Instruction Level Parallelism (ILP)

Von intel und HP entwickelt.
Bei einem **„out of order processor“** ist der Abschluss von Befehlen in einer Reihenfolge, die nicht der programmierten entspricht. Es wird eine **parallel execution** Unterstützung erreicht, keine Register-Abhängigkeiten um parallelität auszuführen.

Was steckt hinter „IA-64 parallel execution semantic“?
Befehls-Gruppen; code wird in Gruppen, die eigenständig ausgeführt werden können eingeteilt, so kann parallelität sichergestellt werden.
Als Kriterium werden zwei Semikolons verwendet, die im Programmcode an entsprechender Stelle hinten angehängt werden.

Zusammengesetzte Sprungbedingungen aus AND und OR

„multiway branch“ erlaubt mehrere Normale Sprünge in 1 **Single instruction group**.

Speculative instructions ist im Zusammenhang mit Sprung eine Vorhersage, damit keine Wartezeit beim Sprungbefehl entsteht. D.h. wenn Befehle ausgeführt werden, ohne dass gewusst wird, ob sie jemals gebraucht werden.
Lösung im IA-64: Prediction erlaubt compiler, Instructionen **von multiple conditional paths** zur selben Zeit auszuführen, d.h. beide Bedingungen parallel auszuführen.

Exceptions: terminate; all exceptions must be delivered ; trotzdem das Programm weiterläuft

Umordnung von **load-instruktionen**
Data speculation kann prüfen, ob Pointer mehrere mal referenziert sind, wenn ja, dann schützt diese der Compiler vor der Umordnung der Instruktionen.

Dateiverwaltung

Einführung

Dateien stellen aus einer logischen Sicht gesehen eine sequentielle Speicherung von Bytes auf einem Massenspeicher dar. Der Zugriff auf eine Datei erfolgt über ihren Namen und einen Dateizeiger, der bei Lesevorgängen jeweils um die Anzahl transferierter Bytes nach hinten verschoben wird. Damit kann eine Datei in unterschiedlich grossen Blöcken vom Anfang bis an das Ende ausgelesen werden (*sequential access*). Über eine spezielle Funktion kann der Dateizeiger aber auch an eine beliebige gewünschte Stelle innerhalb der Datei verschoben werden, womit ein wahlfreier (*random access*) Zugriff möglich wird. Diese Regeln gelten nicht nur für Lese- sondern auch für Schreibvorgänge.

In einfacheren Fällen werden Dateien von Applikationen zur Ablage von strukturierten Daten verwendet, wie dies z.B. eine Kundenkartei darstellt. Dabei wird für jeden Kunden eine durch den Programmentwickler definierte gleichbleibende Strukur der abzulegenden Daten verwendet (sog. Datensatz).

Definition[1] **Datensatz** (Synonyme: Satz, logischer Satz, *record*):
Als Datensatz bezeichnet man die Zusammenfassung logisch zusammengehörender Daten, die dasselbe Objekt oder denselben Sachverhalt betreffen, zu einer logischen Einheit.
Beispiel: Adressdaten eines Kunden.

Definition **Feld** (*element*):
Ein Feld ist die benannte Komponente der Datenstruktur eines Datensatzes.
Felder können selbst unstrukturiert, strukturiert (d.h. weitere Komponenten enthalten) und wiederholt in einem Datensatz auftreten.
Beispiel: Name des Kunden.

Definition **Datensatztyp** (*record type*):
Den formalen Aufbau eines Datensatze aus Feldern bezeichnet man als Datensatztyp. Er umfasst den Datensatznamen, Feldnamen und Feldtypen.
Beispiele: Kundendaten, Auftragsdaten, Artikeldaten.

Definition **Datei** (*file*):
Unter einer Datei versteht man die Zusammenfassung von Datensätzen desselben Datensatztyps.
Beispiel: gesamter Kundendatenstamm einer Firma.

Definition **Schlüssel** (*key*):
Sei D eine Datei des Datensatztyps R. Sei A ein Feld von R.
A ist ein Schlüssel in D, wenn alle Datensätze von D unterschiedliche A-Werte haben. A muss zudem minimal sein, d.h es existiert keine Teilmenge von A, die vorgenannte Bedingungen erfüllt.
Beispiele: Kundennummer, Auftragsnummer, Artikelnummer.

Definition **Primärschlüssel**:
Ein Schlüssel ist dann Primärschlüssel, wenn er Schlüssel in eine Datei ist und von vornherein als solcher festgelegt wurde.
Beispiele: Kundennummer, Auftragsnummer, Artikelnummer.

Definition **Sekundärschlüssel**:
Ein Sekundärschlüssel ist ein beliebiges Feld (oder eine Kombination von Feldern), mit dessen Hilfe man auf die Datensätze einer Datei zugreift.

Problem beim sortieren: Sortierreihenfolge bei gleichem Namen ist undefiniert

Primärschlüssel muss immer verschieden sein (Bsp. PrimaryKey Kundennummer, die drei Kunden „Müller“ haben unterschiedliche Kundennummern)

Datensätze abbilden anhand byte offset, wenn fixe Datensatzgrösse kann eine relative Datensatznummer in einen Byteversatz umberechnet werden, Beginnend bei 0 lückenlos auffüllen.

Einträge (Kunden) suchen über Primärschlüssel (Kundennummer) oder Sekundärschlüssen (Name,Vorname)

Datensatz=Objekt in OOP

Diskstruktur und Dateisystem

-> Bd. II, Kap. 1.6
DSKprobe

Bootblock in Diskette:
Grösse des Boot-Bereichs: 512 Byte
Grösse der FAT: 4608 Byte *=9*512*
Anzahl der FAT's: 2
Maximalanzahl von Einträgen im RootDirectory: 224
Anzahl verfügbare Cluster für Dateien 2847
= Gesamtanzahl – 1 (Bootblock) – 18 (2FAT's) – 224/16 (Platz Root Dir)
Anzahl Blöcke pro Cluster: 1

Allgemeine Formel für Tabelle auf S. 127:

$$AnzahlVerfügbareCluster = \frac{0x13}{0x0d} - \frac{0x10 \cdot 0x16}{0x0d} - \frac{0x11}{16} - \frac{0x0e}{0x0d} - \frac{0x1e}{0x0d}$$

Dateien löschen und entlöschen:
Wenn Datei durch Ändern des Verzeichniseintrags „gelöscht" ist, können dessen Cluster noch nicht verwendet werden, da sie in der FAT immer noch als belegt markiert sind.
Erst wenn die Cluster als frei markiert sind, können diese wieder überschrieben werden.

Anzahl Zylinder / Sektoren: Media De-Skriptor auf S. 127 suchen und mit Wert in Tabelle 144 S. 132 vergleichen.

Diskstruktur und Partitionierung

Beispiel FAT-16
Eckwerte: 512 Byte pro Block
Max. 2^6 Blöcke pro cluster
Max. 2^16 Cluster pro logischem Laufwerk

Max Anz. Byte pro logischem Laufwerk: 512*2^6 * 2^16
Anz. Byte exkl. Verzeichniseintrag werden auf log. LW mit 1.6GB Kapazität benötigt um 1Byte Datei zu speichern? = 1 Cluster = 512 Byte * 2^6
Anz log LW auf 1.6 GB-Platte um Clustergrösse von 8kB:
8192 * 2^16 = 0.5GB => 4 Partitionen

Die Dateien einer Applikation können in drei Gruppen unterteilt werden (vereinfachte Annahme):

Dateiart	Anzahl	Grösse in Byte (pro Datei)
Treiberkonfiguration	170	700
Menue-Subsysteme	14	165'000
Applikationen	2	820'000

Die Speicherung erfolgt auf einem logischen Laufwerk mit 1.6 GB Kapazität (FAT-16).
Bestimmen Sie auf Byte genau:

a) Wieviel Platz (exkl. Verzeichniseinträge) wird auf dem Laufwerk belegt?

b) Wieviel Platz wird von dem Anwendungssystem effektiv benötigt?

c) Wieviel Platz geht infolge der Blockspeicherung bzw. der internen Fragmentierung "verloren"?

a) 170 · 32'768 Byte = 5570560 Byte

$\frac{165'000}{32'768} = 5{,}03 \Rightarrow 6$ Cluster $\Rightarrow$ 14 · 6 · 32'768 Byte = 2752512 Byte

$\frac{820'000}{32'768} = 25{,}02 \Rightarrow 26$ Cluster $\Rightarrow$ 26 · 2 · 32'768 Byte = 1703936 Byte

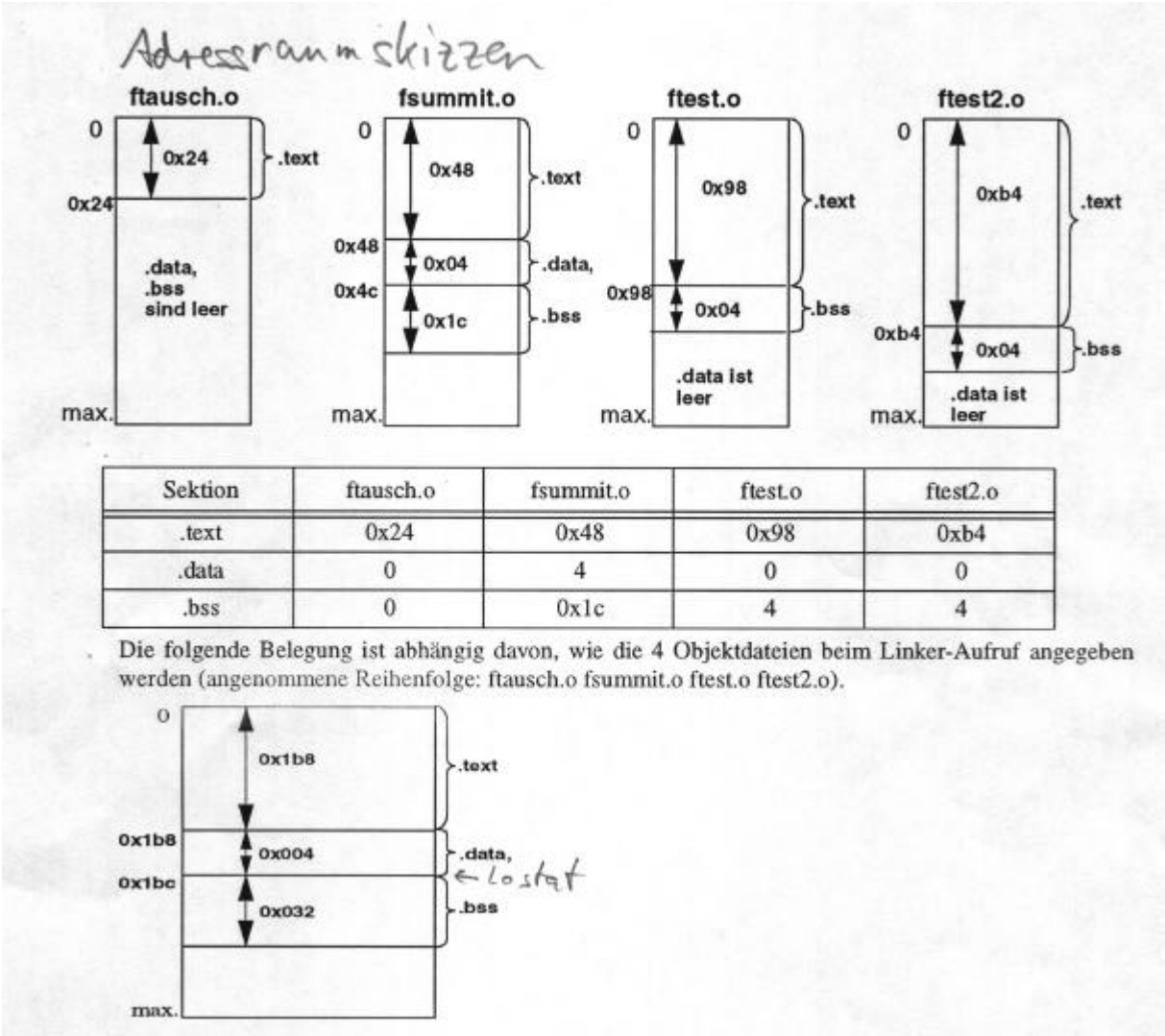
Total = 5570560 Byte + 2752512 Byte + 1703936 Byte = 10027008 Byte

b) 170 · 700 B + 14 · 165'000 B + 2 · 820'000 B = 4'069'000 Byte

c) 10'027'008 Byte − 4'069'000 Byte = 5'958'008 Byte

Programmübersetzung

-> Bd. II, Kap. 1.7



Sektion	ftausch.o	fsummit.o	ftest.o	ftest2.o
.text	0x24	0x48	0x98	0xb4
.data	0	4	0	0
.bss	0	0x1c	4	4

Die folgende Belegung ist abhängig davon, wie die 4 Objektdateien beim Linker-Aufruf angegeben werden (angenommene Reihenfolge: ftausch.o fsummit.o ftest.o ftest2.o).

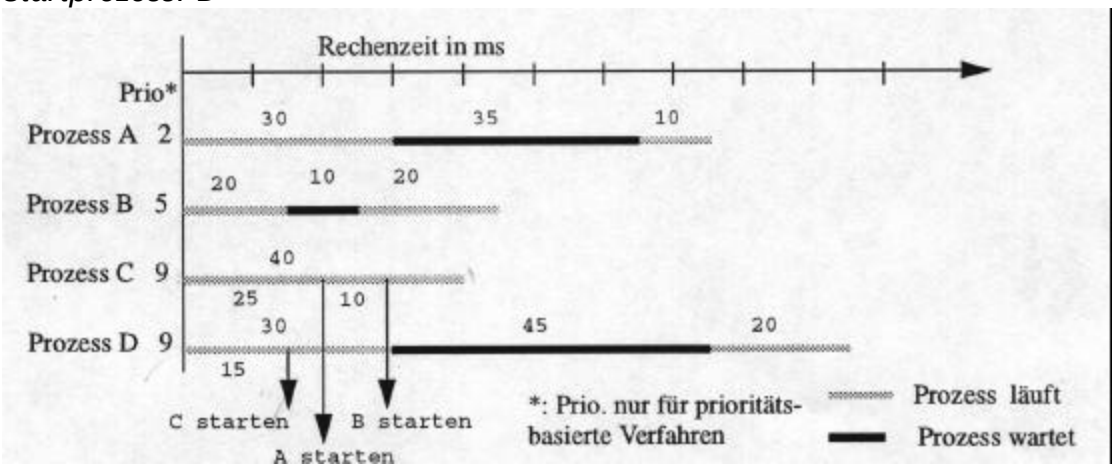
Parallele Prozesse unter UNIX

Fork() = Prozessverdopplung

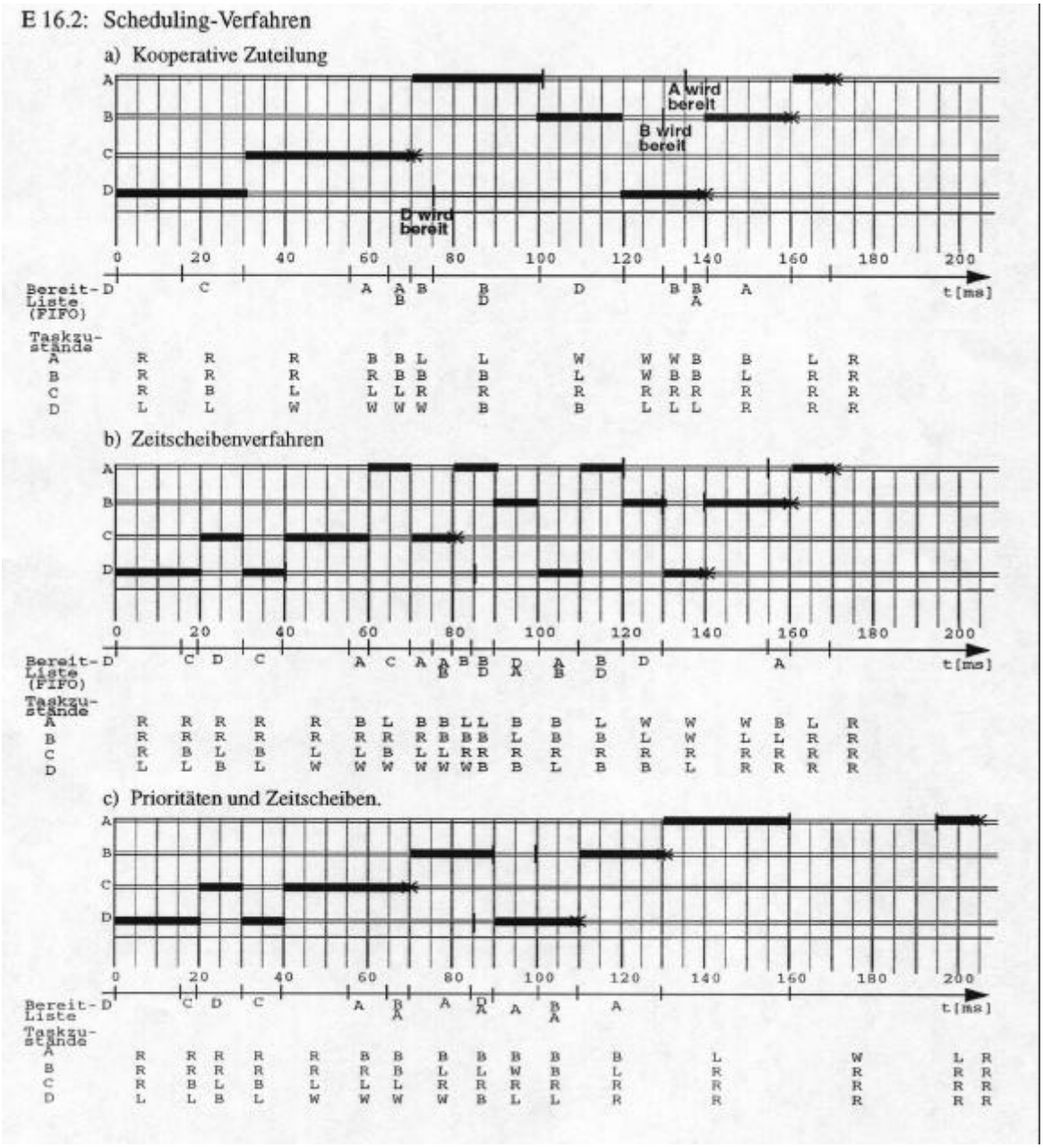
Zombie: Band 1 S. 179
Ist ein Zombie-Prozess, falls Kindprozess terminiert bevor Elternprozess wartet.
Ist kein Zombie, wenn Elternprozess auf Kindprozess wartet.

Prozessverwaltung

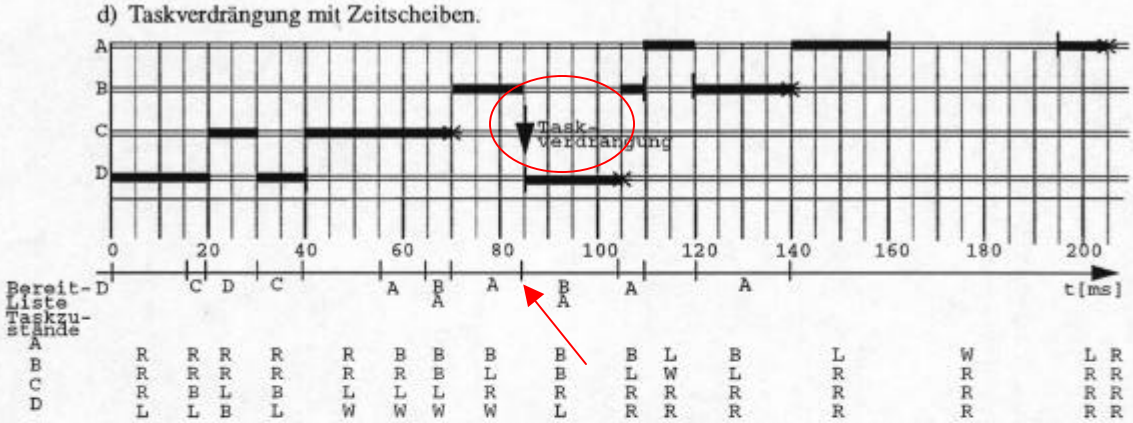
Startprozess: D



Buch Band 1 Seite 182

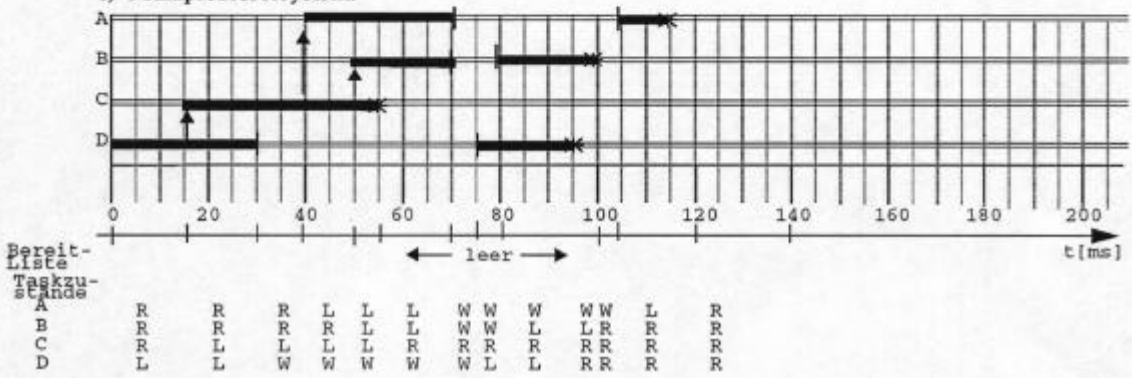


d) Taskverdrängung mit Zeitscheiben.

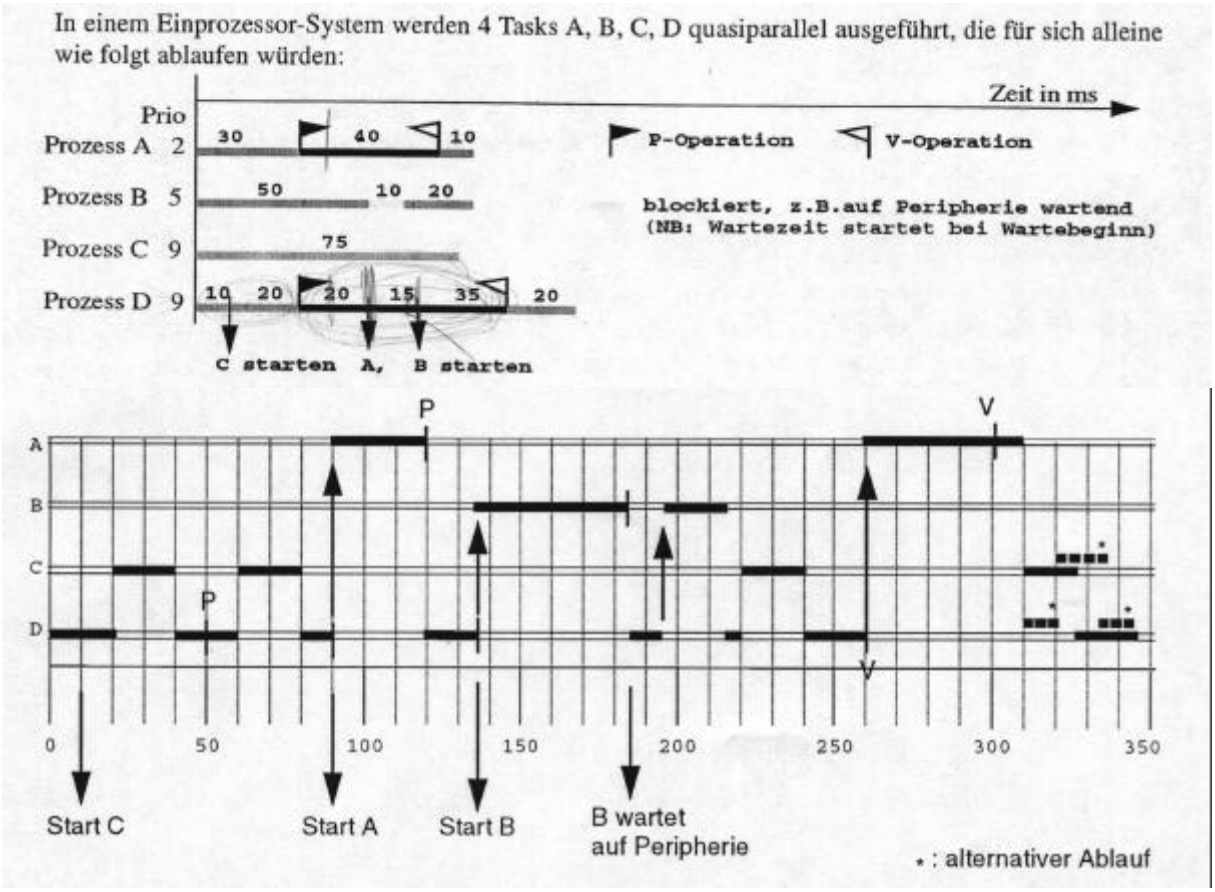


Hinweis: bei den prioritätsorientierten Verfahren wird die Bereit-Liste *primär* nach Prioritäten und *sekundär* nach FCFS (First Come - First Served) organisiert.

e) Multiprozessorsystem:



Semaphoren



Verbesserungsmöglichkeiten zum Beispiel
Verwendung eines Semaphors mit Prioritätsvererbung (priority inheritance); falls kein derartiger Semaphortyp verfügbar: vor Aufruf von P in D die Priorität von D auf die von A anheben. Nach Aufruf von V die Priorität wieder auf den ursprünglichen Wert absenken (priority ceiling Methode).
Ist auch diese Möglichkeit mangels eines entsprechenden Systemaufrufs nicht verfügbar, so kann Prozess D den Scheduler blockieren mit lock() / unlock() für die Dauer des kritischen Bereiches.

Problem der gemeinsamen Ressourcen erkennen:
Kritischer Bereich mit p() und v() schützen

Strategie	preemptive	connected	round-robin	cooperative	priority based	inactive	scheduling
E16.2 a)				X			X
E16.2 b)			X	(X)			X
E16.2 c)			(X)	(X)	X		X
E16.2 d)	X		(X)	(X)	X		X

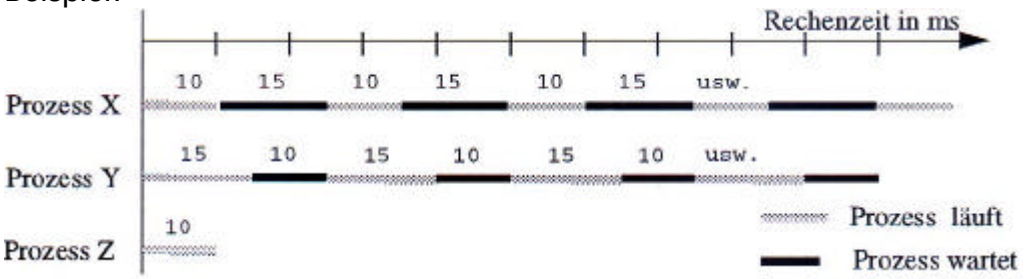
NB: (X) heisst in Praxis nicht gebräuchlich, da nur sekundär gebrauchte Strategie.

a) Kooperative Zuteilung
b) Zeitscheibenverfahren
c) Prioritätsabhängige Zuteilung
d) Priorisierte Zuteilung mit Verdrängung

Dynamische Prioritätsanhebungen, damit Tasks tiefer Priorität nicht verhungern, d.h. irgendwann auch einmal den Prozessor erhalten.
Dabei gilt die *prioritätsabhängige Zuteilung* mit folgender Erweiterung:

> *Die Priorität einer Task wird für eine neue Zeitscheibe (Zeitscheibengrösse 10 ms) dann um 1 erhöht, wenn die Task in der alten Zeitscheibe immer noch „bereit bleibt". Bekommt eine Task den prozessor, so wird ihre Priorität beim nächstfolgenden Zeitscheibenende auf den Anfangswert zurückgesetzt.*

Beispiel:



Lösung