Prozesse am Laufen zu haben. Diese durch Untersuchung einzelner Zeitreihen mögliche Charakterisierung von Prozessen kann von der ↑Ablaufplanung in verschiedener Hinsicht gewinnbringend genutzt werden, beispielsweise um die Auslastung der ↑CPU oder der ↑Peripherie zu maximieren oder die Ansprechempfindlichkeit des ↑Betriebssystems zu verbessern.
Ist der Nutzungswert eines von einem Prozess zu belegenden Betriebsmittels unbekannt, entscheidet dieser Wert jedoch über die ↑Dringlichkeit der Betriebsmittelzuteilung, kann aus entsprechend beobachteten Werten in der Vergangenheit — oder während eines ↑Produktionsbetriebs — auf den zu veranschlagenden Wert in der Zukunft geschlossen werden. Dabei wird angenommen, dass das Nutzungsverhalten eines Prozesses in Bezug auf ein bestimmtes Betriebsmittel mit gewisser Wahrscheinlichkeit auch zukünftig gleich bleibt. So liegt es nahe, den zu erwartenden Nutzungswert s_{n+1} als *arithmetisches Mittel* der bisher wirklich gemessenen Nutzungswerte s_i, $1 \leq i \leq n$, zu definieren:

$$s_{n+1} = \frac{1}{n} \cdot \sum_{i=1}^{n} t_i$$

Der Nachteil dieser Lösung ist jedoch, dass sie eben alle Phasen, in denen ein Prozess ein bestimmtes Betriebsmittel genutzt hat, gleich bewertet. Lange zurückliegende Aktivitäten des Prozesses haben damit die gleiche Wichtigkeit wie seine zum gegenwärtigen Zeitpunkt stattfindenden Aktivitäten. Ein besserer Ansatz ist es daher, Nutzungswerte mit zunehmenden Abstand („Alter“) vom gegenwärtigen Zeitpunkt schwächer zu gewichten:

$$s^*_{n+1} = \frac{1}{w} \cdot s_{n+1} + \frac{w-1}{w} \cdot s^*_n,$$

wobei $1/w$ den *Wichtungsfaktor* α beschreibt. Die Prognose wird jedoch viel direkter das aktuelle Prozessverhalten reflektieren, wenn die Mittlung nicht den Mittelwert s_{n+1} aller bisher gemessenen Nutzungswerte betrifft, sondern lediglich den jeweils zuletzt gemessenen Wert t_n (↑exponentielle Glättung):

$$s^*_{n+1} = \alpha \cdot t_n + (1-\alpha) \cdot s^*_n, \text{ mit } 0 \leq \alpha \leq 1$$

Abschätzungen dieser Art sind typisch für die ↑mitlaufende Planung von Prozessen auf Grundlage ihrer jeweiligen Rechenstoßlängen, beispielsweise wie im Falle von ↑SPN.

Zeitscheibe Zeitintervall, für das ein ↑Prozess den ↑Prozessor vom ↑Planer zugeteilt bekommt.

Zeitteilverfahren ↑Multiplexverfahren, das jedem ↑Prozess immer nur für ein bestimmtes Zeitintervall (↑*time slice*) den ↑Prozessor zuteilt. Technische Grundlage dafür bildet ein von dem ↑Planer benutzter ↑Zeitgeber. Der Ablauf des Zeitintervalls bewirkt eine ↑Unterbrechung des laufenden Prozesses, woraufhin der Planer als Teil der ↑Unterbrechungsbehandlung aufgerufen wird. In dieser Situation wird der Planer dem unterbrochenen Prozess bedingt den Prozessor entziehen, das heißt, eventuell einen ↑Prozesswechsel erzwingen: der unterbrochene Prozess wird vom Prozessor verdrängt (↑*preemption*). Dies geschieht jedoch nur, wenn neben dem logisch noch laufenden (↑Prozesszustand), aber tatsächlich unterbrochenen, Prozess wenigstens ein weiterer Prozess bereit zur ↑Einlastung zur Verfügung steht und dieser

keine niedrigere Priorität als der unterbrochene Prozess besitzt. In dem Fall wird der unterbrochene Prozess auf die ↑Bereitliste gesetzt und ein anderer Prozess (mit gleicher/höherer Priorität), zu dem dann als nächster gewechselt werden soll, davon herunter genommen. Gibt es keinen solchen Prozess, wird der unterbrochene Prozess für ein weiteres Zeitintervall fortgesetzt. Je nach Verfahren ist das Zeitintervall fest oder variabel.

Zentraleinheit Mittelpunkt von einem ↑Rechensystem in logischer Hinsicht, durch dem alle anderen Komponenten der ↑Peripherie kontrolliert und gesteuert werden. Synonym für ↑CPU.

Zugriffskontrollliste Datenstruktur, die für jedes ↑Subjekt das ↑Zugriffsrecht auf ein und dasselbe ↑Objekt speichert: die Liste ist physischer Bestandteil des Objekts und nicht des Subjekts (im Gegensatz zur ↑Befähigung). Dabei ist ein Listeneintrag ein Paar von ↑Prozessidentifikation (Subjekt) und Zugriffsrecht. Bei Auslösung einer ↑Aktion bezogen auf ein bestimmtes Objekt wird der Listeneintrag anhand der Identifikation für den betreffenden ↑Prozess aufgesucht: der Auftrag des Prozesses für die gewünschte Aktion durchläuft eine Eingangsprüfung. Schlägt die Suche fehl oder ist das im gefundenen Listeneintrag verbuchte Zugriffsrecht für den Prozess unzureichend, tritt eine ↑Ausnahmesituation ein. Steht der Prozess gegebenenfalls mehrmals auf der Liste, wird für gewöhnlich der zuerst gefundene Listeneintrag für ihn zur Rechteüberprüfung herangezogen: der zuerst gelistete Eintrag für einen Prozess dominiert nachfolgende Einträge desselben Prozesses. Der Widerruf (*revocation*) eines Zugriffsrechts ist einfach und bedeutet lediglich, einen bestimmten Listeneintrag zu löschen. Für gewöhnlich wird in dem Fall immer der dominierende Listeneintrag gelöscht. Auch der Widerruf aller Zugriffsrechte eines Prozesses auf dasselbe Objekt ist leicht möglich, indem lediglich alle Listeneinträge dieses Prozesses für das Objekt zu löschen sind.

Zugriffsmatrix System, das einzelne Faktoren von ↑Subjekt und ↑Objekt darstellt und zur verkürzten Beschreibung von Zugriffsrechten dient. Typischerweise sind die Spalten der Matrix durch eine endliche Menge von Objekten und die Zeilen der Matrix durch eine endliche Menge von Subjekten definiert. Jeder Matrixeintrag beschreibt dann die Menge von Zugriffsrechten, die ein Subjekt auf ein Objekt besitzt.
Für den praktischen Einsatz in einem ↑Betriebssystem, um nämlich die Zugriffsrechte von jedem ↑Prozess (Subjekt) auf die vorhandenen ↑Betriebsmittel (Objekte) zu beschreiben, ist die Matrixdarstellung ungeeignet. Da die Prozess für gewöhnlich nur auf einen Teil der Betriebsmittel zugreifen, wäre die Matrix nur sehr dünn besetzt. Um Speicherplatz einzusparen, wird die Matrix daher entweder zeilen- oder spaltenweise abgelegt. Im Falle der zeilenweise Speicherung bedeutet dies dann, dass jedem Prozess eine ↑Befähigung auf die von ihm verwendeten Betriebsmittel gegeben wird. Wohingegen die spaltenweise Speicherung jedem Betriebsmittel eine ↑Zugriffskontrollliste zuordnet, in der dann nur die Prozesse verzeichnet sind, die dieses Betriebsmittel verwenden wollen.

Zugriffsrecht Erlaubnis, Dinge oder Rechte zu nutzen, die nicht allgemein zugänglich sind (Duden). Das Ding ist ein ↑Objekt und das Recht gibt an, dass und gegebenenfalls auch in welcher Weise auf dieses Objekt durch einen ↑Prozess zugegriffen werden darf. Bereits Kenntnis von der ↑Adresse eines Objekts kann einem Prozess das Recht zum Zugriff darauf gewähren (↑Einadressraummodell). Für gewöhnlich trifft das jedoch nur auf Adressen zu, die (a) nur schwer oder nicht erraten werden können und (b) einem ↑Adressbereich angehören, dessen Größe ein bloßes Ausprobieren von Adressen impraktikabel macht. Das Recht allein nur zum Zugriff kann einem Prozess genommen oder gezielt erteilt werden (↑logischer Adressraum). Eine weitere Differenzierung ist über die Art des Zugriffs möglich, beispielsweise lesen, schreiben und ausführen für sehr einfache Objekte als ↑Seite oder ↑Segment in einem ↑Prozessadressraum. Darüberhinaus können komplexe Operationen wie etwa erzeugen, eröffnen, vergrößern, verkleinern, verschmelzen, aufteilen, duplizieren oder zerstören weitere Objektrechte definieren. Die möglichen Rechte sind somit von dem Typ des Objektes selbst abhängig: so wird eine einzelne ↑Speicherzelle oder ein ↑Maschinenwort nur gelesen oder beschrieben werden können, für einen ↑Verbund oder eine ↑Datei dagegen können weit umfangreichere Rechte gelten.

Zugriffszeit Zeitspanne von der Auslösung bis zur Ausführung einer Operation oder Operationsfolge.

zurückgestellter Prozeduraufruf (en.) ↑*deferred procedure call*. Mechanismus in einem ↑Betriebssystem (urspr. ↑Windows NT), um vorangigen ↑Prozessen die nachgeschaltete Durchführung von nachrangigen ↑Aufgaben zu ermöglichen und diese dadurch nicht unnötig zu verzögern. Ein typisches Beispiel ist das Zusammenspiel zwischen einem ↑FLIH und seinem ↑SLIH, wobei letzterer eine Prozedur darstellt, deren Aufruf zurückgestellt ist.

Zustandssicherung Maßnahme zum Sicherstellen der augenblicklichen physischen Beschaffenheit von einem ↑Prozess in Bezug auf den ihm zugeteilten ↑Prozessor. Die betrifft wenigstens den ↑Prozessorstatus, kann jedoch auch bestimmte Datenbestände umfassen, die über den ↑Adressraum eines Prozesses zugänglich sind. Beispiel für letzteres ist das Ziehen einer Sicherungskopie (*backup*) von im ↑Hauptspeicher liegende Datenstrukturen, um ein Zurückrollen (*rollback*) des Prozesses im Rahmen der Erholung (*recovery*) nach dem Scheitern einer ↑Aktion oder einem Systemausfall duchführen zu können. Im einfachsten Fall wird mit der Maßnahme nur der Prozessorstatus gesichert, um einen unterbrochenen Prozess vom Prozessor wegschalten und später auf demselben oder einem anderen Prozessor, jedoch mit demselben ↑Programmiermodell, wieder fortsetzen zu können.

Zwischenankunftszeit Zeitspanne zwischen zwei aufeinanderfolgenden Aufträgen an einer Bedienstation (↑Prozessor, ↑Peripheriegerät), allgemein zwischen zwei Ereignissen derselben Art. Gemessen über mehrere solcher Aufträge/Ereignisse wird gegebenenfalls unterschieden zwischen Minimal-, Durchschnitts- und Maximalwerten für diese Zeitspanne. So ist beispielsweise im Falle einer ↑Unterbrechung zwar nicht ihr exakter Zeitpunkt vorhersehbar, nicht selten jedoch die *minimale Zeitspanne* zwischen zwei solcher Ereignisse.

Zwischenkode ↑Maschinenkode für eine ↑virtuelle Maschine, der durch einen ↑Interpreter, der in Software realisiert ist (↑CSIM), ausgeführt wird.

Zwischenspeicher Hilfsspeicher: schneller Pufferspeicher zur Verbergung der ↑Latenzzeit für Zugriffe auf den im Vergleich zur ↑Zykluszeit der ↑CPU langsameren ↑Arbeitsspeicher. Die Puffergröße ist ganzzahlige Vielfache der Größe einer ↑Zwischenspeicherzeile.

Zwischenspeicherfehlzugriff Fehlzugriff (*miss*) auf eine im ↑Zwischenspeicher gewähnte Informationseinheit (↑Text, ↑Daten). Für die bei dem Zugriff von der ↑CPU im ↑Abruf- und Ausführungszyklus applizierte ↑Adresse ist kein zugehöriger Eintrag im Zwischenspeicher vermerkt. In dem Fall liest der Zwischenspeicher eigenständig den Inhalt von dem ↑Speicherbereich (im ↑Hauptspeicher) ein, in den diese Adresse fällt. Dieser Bereich hat die Größe einer ↑Zwischenspeicherzeile. Ist der Zwischenspeicher voll, überschreibt der Einlesevorgang eine der Zwischenspeicherzeilen, die in jüngster Zeit am wenigsten referenziert wurde (↑LRU). Wurde ein in dieser Zeile liegendes ↑Speicherwort verändert, muss vor Überschreibung der Zeile ihr aktueller Inhalt zurück in den Hauptspeicher gesichert werden. Der Fehlzugriff ist in funktionaler Hinsicht nicht von einem ↑Prozess wahrnehmbar, wohl aber nichtfunktional: ist das Datum aus dem Hauptspeicher zu lesen, kann in dem Fall (ohne Zurückschreiben) leicht eine um den Faktor 40 höhere ↑Latenz (4 vs. 150 Zyklen) zu Buche schlagen (die mit Zurückschreiben doppelt so hoch sein kann).

Zwischenspeicherzeile Verwaltungseinheit im ↑Zwischenspeicher, deren Größe ganzzahlige Vielfache der Größe von einem ↑Speicherwort ist. Jede dieser Einheit ist die Kopie des Inhalts eines entsprechend großen Bereichs im ↑Hauptspeicher. Der Zugriff auf den Inhalt eines nicht im Zwischenspeicher liegenden Speicherworts bewirkt den blockweisen Transfer der assoziierten Verwaltungseinheit. Wichtiger Aspekt für die ↑Performanz dabei ist die ↑Granularität der Zeile und die jeweiligen Eigentümerschaften der Originalspeicherworte in ihr (↑*false sharing*).

Zykluszeit Zeitspanne vom Start bis zur Vollendung einer Operation oder Operationsfolge.