Betriebssystemtechnik

Adressräume: Trennung, Zugriff, Schutz

VIII. Interprozesskommunikation

Wolfgang Schröder-Preikschat

13. Juni 2019

Gliederung

Einleitung

Konzepte
- Modelle
- Semantiken
- Adressierungsarten
- Verbindungen

Zusammenfassung

Informationsaustausch zwischen Prozessen

Konsequenz der isolierten Adressräume sowie der physikalischen und logischen Verteilung von Funktionen

- die Interaktionen der Prozesse basieren auf Botschaftenaustausch
  - dem Nachrichtenversenden (*message passing*)
  - aber auch Speicherdirektzugriff (*direct memory access*, DMA)
    - also dem Kopieren von Daten direkt heraus aus Prozessadressräumen
    - nicht zu verwechseln mit der Mitbenutzung von Teilen dieser Adressräume
- die Modelle unterscheiden Rollen, Synchronität und Ablaufverhalten

Thema sind system-/maschinennahe (*low-level*) Konzepte, die sich zur Implementierung von Betriebssystemen eignen[1]

- betriebsmittelschonende, effiziente und performante Techniken
  - für Konstruktionen potentiell mehrfädiger Systemprozesse [5]
- Ansätze, die für mikrokernbasierte Betriebssysteme umgesetzt sind

[1] Kommunikationskonzepte UNIX-ähnlicher Betriebssysteme scheiden aus.

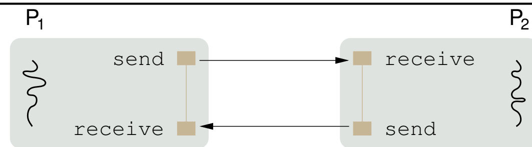
Gliederung

Einleitung

Konzepte
- Modelle
- Semantiken
- Adressierungsarten
- Verbindungen

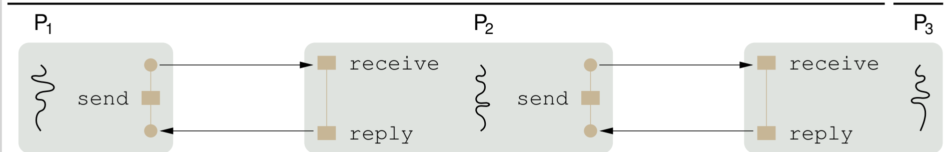
Zusammenfassung

Gleichberechtigung



- Primitiven
  - `send`
    - versendet eine Nachricht an einen Empfangsprozess
    - erwartet den Nachrichtenausgang, -eingang oder -empfang
  - `receive`
    - empfängt eine Nachricht von einem Sendeprozess
    - erwartet den Nachrichteneingang
- Rolle
  - ein Prozess agiert als Sender, der andere als Empfänger, *aber*
  - die Prozesse können ihre Rollen tauschen, *wobei*
  - mit solch einem Rollentausch höchste Vorsicht geboten ist:
    - zum Kommunikationszeitpunkt müssen die Prozessrollen verschieden sein
    - sonst droht die Verklemmung (*deadlock*) ⇝ Verklemmungsvorbeugung

Ungleichberechtigung



- Primitiven
  - `send`
    - versendet einen Auftrag an einen Empfangsprozess und
    - erwartet die Auftragsbeantwortung
  - `receive`
    - erwartet/empfängt einen Auftrag von einem Sendeprozess
  - `reply`
    - versendet eine Antwort an den beauftragenden Sendeprozess
- Rolle
  - ein Prozess agiert als Auftraggeber, der andere als Auftragnehmer
  - in Bezug auf dieses Paar erfolgt kein Rollentausch, *aber*
  - für andere Paarungen können die Prozesse ihre Rollen tauschen, z.B. P_2:
    - (P_1, P_2) P_1 ist Auftraggeber für P_2, der Auftragnehmer von P_1 ist
    - (P_2, P_3) P_2 ist Auftraggeber für P_3, der Auftragnehmer von P_2 ist

Anekdote I

aus der „PEACE-Klinik" [11]

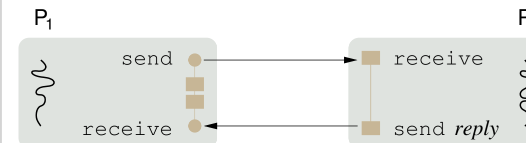
Umsetzung des Modells von Auftraggeber und -nehmer in einem nachrichtenversendenden Minimalkern (*message-passing kernel*)

- verlorengegangenes Aufwecken (*lost wake-up*)
  1. der Auftraggeber hat einen Auftrag gestellt, ist aber noch nicht blockiert
  2. der Auftragnehmer hat den Auftrag erhalten und sofort geantwortet
  3. die Antwort wird verworfen, da der Auftraggeber noch nicht blockiert ist
  4. der Auftraggeber blockiert und erwartet eine nicht eingehende Antwort
- Nichtsequentialität (*concurrency*): Operationen und Operationsfolgen

> *Man hatte nicht erwartet, dass Auftrags- und Anwortnachricht so schnell ihre Runde über das Netzwerk machen konnten, obwohl Sende- und Empfangsknoten jeweils nur einen Prozess bzw. Programmfaden aufwiesen.*

- Auftraggeber *vor* Auftragstellung in den „blockiert-auf"-Zustand setzen
- seinen privaten Nachrichtenpuffer so als Empfangsressource garantieren
- die Antwort wurde nicht verworfen, der Auftraggeber blockierte nicht

Ungleichberechtigung über Gleichberechtigung?



?

Nachbildung von `send-receive-reply` durch `send-receive` jeweils für Auftragstellung und -beantwortung birgt Untiefen

- Blockierung des Auftragnehmers (P_2) bei Auftragsbeantwortung
  - *reply*
    - blockiert nicht, da der Auftraggeber (P_1) die Antwort erwartet
    - P_1 hat die Betriebsmittel (Puffer) für P_2 garantiert
  - `send`
    - lässt P_2 beim Versenden der Antwortnachricht blockieren
      - i wenn P_1 noch nicht empfangsbereit ist *oder*
      - ii Betriebsmittel (Puffer) zum Empfang (noch) nicht garantiert hat
- der Auftragnehmer (P_2) muss dem Auftraggeber (P_1) vertrauen
  - also: ein Systemprozess (P_2) vertraut einem Anwendungsprozess (P_1) !?
  - der Systemprozess (P_2) „benutzt" [10] den Anwendungsprozess (P_1) !?

Anekdote II

Vorsicht mit Argumenten, wonach nichtblockierendes Senden spezielle Primitiven für das Auftraggeber-/-nehmermodell unnötig macht

- die Sequenz (`send`, `receive`) eines Prozesse selbst ist nicht atomar
  - der Kern kennt den semantischen Zusammenhang beider Primitiven nicht

    `send` P_1 versendet die Auftragsnachricht nicht-/blockierend
    `receive` P_2 nimmt diese (blockierend) von P_1 an
    `send` P_2 antwortet P_1 nichtblockierend
    `receive` P_1 empfängt die Antwortnachricht blockierend
  - der Kern müsste P_2 eine Pufferressource garantieren, veranlasst durch P_1
- der Auftraggeber muss die Annahme der Antwortnachricht zusichern
  - i durch Schutz des P_1-seitigen kritischen Abschnitts (`send`, `receive`) *oder*
    - der normalerweise den P_2-seitigen Abschnitt (`receive`, `send`) umfasst
  - ii durch Bindung einer Pufferressource an die zukünftige Antwortnachricht
- eine nichtblockierend sendende Primitive allein löst das Problem nicht

↪ **beachte**: blockierendes Senden *und* Empfangen als Elementaroperation

Prinzipielle Aktionen

- Datentransfer vom Sende- zum Empfangsprozess
  - Botschaftenaustausch über einen gemeinsamen Kommunikationskanal
    - ein realer Übertragungsweg für gespeicherte Informationen
    - über Prozess-, Adressraum-, Prozessor(kern)- oder Rechnergrenzen hinweg
  - dem Speicherdirektzugriff (*direct memory access*, DMA) nicht unähnlich
- Synchronisation von Sende- und Empfangsprozess
  - der Fortschritt des Empfangsprozesses hängt ab vom Sendeprozess
    - die Nachricht ist ein konsumierbares Betriebsmittel
    - erzeugt vom Sende- und verbraucht vom Empfangsprozess
    - verbraucht werden kann nur, was zuvor erzeugt worden ist
    - ↪ Nachrichten werden in Nachrichtenpuffern (Arbeitsspeicher) gehalten
  - der Fortschritt des Sendeprozesses hängt ab vom Empfangsprozess
    - der Nachrichtenpuffer ist ein wiederverwendbares Betriebsmittel
    - entleert vom Empfangs- und zuvor gefüllt vom Sendeprozess
    - gefüllt werden kann nur, wenn noch Pufferplatz zur Verfügung steht
    - ↪ Nachrichtenpuffer sind an Prozesse (statisch/dynamisch) zu binden
  - die Koordinierung geschieht implizit mit der angewandten Primitive

Botschaftenaustauch

Nachrichtenversenden (*message passing*) von einem Sende- zu einem Empfangsprozess über ein gemeinsames Betriebsmittel

- synchron und blockierend
  - der Sender wartet (passiv) im `send` auf das `receive` des Empfängers
  - der Empfänger wartet (passiv) im `receive` auf das `send` des Senders
  - ↪ der Datentransfer erfolgt beim Rendezvous von Sender und Empfänger
    - End-zu-End, ohne modellbedingte Zwischenpufferung der Nachricht
    - direkt zwischen den Adressräumen der beiden involvierten Prozesse
- asynchron und blockierend oder nichtblockierend
  - ggf. ausnahmebedingtes Warten im `send` oder `receive`, etwa bei:
    - i Zwischenpufferung von Nachrichten (*bounded buffer*), zur Abwendung etwaiger Pufferüber- und/oder -unterläufe
    - ii Nichtverfügbarkeit von Nachricht (sendeseitig) oder Nachrichtenplatzhalter bzw. -puffer (empfangsseitig)
    - iii Aufbrauch von Deskriptoren zur sende- oder empfangsseitigen Erfassung der zur übertragenden Nachrichten
  - explizite Entkopplung (implizit gekoppelter) kommunizierender Prozesse

Blockierende vs. nichtblockierende Kommunikation

- die Prozessblockade synchronisiert auf die Betriebsmittelbereitstellung

Sender
- benötigt wiederverwendbare Betriebsmittel

  synchrone IPC ⇒ Nachrichtenplatzhalter ↦ Ziel
  asynchrone IPC ⇒ Zwischenpuffer
  ⇒ Nachrichten(platzhalter)deskriptor

Empfänger
- benötigt konsumier- *und* wiederverwendbare Betriebsmittel

  synchrone IPC ⇒ Nachricht ↦ Quelle
  asynchrone IPC ⇒ Zwischenpuffer
  ⇒ Nachrichten(platzhalter)deskriptor

- asynchrone IPC bedeutet nicht nichtblockierende Kommunikation !!!
  - es meint, Sende- und Empfangsprozess logisch voneinander zu entkoppeln
- „echt nichtblockierend" bedeutet, `send`/`receive` scheitern zu lassen

Aktive Nachrichten

Mechanismus zur asynchronen, nichtblockierenden Kommunikation in (massiv-) parallelen Rechensystemen ⇝ [12]

- der Nachrichtenkopf enthält die Adresse der Empfangsroutine
  - diese wird direkt bei Nachrichteneingang aufgerufen, mit der Aufgabe:
    - i die restliche Nachricht direkt aus dem Netzwerk zu extrahieren[2] *und*
    - ii die so empfangenen Daten in die laufende Berechnung zu integrieren[2]
  - die Behandlungsroutine läuft verdrängungsfrei durch (*run to completion*)
- das Modell ist dem einer Unterbrechungsbehandlung nicht unähnlich
  - nur dass der Sendeprozess den effektiven Unterbrechungsvektor liefert
  - über diesen Vektor wird bei Eingang des Nachrichtenkopfes verzweigt
    - Aufgabe der ersten Unterbrechungsbehandlungsstufe (FLIH): Betriebssystem
- die Unterbrechungsbehandlung erfolgt direkt im Anwendungskontext
  - nebenläufig zum Empfangsprozess in dessen (logischen) Adressraum
  - ungepuffert im System, ggf. auch ungepuffert auf Anwendungsebene
- das Betriebssystem macht einen Hochruf (*upcall* [2]) zur Anwendung

[2]Umgekehrt werden Berechnungsdaten extrahiert, bei Bedarf „abgeholt".

Aktive Nachrichten und Programmiermodelle

Ausgangspunkt ist das SPMD (*single program*, *multiple data*, [3]) Modell zum Betrieb des Rechensystems[3]

- dabei führen alle Prozessoren dasselbe Programm aus
  - in Bezug auf dessen Textsegment unterscheiden sich zwei Ansätze:
    - i gemeinsame Benutzung im prozessorglobalen Speicher (*shared memory*)
    - ii Replikation in die prozessorlokalen Speicher (*distributed memory*)
  - für alle Programminkarnationen gilt derselbe (logische) Adressbereich
    - die Behandlungsroutine liegt in allen Inkarnationen an derselben Adresse
    - damit ist die lokale Adresse der Behandlungsroutine global eindeutig
    - der Sendeprozess kennt damit die Adresse der entfernten Behandlungsroutine
- jeder Prozessor operiert auf anderen/eigenen Datensätzen
  - Prozesse haben starke Lokalität, minimieren Interferenzen mit anderen

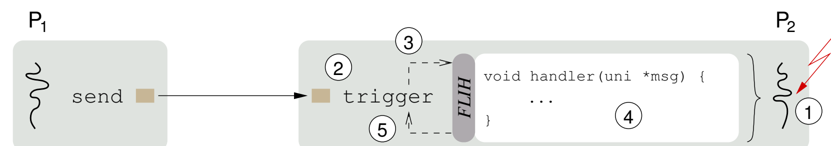
Erweiterungen für eine Loslösung von SPMD nutzen Programmfäden als Residenten, die nicht der Einplanung (*scheduling*) unterliegen

- „reaktive Objekte" [11, S. 192] im Kontext der Anwendung

[3]Untergruppe des MIMD (*multiple Instruction*, *multiple data*, [4]) Prinzips.

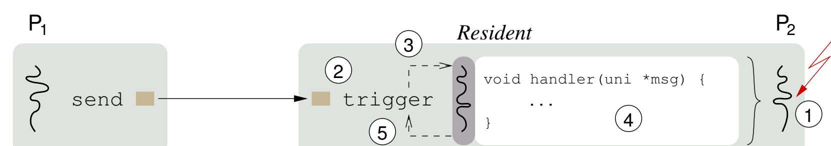
Aktive Nachrichten und Residenten

- logischer Verlauf der Verarbeitung einer aktiven Nachricht:



1. Unterbrechung der laufenden Berechnung ⇝ Nachrichten(kopf)ankunft
2. Auslösung der ersten Unterbrechungsbehandlungsstufe ⇝ Sicherung
3. Aufruf der Nachrichtenbehandlungsroutine ⇝ Kontexterzeugung
4. Behandlung der aktiven Nachricht ⇝ Extraktion/Integration
5. Beendigung der Behandlungsmaßnahmen ⇝ Kontextzerstörung

- Vergleich zu Residenten: 3. und 5. ⇝ Aktivierung/Deaktivierung



Varianten I

vgl. [8, S. 37]

- *no-wait send*
  - der Sendeprozess wartet, bis die Nachricht zum Absenden durch das Transportsystem zusammengestellt worden ist
  - Zusammenstellung bedeutet, die Daten einer Nachricht zu erfassen:
    - i entweder durch Kopieren in einen Nachrichtenplatzhalter (Puffer) oder
    - ii durch Aufbau einer dynamischen Datenstruktur als Nachrichtendeskriptor
- *synchronization send*
  - der Sendeprozess wartet, bis die Nachricht von dem Empfangsprozess angenommen worden ist (End-zu-End)
  - beide Prozesse synchronisieren für den Nachrichtentransfer [7, S. 669]
- *remote-invocation send*
  - der Sendeprozess wartet, bis die Nachricht von dem Empfangsprozess angenommen, verarbeitet und beantwortet worden ist (End-zu-End)
  - ein prozessübergreifender Prozeduraufruf [6, S. 935] bzw. [9]

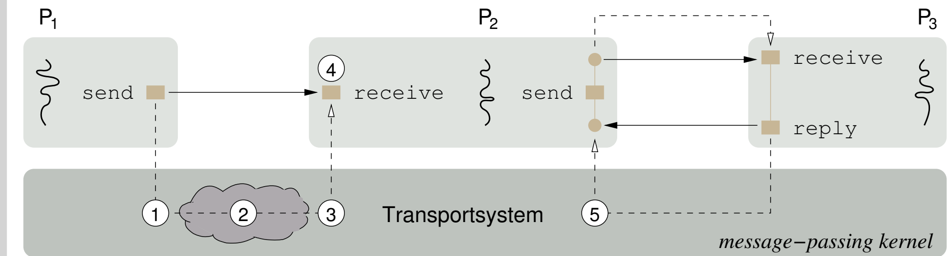
Varianten II

- *non-blocking send* ≡ *no-wait send*
- *explicit-blocking send* ≡ *synchronization send*
- *request/reply* ≡ *remote-invocation send*

- *blocking send*
  - der Sendeprozess wartet, bis die Nachricht sendeseitig freigestellt wurde
    - d.h., in das Transportsystem eingespeist worden ist bzw.
    - den Prozessadressraum oder Rechner verlassen hat
- *reliable-blocking send*
  - der Sendeprozess wartet, bis die Nachricht empfangsseitig postiert wurde
    - d.h. empfangsseitig in einem Nachrichtenpuffer bereitgestellt und
    - dort dem Empfangsprozess zum Empfang zugeordnet worden ist

Varianten im Vergleich

- die Unterschiede schlagen sich in der Sendesemantik nieder:



1. *non-blocking send* (*no-wait send*) – Nachricht ist evtl. noch da
2. *blocking send* – Nachricht ist „auf dem Weg“ (Wolke: „im Netzwerk“)
3. *reliable-blocking send* – Nachricht ist drüben
4. *explicit-blocking send* (*synchronization send*) – Nachricht angenommen
5. *remote-invocation send* (*request/reply*) – Nachricht wurde verarbeitet

- End-zu-End-Semantik auf Prozessebene bieten nur 4. und 5.

Kommunikationsendpunkt

- der Bezeichner (*identifier*) für den Bestimmungsort einer Nachricht
  - festgelegt durch ein bestimmtes, problemspezifisches Bezugssystem
    - das den kommunizierenden Prozessen einer „Domäne“ gemeinsam ist
  - sein Wert muss – für eine gewisse Zeitdauer – systemweit eindeutig sein
- adressiert werden Entitäten verschiedener Abstraktionsebenen:

Prozesskontrollblock	▪ direkte Adresse des Sende-/Empfangsprozesses ▪ Deskriptor einer Prozessinkarnation
Prozessanschluss	▪ indirekte Adresse des Sende-/Empfangsprozesses ▪ Zugang (*port*, [1]) zu einer Prozessinkarnation
Postkasten	▪ Adresse eines Behältnisses für Nachrichten ▪ Aus- oder Eingang (*mailbox*)
Prozedur	▪ Adresse der Verarbeitungsroutine der Nachricht ▪ Nachrichtenextraktion/-integration [12, S. 256]

- bis auf den Postkasten stehen hinter diesen Entitäten – am Endpunkt – immer Aktivitätsträger, d.h., Nachrichten verarbeitende Einheiten

Eindeutigkeit und Lebensdauer

Mehrdeutigkeit des Bezeichners eines Kommunikationsendpunkts ist konstruktiv auszuschließen oder unwahrscheinlich zu machen

- Optionen in vernetzten/verteilten Systemen:

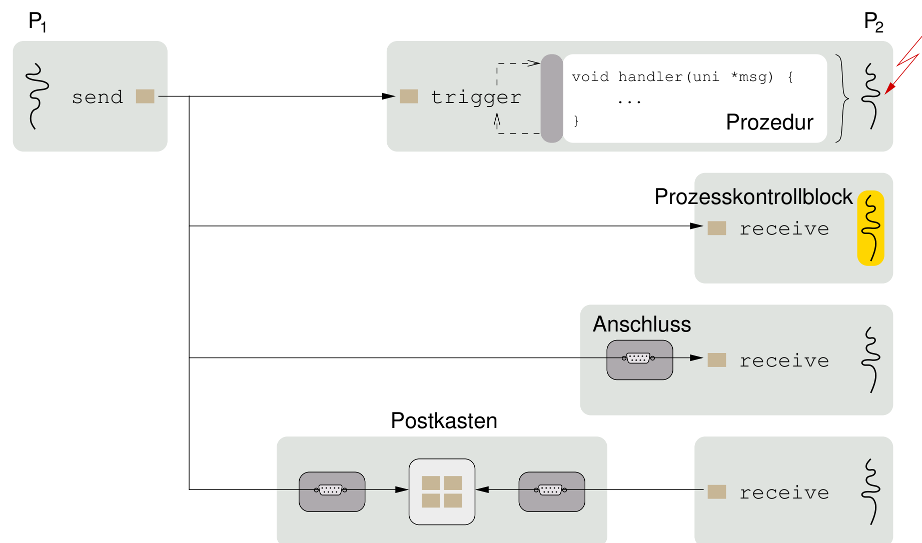
Zufallszahl	▪ hängt ab von der Güte des Zufallszahlengenerators
Zeitstempel	▪ setzt eine synchronisierte Uhr voraus
Knotennummer	▪ MAC-Adresse

- aber bereits lokal ist die eindeutige Wertzuordnung erforderlich

Unikat	▪ vergebene Objektidentifikation (einer Generation)
Generationszähler	▪ unterscheidet Unikate
reale Adresse	▪ Objektidentifikation im realen Adressraum

- der Eindeutigkeitsgrad hängt stark vom gewählten Wertebereich ab
  - der wiederum durch den gegebenen Anwendungsfall bestimmt ist
  - entscheidend sind Dynamik und Größe des Systems von Prozessen
- der Bezeichner repräsentiert eine strukturierte, numerische Adresse
  - mit globalen (Ort) und lokalen (Unikat, Generationszähler) Adressanteilen

Kommunikationsendpunkte im Vergleich

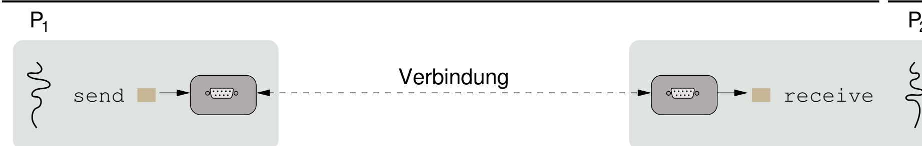


Kommunikationsstart- und -endpunkte

Kommunikation zwischen Prozessen muss nicht ungerichtet verlaufen, sondern kann auch „entlang einer Strecke" gerichtet erfolgen

- gerichtete Kommunikation kennt einen Start- *und* Endpunkt
  - Startpunkt
    - Anschluss beim Sendeprozess
    - Ausgang, über den der Nachrichtenversand erfolgt
  - Endpunkt
    - Anschluss beim Empfangsprozess
    - Eingang, über den die Nachrichtenannahme erfolgt
- zwischen beiden Punkten besteht eine feste Verbindung auf Zeit
  - die über eine Verbindung laufende Kommunikation kennt drei Phasen:
    1. Verbindungsaufbau vom Start- zum Endpunkt
    2. Nachrichtenübermittlung zwischen Start- und Endpunkt
    3. Verbindungsabbau am Start- oder Endpunkt
  - desweiteren kann sie darüber uni- oder bidirektional verlaufen
- Verbindungen machen Beziehungen zwischen den Prozessen explizit
  - sie drücken Datenabhängigkeiten aus, reflektieren Gütemerkmale
  - sie unterstützen die Verklemmungsvorbeugung oder -vermeidung

Verbindungsorientierte Kommunikation



- die Anschlussbezeichner haben in dem Modell lokale Signifikanz:
  - Startpunkt: lokale Adresse im Sendeprozess
  - Endpunkt: lokale Adresse im Empfangsprozess
- sie sind nur eindeutig im Kontext des jeweiligen Prozesses:
  - Ortstransparenz: den Bezeichnern fehlt Ortsinformation
  - Migrationstransparenz: die Prozesse sind unbemerkt verschiebbar
- beim Verbindungsaufbau ist die Örtlichkeit der Prozesse aufzulösen:
  - Namensdienst
    - $name \mapsto (site, port)$
    - Empfangsprozess: $create(name_r, site_r, port_r)$
    - Sendeprozess: $connect(port_s, resolve(name_r))$

↪ der Namensdienst ist auch für verbindungslose Kommunikation gut

Gliederung

Einleitung

Konzepte
- Modelle
- Semantiken
- Adressierungsarten
- Verbindungen

Zusammenfassung

Resümee

- kooperierende Prozesse müssen Informationen austauschen können
  - Botschaftenaustausch als Konsequenz isolierter Adressräume
- sie gehen verschiedene Rollenspiele ein und wechseln diese ggf.
  - gleichberechtigte Kommunikation ⇝ Sender und Empfänger
  - ungleichberechtigte Kommunikation ⇝ Auftraggeber und Auftragnehmer
    - Realisierung durch gleichberechtigte Kommunikation ist problematisch
- prinzipielle Aktionen bei IPC sind Datentransfer und Synchronisation
  - die Koordinierung geschieht implizit mit der angewandten Primitive
    - synchrone/asynchrone und blockierende/nichtblockierende IPC
  - Unterschiede ergeben sich in der Sendesemantik der jeweiligen Primitive
    - *non-blocking*, *blocking*, *reliable blocking*, *explicit blocking*, *remote invocation*
  - aktive Nachrichten als Kommunikationsmechanismus paralleler Systeme
- Bezeichner des Bestimmungsort einer Nachricht als „Objektadressen“
  - Prozedur, Prozesskontrollblock, Prozessanschluss, Postkasten
- verbindungsorientierte Kommunikation zwischen Prozessanschlüssen

Literaturverzeichnis I

[1] Balzer, R. M.:
PORTS—A Method for Dynamic Interprogram Communication and Job Control.
In: *Proceedings of the Spring Joint Computer Conference (AFIPS'71)* Bd. 38
American Federation of Information Processing Societies, 1971, S. 485–489

[2] Clark, D. D.:
The Structuring of Systems Using Upcalls.
In: Cheriton, D. R. (Hrsg.) ; Birrell, A. (Hrsg.): *Proceedings of the Tenth ACM Symposium on Operating System Principles (SOSP '85)*, ACM, 1985. –
ISBN 0–89791–174–1, S. 171–180

[3] Darema-Rogers, F. ; George, D. A. ; Norton, V. A. ; Pfister, G. F.:
A VM Parallel Environment / IBM.
1985 (RC 11225). –
IBM Research Report

[4] Flynn, M. J.:
Very High-Speed Computing Systems.
In: *Proceedings of the IEEE* 54 (1966), Dez., Nr. 12, S. 1901–1909

Literaturverzeichnis II

[5] Gentleman, W. M.:
Message Passing Between Sequential Processes: The Reply Primitive and the Administrator Concept.
In: *Software—Practice and Experience* 11 (1981), Nr. 5, S. 435–466

[6] Hansen, P. B.:
Distributed Processes: A Concurrent Programming Concept.
In: *Communications of the ACM* 21 (1978), Nov., Nr. 11, S. 934–941

[7] Hoare, C. A. R.:
Communicating Sequential Processes.
In: *Communications of the ACM* 21 (1978), Aug., Nr. 8, S. 666–677

[8] Liskov, B. J. H.:
Primitives for Distributed Computing.
In: *Proceedings of the Seventh ACM Symposium on Operating System Principles (SOSP 1979)*, ACM, 1979. –
ISBN 0–89791–009–5, S. 33–42

[9] Nelson, B. J.:
Remote Procedure Call.
Pittsburg, PA, USA, Department of Comupter Science, Carnegie-Mellon University, Diss., Mai 1981

Literaturverzeichnis III

[10] Parnas, D. L.:
On a 'Buzzword': Hierarchical Structure.
In: Rosenfeld, J. L. (Hrsg.): *Information Processing 74, Proceedings of the IFIP Congress 74.*
New York, NY, USA : North-Holland Publishing Company, 1974. –
ISBN 0–7204–2803–3, S. 336–339

[11] Schröder-Preikschat, W. :
The Logical Design of Parallel Operating Systems.
Upper Saddle River, NJ, USA : Prentice Hall International, 1994. –
ISBN 0–13–183369–3

[12] von Eicken, T. ; Culler, D. E. ; Goldstein, S. C. ; Schauser, K. E.:
Active Messages: a Mechanism for Integrated Communication and Computation.
In: Gottlieb, A. (Hrsg.): *Proceedings of the 19th International Symposium on Computer Architecture (ISCA '92)*, ACM, 1992. –
ISBN 0–89791–509–7, S. 256–266