

# Systemprogrammierung

Grundlage von Betriebssystemen

## Teil C – IX.2 Prozessverwaltung: Einplanungsverfahren

Wolfgang Schröder-Preikschat

29. Oktober 2015



## Agenda

Einführung

Einordnung  
Klassifikation

Verfahrensweisen  
Kooperativ  
Verdrängend  
Probabilistisch  
Mehrstufig

Zusammenfassung



## Gliederung

Einführung

Einordnung  
Klassifikation

Verfahrensweisen  
Kooperativ  
Verdrängend  
Probabilistisch  
Mehrstufig

Zusammenfassung



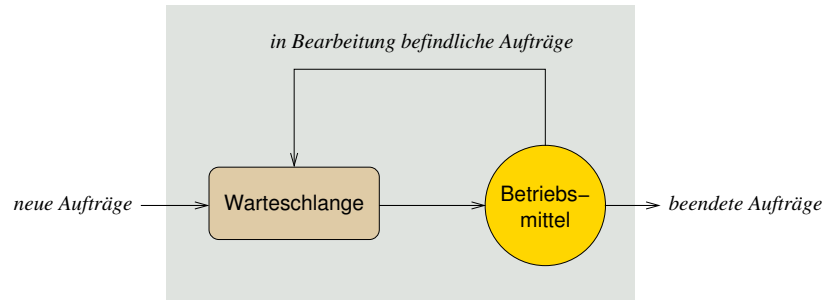
## Lehrstoff

- gängige Klassen der **Ein-/Umplanung** von Prozessen kennenlernen und in ihrer Bedeutung einschätzen können
  - jedes Verfahren einer Klasse hat bestimmte **Gütemerkmale** im Fokus
  - bei mehreren Merkmalen müsste ein **Kompromiss** gefunden werden
  - scheidet Konfliktlösung aus, ist eine geeignete **Priorisierung** vorzunehmen
- die Verfahren auf **nicht-funktionale Eigenschaften** untersuchen, so Gemeinsamkeiten und Unterschiede erfassen
  - Gerechtigkeit, minimale Antwort- oder Durchlaufzeit
  - maximaler Durchsatz, maximale Auslastung
  - Termineinhaltung, Dringlichkeiten genügend, Vorhersagbarkeit
- erkennen, dass es die „eierlegende Wollmilchsau“ auch in einer virtuellen Welt nicht geben kann. . .

*Kein einziges Verfahren zur Ein-/Umplanung von Prozessen hat nur Vorteile, befriedigt alle Bedürfnisse, genügt allen Ansprüchen.*



## ■ Verwaltung von (betriebsmittelgebundenen) Warteschlangen



Ein einzelner Einplanungsalgorithmus ist charakterisiert durch die Reihenfolge von Prozessen in der Warteschlange und die Bedingungen, unter denen die Prozesse in die Warteschlange eingereiht werden. [7]



- die Charakterisierung von **Einplanungsalgorithmen** macht glauben, Betriebssysteme fokussiert „mathematisch“ studieren zu müssen:
  - R. W. Conway, L. W. Maxwell, L. W. Millner. *Theory of Scheduling*.
  - E. G. Coffman, P. J. Denning. *Operating System Theory*.
  - L. Kleinrock. *Queuing Systems, Volume I: Theory*.
- praktische Umsetzung offenbart jedoch einen **Querschnittsbelang** (*cross-cutting concern*), der sich kaum modularisieren lässt
  - spezifische Betriebsmittelmerkmale stehen ggf. Bedürfnissen der Prozesse, die Aufträge zur Betriebsmittelnutzung abgesetzt haben, gegenüber
  - dabei ist die Prozessreihenfolge in Warteschlangen (bereit, blockiert) ein Aspekt, die Auftragsreihenfolge dagegen ein anderer Aspekt
  - **Interferenz**<sup>1</sup> bei der Durchsetzung der Strategien kann die Folge sein
- Einplanungsverfahren stehen und fallen mit den Vorgaben, die für die jeweilige **Zieldomäne** zu treffen sind
  - die „Eier-legende Wollmilchsau“ kann es nicht geben
  - Kompromisslösungen sind geläufig — aber nicht in allen Fällen tragfähig

<sup>1</sup>lat. *inter* zwischen und *ferire* von altfrz. *s'entreferir* sich gegenseitig schlagen.



## Gliederung

Einführung

Einordnung  
Klassifikation

Verfahrensweisen  
Kooperativ  
Verdrängend  
Probabilistisch  
Mehrstufig

Zusammenfassung



## Kooperativ vs. Präemptiv

... bereit zur Zusammenarbeit

- **Souverän** ist die Anwendung oder das Betriebssystem verhält sich Entwicklungen gegenüber zuvorkommend, vorsorglich, vorbeugend

**kooperative Planung** (*cooperative scheduling*)

- Ein-/Umplanung voneinander abhängiger Prozesse
- Prozessen wird die CPU nicht zugunsten anderer Prozesse entzogen
- der laufende Prozess gibt die CPU nur mittels Systemaufruf ab
  - die Systemaufrufbehandlung aktiviert (direkt/indirekt) den Scheduler
  - systemaufruffreie Endlosschleifen beeinträchtigen andere Prozesse
- **CPU-Monopolisierung** ist möglich: *run to completion*

**präemptive Planung** (*preemptive scheduling*)

- Ein-/Umplanung voneinander unabhängiger Prozesse
- Prozessen kann die CPU entzogen werden, zugunsten anderer Prozesse
- der laufende Prozess wird **ereignisbedingt** von der CPU **verdrängt**
  - die Ereignisbehandlung aktiviert (direkt/indirekt) den Scheduler
  - Endlosschleifen beeinträchtigen andere Prozesse nicht (bzw. kaum)
- Monopolisierung der CPU ist nicht möglich: **CPU-Schutz**



## Deterministisch vs. Probabilistisch

- alle Abläufe durch ***à priori* Wissen** eindeutig festlegen könnend oder die **Wahrscheinlichkeit** berücksichtigend
  - deterministische Planung** (*deterministic scheduling*)
    - alle Prozesse (CPU-Stoßlängen)<sup>2</sup> und ggf. auch **Termine** sind bekannt
    - die genaue Vorhersage der CPU-Auslastung ist möglich
    - das System stellt die Einhaltung von **Zeitgarantien** sicher
    - die Zeitgarantien gelten unabhängig von der jeweiligen Systemlast
  - probabilistische Planung** (*probabilistic scheduling*)
    - Prozesse (exakte CPU-Stoßlängen) sind unbekannt, ggf. auch Termine
    - die CPU-Auslastung kann lediglich abgeschätzt werden
    - das System kann Zeitgarantien weder geben noch einhalten
    - Zeitgarantien sind durch die Anwendung sicherzustellen
- dabei fällt die Abgrenzung nicht immer so scharf aus: wahrscheinliche Abläufe vorherzusagen, ist sehr nützlich...

<sup>2</sup>Bei (strikten) Echtzeitsystemen mindestens die Stoßlänge des „schlimmsten Falls“ (*worst-case execution time, WCET*).



## Statisch vs. Dynamisch

- Abläufe entkoppelt von oder gekoppelt mit der Programmausführung bestimmen und entsprechend entwickeln
  - vorlaufende Planung** (*off-line scheduling*)
    - vor** Betrieb des Prozess- oder Rechensystems
    - die Berechnungskomplexität verbietet Planung im laufenden Betrieb
      - z.B. die Berechnung, dass alle Zeitvorgaben garantiert eingehalten werden
      - unter Berücksichtigung jeder abfangbaren katastrophalen Situation
    - Ergebnis der Vorberechnung ist ein **vollständiger Ablaufplan**
      - u.a. erstellt per Quelltextanalyse spezieller „Übersetzer“
      - oft zeitgesteuert abgearbeitet als Teil der Prozesseinlastung
    - die Verfahren sind zumeist beschränkt auf **strikte Echtzeitsysteme**
  - mitlaufende Planung** (*on-line scheduling*)
    - während** Betrieb des Prozess- oder Rechensystems
    - Stapelsysteme, interaktive Systeme, verteilte Systeme
    - schwache und feste Echtzeitsysteme
- auch hier ist die Abgrenzung nicht immer so scharf: von vorläufigen Ablaufplänen ausgehen zu können, ist sehr hilfreich...



## Asymmetrisch vs. Symmetrisch (1)

... Standpunktfrage

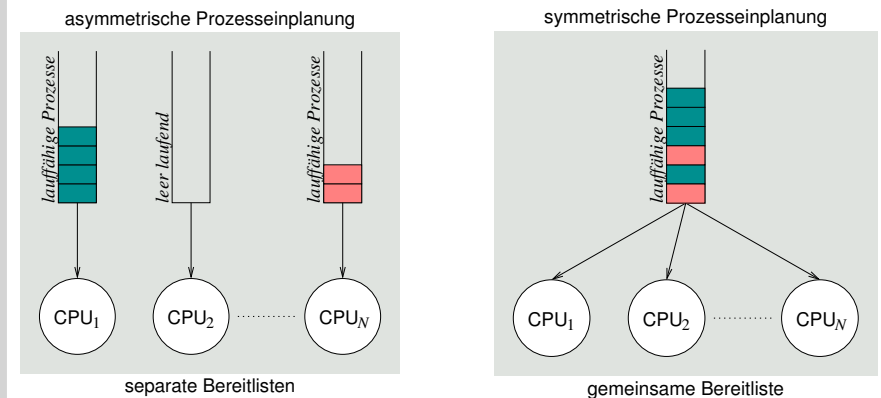
- für **mehrere Prozessoren** Abläufe nach verschiedenen Kriterien oder ihren wechselseitigen Entsprechungen festlegen
  - ungleichmäßige Planung** (*asymmetric scheduling*)
    - je nach den Prozesseigenschaften der **Maschinenprogrammebene**
    - obligatorisch in einem asymmetrischen Multiprozessorsystem
      - Rechnerarchitektur mit **programmierbare Spezialprozessoren**
      - z.B. Grafik- und/oder Kommunikationsprozessoren einerseits
      - ein Feld konventioneller (gleichartiger) Prozessoren andererseits
    - optional in einem symmetrischen Multiprozessorsystem (s.u.)
      - das Betriebssystem hat freie Hand über die Prozessorvergabe
    - Prozesse in funktionaler Hinsicht ungleich verteilen (müssen)

### gleichmäßige Planung (*symmetric scheduling*)

- je nach den Prozesseigenschaften der **Befehlssatzebene**
- identische Prozessoren, alle geeignet zur Programmausführung
- Prozesse möglichst gleich auf die Prozessoren verteilen: **Lastausgleich**



## Asymmetrisch vs. Symmetrisch (2)



- |  |                                 |
|--|---------------------------------|
| ■ lokale Bereitliste                         | ■ globale Bereitliste           |
| ■ ggf. ungleichmäßige Auslastung             | ■ ggf. gleichmäßige Auslastung  |
| ■ <u>ohne</u> gegenseitige Beeinflussung     | ■ gegenseitige Beeinflussung    |
| ■ <u>keine</u> Multiprozessorsynchronisation | ■ Multiprozessorsynchronisation |



## Gliederung

Einführung

Einordnung  
Klassifikation

Verfahrensweisen  
Kooperativ  
Verdrängend  
Probabilistisch  
Mehrstufig

Zusammenfassung



## Klassische Planungs- bzw. Auswahlverfahren

- betrachtet werden grundlegende Ansätze für **Uniprozessorsysteme**, je nach Klassifikationsmerkmal bzw. nichtfunktionaler Eigenschaft:

kooperativ FCFS

gerecht

- wer zuerst kommt, mahlt zuerst...

verdrängend RR, VRR

reihum

- jeder gegen jeden...

probabilistisch SPN (SJF), SRTF, HRRN

priorisierend

- die Kleinen nach vorne...

mehrstufig MLQ, FB (MLFQ)

- Multikulti...

- dabei steht die Fähigkeit zur **Interaktion** mit „externen Prozessen“ (insb. dem Menschen) als Gütemerkmal im Vordergrund
  - d.h., Auswirkungen auf die **Antwortzeit** von Prozessen



## FCFS

*first come, first served*

*Fair, einfach zu implementieren (FIFO), ..., dennoch problematisch.*

- Prozesse werden nach ihrer **Ankunftszeit** (*arrival time*) eingeplant und in der sich daraus ergebenden Reihenfolge auch verarbeitet
  - nicht-verdrängendes Verfahren, setzt kooperative Prozesse voraus
- gerechtes Verfahren auf Kosten einer im Mittel höheren Antwortzeit und niedrigerem E/A-Durchsatz
  - suboptimal bei einem Mix von kurzen und langen CPU-Stößen

Prozesse mit  $\left\{ \begin{array}{l} \text{langen} \\ \text{kurzen} \end{array} \right\}$  CPU-Stößen werden  $\left\{ \begin{array}{l} \text{begünstigt} \\ \text{benachteiligt} \end{array} \right\}$

- Problem: **Konvoieffekt**
  - kurze Prozesse bzw. CPU-Stöße folgen einem langen...



## FCFS: Konvoieffekt

- Durchlaufzeit kurzer Prozesse im Mix mit langen Prozessen:

Prozess	Zeiten					$T_q/T_s$
	Ankunft	$T_s$	Start	Ende	$T_q$	
A	0	1	0	1	1	1.00
B	1	100	1	101	100	1.00
C	2	1	101	102	100	100.00
D	3	100	102	202	199	1.99
Ø					100	26.00

$T_s$  = Bedienzeit,  $T_q$  = Durchlaufzeit

normalisierte Durchlaufzeit ( $T_q/T_s$ )

- ideal für A und B, unproblematisch für D
- schlecht für C
  - sie steht in einem extrem schlechten Verhältnis zur Bedienzeit  $T_s$
  - typischer Effekt im Falle von kurzen Prozessen, die langen folgen



Verdrängendes FCFS, Zeitscheiben, CPU-Schutz.

- Prozesse werden nach ihrer **Ankunftszeit** ein- und in regelmäßigen Zeitabständen (periodisch) umgeplant
  - verdrängendes Verfahren, nutzt **periodische Unterbrechungen**
    - Zeitgeber (*timer*) liefert asynchrone Programmunterbrechungen
  - jeder Prozess erhält eine **Zeitscheibe** (*time slice*) zugeteilt
    - obere Schranke für die CPU-Stoßlänge eines laufenden Prozesses
- Verringerung der bei FCFS auftretenden Benachteiligung von Prozessen mit kurzen CPU-Stößen
  - die **Zeitscheibenlänge** bestimmt die Effektivität des Verfahrens
    - zu lang, Degenierung zu FCFS; zu kurz, sehr hoher Mehraufwand
  - Faustregel: etwas länger als die Dauer eines „typischen CPU-Stoßes“
- Problem: **Konvoieffekt**
  - Prozesse kürzer als die Zeitscheibe folgen einem, der verdrängt wird. . .

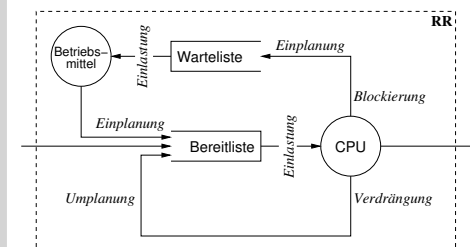


- da die Zuteilung der Zeitscheiben an Prozesse nach FCFS geschieht, werden kurze Prozesse nach wie vor benachteiligt:
  - E/A-intensive Prozesse** schöpfen ihre Zeitscheibe selten voll aus
    - sie beenden ihren CPU-Stoß freiwillig
      - vor Ablauf der Zeitscheibe
  - CPU-intensive Prozesse** schöpfen ihre Zeitscheibe meist voll aus
    - sie beenden ihren CPU-Stoß unfreiwillig
      - durch Verdrängung
- unabhängig davon werden jedoch alle Prozesse immer reihum bedient
- wird eine Zeitscheibe durch einen Prozess nicht ausgeschöpft, verteilt sich die CPU-Zeit zu Ungunsten E/A-intensiver Prozesse
  - E/A-intensive Prozesse werden schlechter bedient
  - E/A-Geräte sind schlecht ausgelastet
  - Varianz der Antwortzeit E/A-intensiver Prozesse kann beträchtlich sein
    - in Abhängigkeit vom jeweiligen Mix von Prozessen



RR mit Vorzugswarteschlange und variablen Zeitscheiben, um interaktive (d.h., E/A-intensive) Prozesse nicht zu benachteiligen.

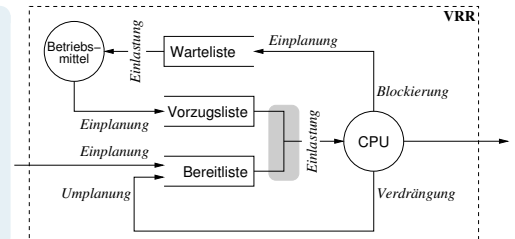
- auf E/A wartende Prozesse werden mit Beendigung ihres E/A-Stoßes bevorzugt eingeplant (d.h., bereit gestellt)
  - **Einplanung** mittels einer der Bereitliste vorgeschalteten **Vorzugsliste**
    - FIFO  $\leadsto$  evtl. Benachteiligung hoch-interaktiver Prozesse; daher. . .
    - aufsteigend sortiert nach dem **Zeitscheibenrest** eines Prozesses
  - **Umplanung** bei Ablauf der aktuellen Zeitscheibe
    - die Prozesse auf der Vorzugsliste werden zuerst eingelastet
    - sie bekommen die CPU für die Restdauer ihrer Zeitscheibe zugeteilt
    - bei Ablauf dieser Zeitscheibe werden sie in die Bereitliste eingereiht
  - erreicht durch strukturelle Maßnahmen — nicht durch analytische
- kein voll-verdrängendes Verfahren
  - die Einlastung auch des kürzesten bereitgestellten Prozesses erfolgt nicht zum Zeitpunkt seiner Bereitstellung
  - sondern frühestens nach Ablauf der aktuellen Zeitscheibe



- Bereitliste**
- lauffähiger Fäden<sup>a</sup>
  - dreiseitig bestückt
    - 2 × Einplanung
    - 1 × Umplanung
  - **unbedingt** bedient
- Warteliste**
- blockierter Fäden

<sup>a</sup>CPU „Warteliste“

- Bereitliste**
- wie bei RR
  - 2-seitig bestückt
    - 1 × Einplanung
    - 1 × Umplanung
  - **bedingt** bedient
- Warteliste**
- wie bei RR
- Vorzugsliste**
- **unbedingt** bedient



Zeitreihen bilden, analysieren und verwerten: nicht verdrängend.

- jeder Prozess wird entsprechend der für ihn im Durchschnitt oder maximal **erwarteten Bedienzeit** eingeplant
  - Grundlage dafür ist *à priori* Wissen über die **Prozesslaufzeiten**:
    - Stapelbetrieb** Programmierer setzen **Frist** (time limit)
    - Produktionsbetrieb** Erstellung einer **Statistik** durch Probeläufe
    - Dialogbetrieb** **Abschätzung** von CPU-Stoßlängen zur Laufzeit
  - Abarbeitung einer aufsteigend nach Laufzeiten sortierten Bereitsliste
    - Abschätzung erfolgt vor (statisch) oder zur (dynamisch) Laufzeit
- Verkürzung von Antwortzeiten und Steigerung der Gesamtleistung des Systems bei Benachteiligung längerer Prozesse
  - ein **Verhungern** (starvation) dieser Prozesse ist möglich
- ohne Konvoi-Effekt — jedoch ist als praktikable Implementierung nur die **näherungsweise Lösung** möglich
  - da die CPU-Stoßlängen nicht exakt im Voraus bestimmbar
  - die obere Grenze einer Stoßlänge nicht selten auch unvorhersagbar ist



- ein **heuristisches Verfahren**, das für jeden Prozess den Mittelwert über seine jeweiligen CPU-Stoßlängen bildet
  - damit ist die erwartete Länge des nächsten CPU-Stoßes eines Prozesses:

$$S_{n+1} = \frac{1}{n} \cdot \sum_{i=1}^n T_i$$

- der Mittelwert über alle seinen gemessenen CPU-Stoßlängen
- Problem dieser Berechnung ist die **gleiche Wichtung** aller CPU-Stöße
- jüngere CPU-Stöße machen jedoch die **Lokalität** eines Prozesses aus
  - diesen Stößen sollte eine größere Wichtung gegeben werden (vgl. S. 23)
- die **Messung** der Dauer eines CPU-Stoßes geschieht im Moment der Prozesseinlastung (d.h., der Prozessumschaltung)
  - Stoppzeit  $T_2$  von  $P_j$  entspricht (in etwa) der Startzeit  $T_1$  von  $P_{j+1}$ 
    - gemessen in **Uhrzeit** (clock time) oder **Uhrtick** (clock tick)
  - dann ergibt  $T_2 - T_1$  die gemessene CPU-Stoßlänge für jeden Prozess  $P_i$
  - der Differenzwert wird im jeweiligen Prozesskontrollblock akkumuliert



- mittels **Dämpfungsfilter** (decay filter), d.h., der **Dämpfung** (decay) der am weitesten zurückliegenden CPU-Stöße:

$$\begin{aligned} S_{n+1} &= \frac{1}{n} \cdot T_n + \frac{n-1}{n} \cdot S_n \\ &= \alpha \cdot T_n + (1 - \alpha) \cdot S_n \end{aligned}$$

- ergänzt um zuletzt gemessener ( $T_n$ ) und geschätzter ( $S_n$ ) CPU-Stoßlänge
- für den konstanten **Wichtungsfaktor**  $\alpha$  gilt dabei:  $0 < \alpha < 1$ 
  - drückt die **relative Wichtung** einzelner CPU-Stöße der Zeitreihe aus
- um die Wirkung des Wichtungsfaktors zu verdeutlichen, die teilweise Expansion der Gleichung wie folgt:
  - $S_{n+1} = \alpha T_n + (1 - \alpha) \alpha T_{n-1} + \dots + (1 - \alpha)^i \alpha T_{n-1} + \dots + (1 - \alpha)^n S_1$
- Beispiel der Entwicklung für  $\alpha = 0.8$ :
  - $S_{n+1} = 0.8 T_n + 0.16 T_{n-1} + 0.032 T_{n-2} + 0.0064 T_{n-3} + \dots$
  - zurückliegende CPU-Stöße des Prozesses verlieren schnell an Gewicht



Verhungersfreies SPN.

- Prozesse werden nach ihrer **erwarteten Bedienzeit** eingeplant und periodisch unter Berücksichtigung ihrer **Wartezeit** umgeplant
  - in regelmäßigen Zeitabständen wird ein Verhältniswert  $R$  berechnet:

$$R = \frac{w + s}{s}$$

- $w$  aktuell abgelaufene Wartezeit eines Prozesses
- $s$  erwartete (d.h., abgeschätzte) Bedienzeit eines Prozesses
- ausgewählt wird der Prozess mit dem größten Verhältniswert  $R$
- die periodische Aktualisierung betrifft alle Einträge in der Bereitliste und findet im Hintergrund des aktuellen Prozesses statt
  - ausgelöst durch einen **Uhrtick** (clock tick)
- Anmerkung: ein Anstieg der Wartezeit eines Prozesses bedeutet seine **Alterung** (aging)
  - der Alterung entgegenwirken (**anti-aging**), beugt Verhungern vor



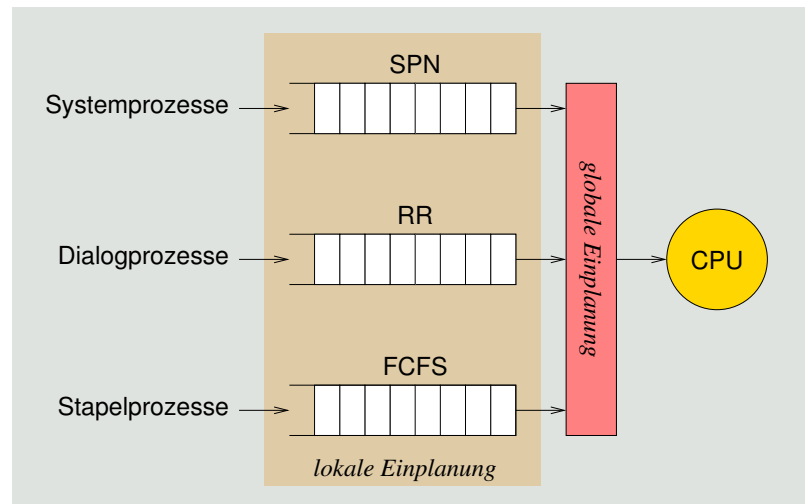
Verdrängendes SPN, Verhungerungsgefahr, Effektivität von VRR.

- Prozesse werden nach ihrer **erwarteten Bedienzeit** eingeplant und in unregelmäßigen Zeitabständen **spontan** umgeplant
  - sei  $T_{et}$  die erwartete CPU-Stoßlänge eines eintreffenden Prozesses
  - sei  $T_{rt}$  die verbleibende CPU-Stoßlänge des laufenden Prozesses
  - der laufende Prozess wird verdrängt, wenn gilt:  $T_{et} < T_{rt}$
- die **Umplanung** erfolgt ereignisbedingt und (ggf. voll) verdrängend im Moment der Ankunftszeit eines Prozesses
  - z.B. bei Beendigung des E/A-Stoßes eines wartenden Prozesses
  - allgemein: bei Aufhebung der Wartebedingung für einen Prozess
- bei **Verdrängung** kommt der betreffende Prozess entsprechend der Restdauer seiner erwarteten CPU-Stoßlänge auf die Bereitliste
  - führt allgemein zu besseren Antwort- und Durchlaufzeiten
  - gegenüber VRR steht der Aufwand zur CPU-Stoßlängenabschätzung



Unterstützt Mischbetrieb: Vorder- und Hintergrundbetrieb.

- Prozesse werden nach ihrem **Typ** (d.h., nach den für sie zutreffend geglaubten Eigenschaften) eingeplant
  - Aufteilung der Bereitliste in separate („getypte“) Listen
    - z.B. für System-, Dialog- und Stapelprozesse
  - mit jeder Liste eine **lokale Einplanungsstrategie** verbinden
    - z.B. SPN, RR und FCFS
  - zwischen den Listen eine **globale Einplanungsstrategie** definieren
    - statisch** – Liste einer bestimmten Prioritätsebene fest zuordnen
      - Verhungerungsgefahr für Prozesse tiefer liegender Listen
    - dynamisch** – die Listen im Zeitmultiplexverfahren wechseln
      - z.B. 40 % System-, 40 % Dialog-, 20 % Stapelprozesse
- dem Prozess einen Typen zuzuordnen ist eine statische Entscheidung
  - sie wird zum Zeitpunkt der Prozesserschöpfung getroffen



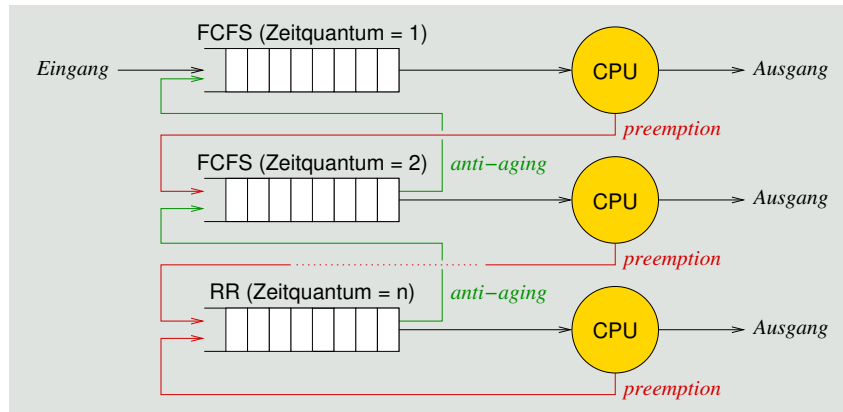
Begünstigt kurze/interaktive Prozesse, ohne die relativen Stoßlängen kennen zu müssen.

- Prozesse werden nach ihrer **Ankunftszeit** ein- und in regelmäßigen Zeitabständen (periodisch) umgeplant
  - Hierarchie von Bereitlisten, je nach Anzahl der **Prioritätsebenen**
    - erstmalig eintreffende Prozesse steigen oben ein
    - Zeitscheibenablauf drückt den laufenden Prozess weiter nach unten
  - je nach Ebene verschiedene Einreihungsstrategien und -parameter
    - unterste Ebene arbeitet nach RR, alle anderen (höheren) nach FCFS
    - die Zeitscheibengrößen nehmen von oben nach unten zu
- **Bestrafung (penalization)**
  - Prozesse mit langen CPU-Stößen fallen nach unten durch
    - ggf. wird der Alterung entgegengewirkt: Prozesse wieder anheben
  - Prozesse mit kurzen CPU-Stößen laufen relativ schnell durch





## FB: Bestrafung und Bewährung



multilevel feedback queue (MLFQ)

## Gliederung

Einführung

Einordnung  
Klassifikation

Verfahrensweisen  
Kooperativ  
Verdrängend  
Probabilistisch  
Mehrstufig

Zusammenfassung

## Gegenüberstellung

	FCFS	RR	VRR	SPN	HRRN	SRTF	FB
kooperativ	✓			(✓)	(✓)		
verdrängend		✓	✓			✓	✓
probabilistisch				✓	✓	✓	
deterministisch	keine bzw. nicht von sich aus allein ~ EZS [8]						

- **MLQ** erlaubt die Kombination all dieser Verfahren, jedoch abgestuft und nicht alle zusammen auf derselben Ebene
  - dadurch wird letztlich eine **Priorisierung** der Strategien vorgenommen
    - entsprechend der globalen Strategie, die den Ebenenwechsel steuert
  - teilweise wird so speziellen Anwendungsbedürfnissen entgegengekommen
    - z.B. FCFS priorisieren ~ „number crunching“ fördern
- jedes dieser Verfahren stellt bestimmte **Gütemerkmale** [3] in den Vordergrund und vergibt damit indirekt Prioritäten an Prozesse

## Prioritäten setzende Verfahren

*Statische Prioritäten (MLQ) vs. dynamische Prioritäten (VRR, SPN, SRTF, HRRN, FB).*

- **Prozessvorrang** bedeutet die bevorzugte Einlastung von Prozessen mit höherer Priorität und wird auf zwei Arten bestimmt:
  - statisch**
    - Zeitpunkt der **Prozesserzeugung** ~ Laufzeitkonstante
    - wird im weiteren Verlauf nicht mehr verändert
    - erzwingt die deterministische Ordnung zw. Prozessen
  - dynamisch**
    - „jederzeit“ im **Prozessintervall** ~ Laufzeitvariable
    - die Berechnung erfolgt durch das Betriebssystem
      - ggf. in Kooperation mit den Anwendungsprogrammen
    - erzwingt **keine** deterministische Ordnung zw. Prozessen
- damit ist allerdings noch nicht **Echtzeitverarbeitung** garantiert, bei der Prozessvorrang eine maßgebliche Rolle spielt
  - die **Striktheit von Terminvorgaben** ist einzuhalten: weich, fest, hart
    - entsprechend der jeweiligen Anforderungen der Anwendungsdomäne
  - keines der behandelten Verfahren sichert dies dem Anwendungssystem zu



- Prozesseinplanung unterliegt einer breit gefächerten **Einordnung**
  - kooperativ/verdrängend
  - deterministisch/probabilistisch
  - statisch/dynamisch
  - asymmetrisch/symmetrisch
- die entsprechenden **Verfahrensweisen** sind z.T. sehr unterschiedlich
  - FCFS: kooperativ
  - RR, VRR: verdrängend
  - SPN, HRRN, SRTF: probabilistisch
  - MLQ, FB (MLFQ): mehrstufig
- Prioritäten setzende Verfahren legen einen **Prozessvorrang** fest
  - FCFS: Ankunftszeit
  - RR: Ankunftszeit, VRR: Ankunftszeit nach Beendigung eines E/A-Stoßes
  - SPN: CPU-Stoßlänge, HRRN: Verhältniswert, SRTF: CPU-Reststoßlänge
- eine weitere Dimension ist die **Striktheit von Terminvorgaben**
  - die jedoch keins der behandelten Verfahren an sich berücksichtigt. .



- [1] COFFMAN, E. G. ; DENNING, P. J.:  
*Operating System Theory*.  
Prentice Hall, Inc., 1973
- [2] CONWAY, R. W. ; MAXWELL, L. W. ; MILLNER, L. W.:  
*Theory of Scheduling*.  
Addison-Wesley, 1967
- [3] KLEINÖDER, J. ; SCHRÖDER-PREIKSCHAT, W. :  
Einplanungsgrundlagen.  
In: [5], Kapitel 9.1
- [4] KLEINÖDER, J. ; SCHRÖDER-PREIKSCHAT, W. :  
Prozesse.  
In: [5], Kapitel 6.1
- [5] KLEINÖDER, J. ; SCHRÖDER-PREIKSCHAT, W. ; LEHRSTUHL INFORMATIK 4 (Hrsg.):  
*Systemprogrammierung*.  
FAU Erlangen-Nürnberg, 2015 (Vorlesungsfolien)
- [6] KLEINROCK, L. :  
*Queuing Systems*. Bd. I: Theory.  
John Wiley & Sons, 1975



## Literaturverzeichnis II

- [7] LISTER, A. M. ; EAGER, R. D.:  
*Fundamentals of Operating Systems*.  
The Macmillan Press Ltd., 1993. –  
ISBN 0-333-59848-2
- [8] LIU, J. W. S.:  
*Real-Time Systems*.  
Prentice-Hall, Inc., 2000. –  
ISBN 0-13-099651-3

