

# Überblick

## Prozesseinplanung

- Prozessorzuteilungseinheit
- Ebenen der Prozessorzuteilung
- Zustandsübergänge
- Gütemerkmale
- Verfahrensweisen
- Grundlegende Strategien
- Fallstudien
- Zusammenfassung

## Programmfaden

Einplanungseinheit (engl. *unit of scheduling*) für die Vergabe der CPU

**Ablaufplanung** von Fäden erfolgt **betriebsmittelorientiert** und ist ggf. **ereignisgesteuert** oder **zeitgesteuert**

- ▶ die Laufbereitschaft eines Fadens hängt von der Verfügbarkeit all jener Betriebsmittel ab, die für seinen Ablauf erforderlich sind
- ▶ die Bereitstellung von Betriebsmitteln (ggf. durch andere Fäden) kann die sofortige Einplanung von Fäden bewirken
- ▶ oder die Einplanung erfolgt in fest vorgegebenen Zeitintervallen

**Einplanung** eines Fadens ist nicht gleichzusetzen mit **Einlastung**:

- ▶ Einplanung ist der Vorgang der Reihenfolgenbildung von Aufträgen
  - ▶ Einlastung ist der Moment der Zuteilung von Betriebsmitteln
- Vorgänge, die ent- oder gekoppelt (**zeitversetzt/zeitgleich**) sein können

# Fadenverläufe

Stoßbetrieb (engl. *burst mode*)

**Laufphase:** CPU-Stoß (engl. *CPU burst*)

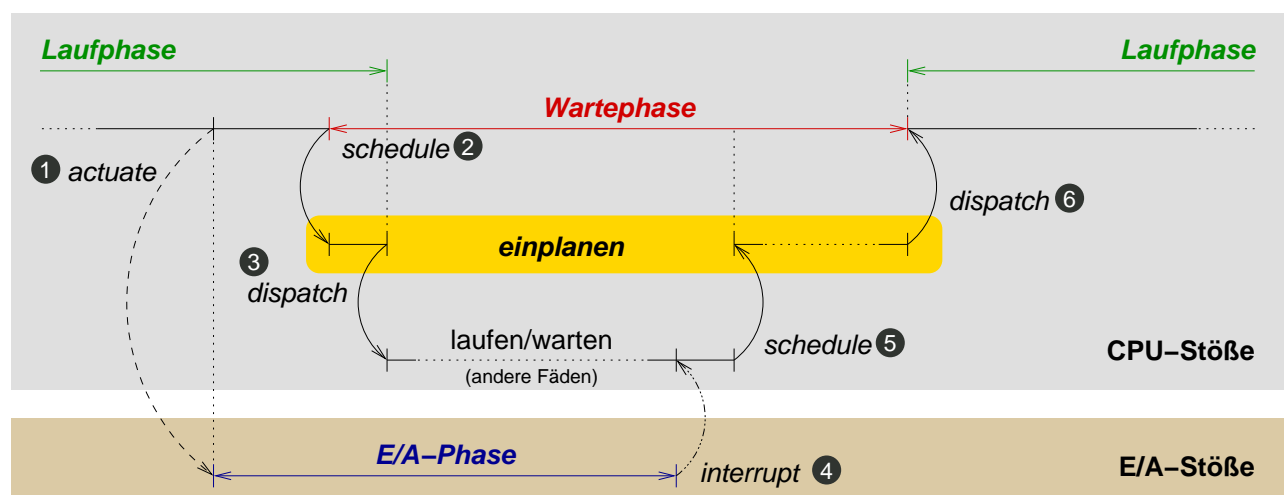
- ▶ aktive Phase eines Fadens (auch: Rechenphase)
  - ▶ alle zur Ausführung erforderlichen Betriebsmittel sind verfügbar
- ▶ der Faden ist **eingelastet**, ihm wurde die CPU zugeteilt

**Wartephase:** E/A-Stoß (engl. *I/O burst*), im weitesten Sinn

- ▶ inaktive Phase eines Fadens (auch: E/A-Phase)
  - ▶ nicht alle zur Ausführung erforderlichen Betriebsmittel sind verfügbar
- ▶ Ein-/Ausgabe abwarten bedeutet, auf Betriebsmittel zu warten
  - ▶ **konsumierbare Betriebsmittel:** Eingabedaten, Nachrichten, Signale, ...
  - ▶ **wiederverwendbare Betriebsmittel:** Puffer, Geräte, ..., die CPU
- ▶ die Betriebsmittel werden letztlich durch andere Fäden bereitgestellt
  - ▶ ein E/A-Gerät kann dabei als „externer Faden“ betrachtet werden

# Fadenverläufe (Forts.)

Lauf-, E/A- und Wartephase von Fäden



## Fadenverläufe (Forts.)

### Zusammenfassung

Fäden durchlaufen (im Betriebssystem) einen **Kontrollfluss** zur **Einplanung** und **Einlastung** anderer Fäden:

1. der laufende Faden stößt einen E/A-Vorgang an (*actuate*)
2. er wartet passiv auf die Beendigung der Ein-/Ausgabe (*schedule*)
  - ▶ Anforderung eines wiederverwendbaren/konsumierbaren Betriebsmittels
3. und lastet einen eingeplanten, lafbereiten Faden ein (*dispatch*)
4. die Beendigung der Ein-/Ausgabe wird signalisiert (*interrupt*)
  - ▶ Bereitsstellung des konsumierbaren Betriebsmittels „Signal“
5. der auf dieses Ereignis wartende Faden wird eingeplant (*schedule*)
6. der Faden wird eingelastet, sobald er an der Reihe ist (*dispatch*)

**Sonderfall:** ein Faden plant und lastet sich selbst ein, wenn sich die CPU mangels anderer lafbereiter Fäden im **Leerlauf** (engl. *idle state*) befindet

## Fäden als Mittel zur Leistungsoptimierung

### Arbeitsteilung in nebenläufigen/parallelen Systemen

**Überlappung** von Lauf- und Wartephasen erhöht die Rechnerauslastung

- ▶ die Wartephase eines Fadens als Laufphase anderer Fäden nutzen
- ▶ die Stöße anderer Fäden zum „Auffüllen“ von Wartephasen nutzen

**Auslastung** von CPU und Peripherie (E/A-Geräte) steigert sich

- ▶ eine CPU kann zu einem Zeitpunkt nur einen CPU-Stoß verarbeiten
- ▶ parallel dazu können jedoch mehrere E/A-Stöße laufen
  - ▶ ausgelöst während eines CPU-Stoßes: in der Laufphase eines Fadens wurden mehrere E/A-Vorgänge gestartet
  - ▶ ausgelöst von mehreren CPU-Stößen: die Wartephase eines Fadens wurde mit Laufphasen anderer Fäden gefüllt
- ▶ als Folge sind CPU und E/A-Geräte andauernd mit Arbeit beschäftigt

**Konsequenz:** bei weniger Prozessoren als Fäden, sind Fäden zu serialisieren

## Zwangsserialisierung von Programmfäden

In Bezug auf eine Instanz des Betriebsmittels „CPU“

Verlängerung der **absoluten Ausführungsdauer** später „eintreffender“ laufbereiter Fäden ist zu beobachten:

- ▶ Ausgangspunkt seien  $n$  Fäden mit gleichlanger Bearbeitungsdauer  $k$
- ▶ der erste Faden wird um die Zeitdauer  $0$  verzögert
- ▶ der zweite Faden um die Zeitdauer  $k$ , der  $i$ -te Faden um  $(i - 1) \cdot k$
- ▶ der letzte von  $n$  Fäden wird verzögert um  $(n - 1) \cdot k$

$$\frac{1}{n} \cdot \sum_{i=1}^n (i - 1) \cdot k = \frac{n - 1}{2} \cdot k$$

Vergrößerung der **mittleren Verzögerung** ist proportional zur Fadenanzahl

## Subjektive Empfindung der Fadenverzögerung

Nur bis zu einer bestimmten Last (# eingeplanter Fäden) ...

Startzeiten von Fäden verzögern sich im Mittel um:  $\frac{n-1}{2} \cdot t_{cpu}$

- ▶ mit  $t_{cpu}$  gleich der mittleren Dauer eines CPU-Stoßes
- ▶ sofern  $t_{cpu} \geq t_{ea}$ , der mittleren Dauer eines E/A-Stoßes
- ▶ die Praxis liefert als Regelfall jedoch ein anderes Bild:  $t_{cpu} \ll t_{ea}$

**Wartephasen bei E/A-Operationen dominieren** die Fadenverzögerung

- ▶ zwischen CPU- und E/A-Stößen besteht eine große Zeitdiskrepanz
- ▶ der proportionale Verzögerungsfaktor bleibt weitestgehend verborgen
- ▶ er greift erst ab einer bestimmten Anzahl von Programmfäden
- ▶ sehr häufig ist die Fadenverzögerung daher nicht wahrnehmbar

☞ **Überlast** durch zuviel eingeplante Fäden **ist zu vermeiden**

# Dauerhaftigkeit von Zuteilungsentscheidungen

## Logische Ebenen der Prozesseinplanung

### langfristige Einplanung (engl. *long-term scheduling*)

[s – min]

- ▶ **Lastkontrolle**, Grad an Mehrprogrammbetrieb einschränken
- ▶ Programme laden und/oder zur Ausführung zulassen
- ▶ Prozesse der mittel- bzw. kurzfristigen Einplanung zuführen

### mittelfristige Einplanung (engl. *medium-term scheduling*)

[ms – s]

- ▶ Teil der **Umlagerungsfunktion** (engl. *swapping*)
- ▶ Programme vom Hinter- in den Vordergrundspeicher bringen
- ▶ Prozesse der langfristigen Einplanung zuführen

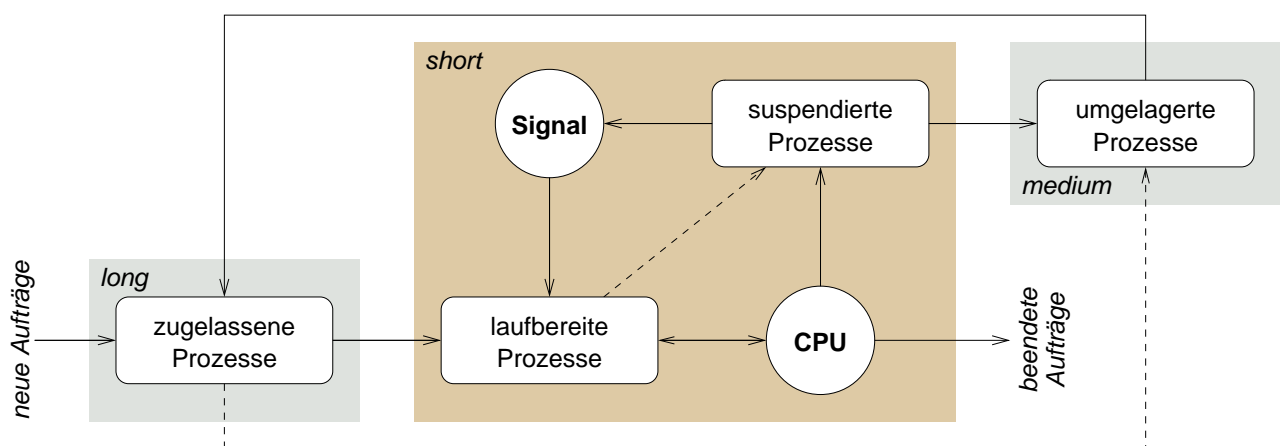
### kurzfristige Einplanung (engl. *short-term scheduling*)

[ $\mu$ s – ms]

- ▶ **Einlastungsreihenfolge** der Prozesse festlegen — obligatorisch

# Phasen der Prozesseinplanung

Lang- und mittelfristige Einplanung sind optional



Voraussetzung für Mehrprozessbetrieb ist die **kurzfristige Einplanung**

- ▶ *laufbereite Prozesse* erwarten die Zuteilung des wiederverwendbaren Betriebsmittels „CPU“, d.h. den Start ihrer Laufphase
- ▶ *suspendierte Prozesse* erwarten die Zuteilung eines konsumierbaren Betriebsmittels „Signal“, d.h. das Ende ihrer Wartephase

## Prozesszustand vs. Einplanungsebene

Prozesse haben in Abhängigkeit von der Einplanungsebene (S. 9-9) zu einem Zeitpunkt einen **logischen Zustand**:

**kurzfristig** (engl. *short-term*)

- ▶ bereit, laufend, blockiert

**mittelfristig** (engl. *medium-term, mid-term*)

- ▶ schwebend bereit, schwebend blockiert

**langfristig** (engl. *long-term*)

- ▶ erzeugt, gestoppt, beendet

Anwendungsfälle legen fest, welche der Einplanungsebenen von einem Betriebssystem wirklich zur Verfügung zu stellen sind, nicht umgekehrt.

## Kurzfristige Einplanung

Festlegung der Prozessorzuteilungsreihenfolge

Betriebssystem bietet **Mehrprozessbetrieb** (engl. *multi-processing*) auf Basis der **Serialisierung von Programmfäden**:

**bereit** (engl. *ready*) zur Ausführung durch den Prozessor (die CPU)

- ▶ der Prozess ist auf der Bereitliste (engl. *ready list*)
- ▶ das Einplanungsverfahren bestimmt die Listenposition

**laufend** (engl. *running*), Zuteilung des Betriebsmittels „CPU“ ist erfolgt

- ▶ der Prozess vollzieht seinen CPU-Stoß
- ▶ zu einem Zeitpunkt pro Prozessor nur ein laufender Prozess

**blockiert** (engl. *blocked*) auf ein bestimmtes Ereignis

- ▶ der Prozess erwartet die Zuteilung eines Betriebsmittels
  - ▶ mit Ausnahme des Betriebsmittels „CPU“
- ▶ ggf. vollzieht der Prozess auch seinen E/A-Stoß

Spezialfall: „blockiert“  $\mapsto$  „laufend“  $\rightsquigarrow$  voll verdrängend (S. 9-17)

## Mittelfristige Einplanung

Festlegung der Umlagerungsreihenfolge

Betriebssystem implementiert die **Umlagerung** (engl. *swapping*) von kompletten Programmen bzw. logischen Adressräumen:

**schwebend bereit** (engl. *ready suspend*)

- ▶ Adressraum des Prozesses ist ausgelagert
  - ▶ verschoben in den Hintergrundspeicher
  - ▶ „*swap-out*“ ist erfolgt
  - ▶ „*swap-in*“ wird erwartet
- ▶ die Einlastung des Prozesses ist außer Kraft
  - ▶ genauer: aller Fäden des Adressraums

**schwebend blockiert** (engl. *blocked suspend*)

- ▶ ausgelagerter ereigniserwartender Prozess
- ▶ Ereignisseintritt  $\mapsto$  „schwebend bereit“

Variante: „schwebend bereit“  $\mapsto$  „bereit“  $\rightsquigarrow$  langfristige Einplanung

## Langfristige Einplanung

Festlegung der Zulassungsreihenfolge

Betriebssystem verfügt über Funktionen zur **Lastkontrolle** und steuert den Grad an Mehrprogrammbetrieb:

**erzeugt** (engl. *created*) und fertig zur Programmverarbeitung

- ▶ der Prozess ist instanziiert, Programm wurde zugeordnet
- ▶ ggf. steht die Speicherzuteilung jedoch noch aus

**gestoppt** (engl. *stopped*) und erwartet seine Fortsetzung

- ▶ der Prozess wurde angehalten (z.B. `~Z` bzw. `kill(2)`)
- ▶ mögl. Gründe: Überlast, **Verklemmungsvermeidung**, ...

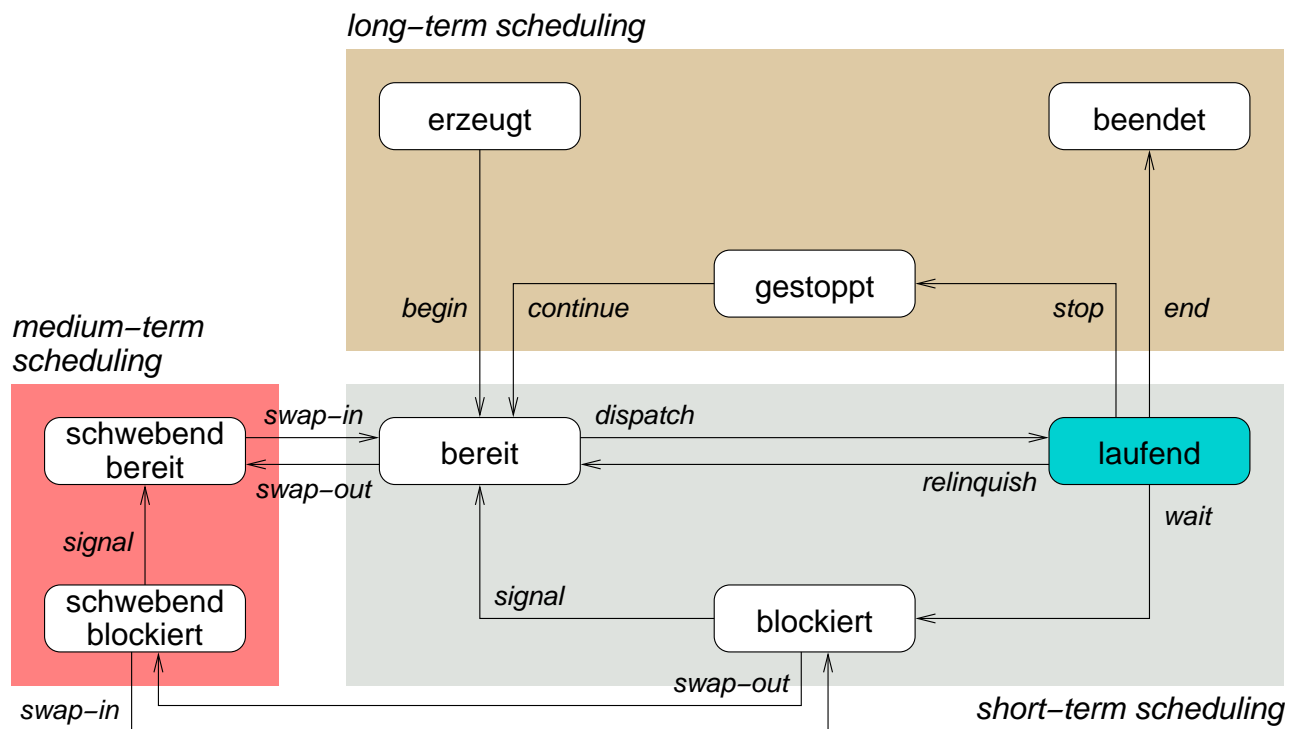
**beendet** (engl. *ended*) und erwartet seine Entsorgung

- ▶ der Prozess ist terminiert, Betriebsmittelfreigabe erfolgt
- ▶ ggf. muss ein anderer Prozess den „Kehraus“ vollenden

Achtung: „gestoppt“ werden können auch bereite/blockierte Prozesse

# Abfertigungszustände im Zusammenhang

Je nach Betriebssystem/-art sind weitere „Zwischenzustände“ anzufinden



## Einplanungs-/Auswahlzeitpunkt

Übergänge in den Zustand „bereit“ aktualisieren die Bereitliste:

- ▶ eine Entscheidung über die **Einlastungsreihenfolge** wird getroffen
- ▶ eine **Funktion der Einplanungsstrategie** wird ausgeführt

**Einplanung** (engl. *scheduling*) bzw. **Umplanung** (engl. *rescheduling*):

- ▶ nachdem ein Prozess erzeugt worden ist *begin*
- ▶ wenn ein Prozess freiwillig die CPU abgibt *relinquish*
- ▶ sofern das von einem Prozess erwartete Ereignis eingetreten ist *signal*
- ▶ sobald ein Prozess wieder aufgenommen werden kann *continue*

Prozesse können dazu gedrängt werden, die CPU freiwillig abzugeben

- ▶ sofern **verdrängende** (engl. *preemptive*) **Prozesseinplanung** erfolgt



# Verdrängende Prozesseinplanung

Ereigniseintritt → Einplanung → Einlastung

**Verdrängung** (engl. *preemption*) des laufenden Prozesses von der CPU bedeutet folgendes:

1. ein Ereignis tritt ein, dessen Behandlungsverlauf zum Planer führt
  - ▶ der das Ereignis ggf. **erwartende Prozess** wird eingeplant
2. der (vom Ereignis unterbrochene) **laufende Prozess** wird eingeplant
3. ein **eingeplanter Prozess** wird ausgewählt und eingelastet
  - ▶ ggf. handelt es sich dabei um den unter 1. eingeplanten Prozess

Einplanung und Einlastung von Prozessen erfolgt nicht immer zeitnah zum Ereigniseintritt (d.h., dem Moment der Verdrängungsaufforderung):

- ▶ die Verdrängung eines Prozesses verzögert sich ggf. unbestimmt lang
- ▶ Ursache dafür ist u.a. die Architektur von Betriebssystem(kern)en

☞ ggf. entstehende **Latenzzeiten** können Anwendungen beeinträchtigen

# Latenzzeiten und Determinismus

Verdrängung als Querschnittsbelang von Betriebssystemen

**Einplanungslatenz** (engl. *scheduling latency*) ist unvermeidbar, nicht jedoch ihre Unbestimmtheit — **deterministische Einplanung**:

- ▶ zu jedem Zeitpunkt ist der nachfolgende Schritt eindeutig festgelegt
  - ▶ unabhängig von Systemlast/-aktivitäten in dem Moment
- ▶ die Latenzzeit ist konstant oder mit fester oberer Schranke variabel

**Einlastungslatenz** (engl. *dispatching latency*), d.h. die Zeitspanne zwischen Einplanung und Verdrängung, führt zur weiteren Differenzierung:

**verdrängend** (engl. *preemptive*) ~ „programmierte Verdrängung“

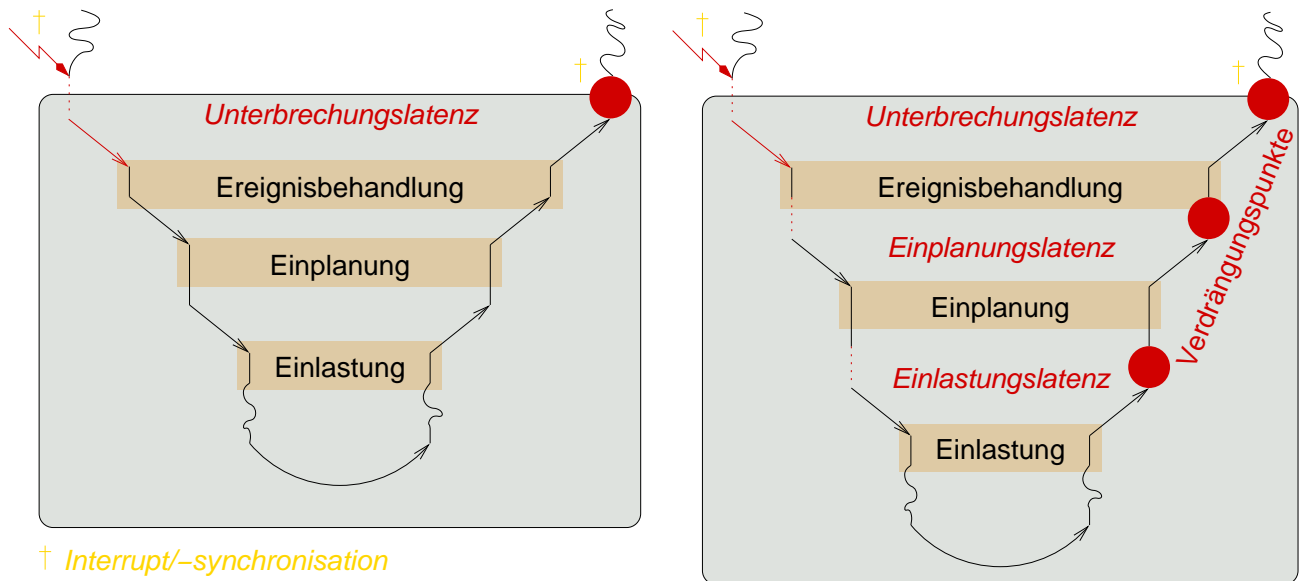
- ▶ Einlastung nur an bestimmten Stellen freigegeben
- ▶ **Verdrängungspunkte** (engl. *preemption points*)

**voll verdrängend** (engl. *full preemptive*) ≡ Einlastung **jederzeit** erlaubt

zu guter Letzt: **Unterbrechungslatenz** (engl. *interrupt latency*)...

# Latenzzeiten in Bezug zum Betriebsmodus

Asynchrone Programmunterbrechungen bleiben eine Quelle der Ungewissheit



voll verdrängend

verdrängend

## Dimensionen der Prozesseinplanung

Kriterien zur Aufstellung einer Einlastungsreihenfolge von Prozessen

benutzerorientierte Kriterien fokussieren auf **Benutzerdienlichkeit**

- ▶ d.h. das vom jeweiligen Benutzer wahrgenommene Systemverhalten
- ▶ bestimmen im großen Maße die Akzeptanz des Systems
  - ▶ bedeutsam für die Anwendungsdomäne in technischer Hinsicht
  - ▶ z.B. Einhaltung und Durchsetzung von Gütemerkmalen

systemorientierte Kriterien haben **Systemperformanz** im Vordergrund

- ▶ d.h. die effektive und effiziente Auslastung der Betriebsmittel
- ▶ bestimmen im großen Maße die „Rentabilität“ des Systems
  - ▶ bedeutsam für die Anwendungsdomäne in kommerzieller Hinsicht
  - ▶ z.B. Amortisierung hoher Anschaffungskosten von Großrechnern

Ausschlusskriterien sind dies nicht, vielmehr eine **Schwerpunktsetzung**:

- ▶ gute Systemperformanz ist auch der Benutzerdienlichkeit förderlich

## Benutzerorientierte Kriterien

**Charakteristische Anforderungserkmale** bestimmter Anwendungsdomänen

- Antwortzeit** Minimierung der Zeitdauer von der Auslösung eines Systemaufrufs bis zur Entgegennahme der Rückantwort, bei gleichzeitiger Maximierung der Anzahl interaktiver Prozesse.
- Durchlaufzeit** Minimierung der Zeitdauer vom Starten eines Prozesses bis zu seiner Beendigung, d.h., der effektiven Prozesslaufzeit und aller anfallenden Prozesswartezeiten.
- Termineinhaltung** Starten und/oder Beendigung eines Prozesses (bis) zu einem fest vorgegebenen Zeitpunkt.
- Vorhersagbarkeit** Deterministische Ausführung des Prozesses unabhängig von der jeweils vorliegenden Systemlast.

## Systemorientierte Kriterien

**Wünschenswerte Anforderungserkmale** vieler Anwendungsdomänen

- Durchsatz** Maximierung der Anzahl vollendeter Prozesse pro vorgegebener Zeiteinheit, d.h., der (im System) geleisteten Arbeit.
- Prozessorauslastung** Maximierung des Prozentanteils der Zeit, während der die CPU Prozesse ausführt, d.h., „sinnvolle“ Arbeit leistet.
- Gerechtigkeit** Gleichbehandlung der auszuführenden Prozesse und Zusicherung, den Prozessen innerhalb gewisser Zeiträume die CPU zuzuteilen.
- Dringlichkeiten** Vorzugbehandlung des Prozesses mit der höchsten (statischen/dynamischen) Priorität.
- Lastausgleich** Gleichmäßige Betriebsmittelauslastung; ggf. auch Vorzugbehandlung der Prozesse, die stark belastete Betriebsmittel eher selten belegen.

## Betriebsart vs. Einplanungskriterien

Prozesseinplanung impliziert eine Betriebsart und umgekehrt

**allgemein** Gerechtigkeit, Lastausgleich

- ▶ **Durchsetzung der jeweiligen Strategie**

**Stapelbetrieb** Durchsatz, Durchlaufzeit, Prozessorauslastung

**interaktiver Betrieb** Antwortzeit; **Proportionalität**:

Für bestimmte Prozesse ein Laufzeitverhalten „simulieren“, das nicht unbedingt dem technischen Leistungsvermögen des Rechensystems entspricht:  
☞ Geschwindigkeit ist Hexerei?

- ▶ Benutzer haben meist eine inhärente Vorstellung über die Dauer bestimmter Aktionen.
- ▶ Dieser (oft auch falschen) Vorstellung sollte das System aus Gründen der Benutzerakzeptanz möglichst entsprechen.

**Echtzeitbetrieb** Dringlichkeit, Termineinhaltung, Vorhersagbarkeit

- ▶ oft im Konflikt mit Gerechtigkeit/Lastausgleich

## Kooperativ vs. Präemptiv

Souverän ist die Anwendung oder das Betriebssystem

*cooperative scheduling* voneinander abhängiger Prozesse

- ▶ „unkooperative“ Prozesse können die **CPU monopolisieren**
- ▶ während der Programmausführung müssen Systemaufrufe erfolgen
  - ▶ **Endlosschleifen ohne Systemaufrufe** im Anwendungsprogramm verhindern Prozesse anderer Anwendungsprogramme
- ▶ alle Systemaufrufe müssen den Scheduler durchlaufen

*preemptive scheduling* voneinander unabhängiger Prozesse

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

## Deterministisch vs. Probabilistisch

Mit oder ohne *à priori* Wissen

*deterministic scheduling* bekannter, exakt vorberechneter Prozesse

- ▶ alle **CPU-Stoßlängen** und ggf. auch **Termine** sind bekannt
  - ▶ bei (strikten) Echtzeitsystemen mindestens die Stoßlänge des „schlimmsten Falls“ (engl. *worst-case execution time*, WCET)
- ▶ die genaue Vorhersage der CPU-Auslastung ist möglich
- ▶ das System stellt die Einhaltung von **Zeitgarantien** sicher
- ▶ die Zeitgarantien gelten unabhängig von der jeweiligen Systemlast

*probabilistic scheduling* unbekannter 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

## Statisch vs. Dynamisch

Entkoppelt von oder gekoppelt mit der Programmausführung

*offline scheduling* statisch, vor der Programmausführung

- ▶ **Komplexität** verbietet Ablaufplanung im laufenden Betrieb
  - ▶ zu berechnen, ob die Einhaltung aller Zeitvorgaben garantiert werden kann, ist ein NP-vollständiges Problem
  - ▶ die Berechnungskomplexität wird zum kritischen Faktor, wenn auf jede abfangbare katastrophale Situation zu reagieren ist
- ▶ Ergebnis der Vorbereitung 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**

*online scheduling* dynamisch, während der Programmausführung

- ▶ Stapelsysteme, interaktive Systeme, verteilte Systeme
- ▶ schwache und feste Echtzeitsysteme

# Asymmetrisch vs. Symmetrisch

An eine CPU gebundene oder ungebundene Programmausführung

*asymmetric scheduling* ist abhängig von Eigenschaften der Ebene  $2/3$

- ▶ obligatorisch in einem asymmetrischen Multiprozessorsystem
  - ▶ die Rechnerarchitektur sieht **programmierbare Spezialprozessoren** vor
  - ▶ z.B. Grafik- und/oder Kommunikationsprozessoren einerseits und ein Feld konventioneller (gleichartiger) CPUs andererseits
  - ▶ auszuführende Programme sind an bestimmte Prozessoren gebunden
- ▶ optional in einem symmetrischen Multiprozessorsystem (s.u.)
  - ▶ das Betriebssystem hat absolut freie Hand über die Prozessorvergabe
- ▶ Prozesse in funktionaler Hinsicht ungleich verteilen (müssen)

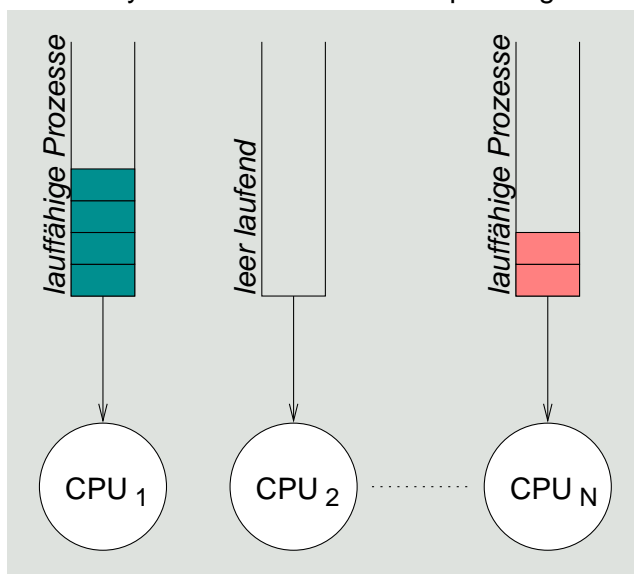
*symmetric scheduling* ist abhängig von Eigenschaften der Ebene  $2$

- ▶ identische Prozessoren, alle geeignet zur Programmausführung
- ▶ Prozesse werden gleich auf die Prozessoren verteilt: **Lastausgleich**

# Asymmetrisch vs. Symmetrisch (Forts.)

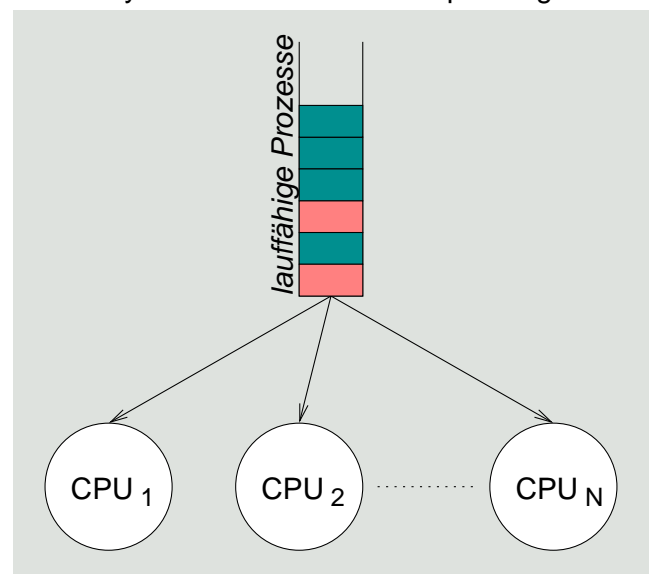
Jedem Prozessor seine eigene oder allen Prozessoren eine gemeinsame Bereitliste

asymmetrische Prozesseinplanung



separate Bereitlisten

symmetrische Prozesseinplanung



gemeinsame Bereitliste

- |                                 |                                 |
|---------------------------------|---------------------------------|
| ▶ ungleichmäßige Auslastung     | ▶ gleichmäßige Auslastung       |
| ▶ Unterbrechungssynchronisation | ▶ Multiprozessorsynchronisation |

# Klassische Einplanungs- bzw. Auswahlverfahren

## Überblick

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)	
	▶ Rasterfahndung. . .	

## FCFS (engl. *first come, first served*)

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

Prozesse werden nach ihrer **Ankunftszeit** (engl. *arrival time*) eingeplant und in der sich daraus ergebenden Reihenfolge auch verarbeitet

- ▶ nicht-verdrängendes Verfahren, setzt kooperative Prozesse voraus

Gerechtigkeit auf Kosten hoher Antwortzeit und niedrigem 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  $\left\{ \begin{array}{l} \text{begünstigt} \\ \text{benachteiligt} \end{array} \right\}$

☞ **Konvoi(d)effekt**: mehrere kurze Aufträge folgen einem langen. . .

## FCFS (Forts.)

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
$\emptyset$					100	26.00

 $T_s$  = Bedienzeit,  $T_q$  = Durchlaufzeit

normalisierte Durchlaufzeit ( $T_q/T_s$ ) von C ist vergleichsweise sehr schlecht

- ▶ sie steht in einem extrem schlechten Verhältnis zur Bedienzeit  $T_s$
- ▶ typischer Effekt im Falle von kurzen Prozessen, die langen folgen

RR (engl. *round robin*)

Verdrängendes FCFS, Zeitscheiben, CPU-Schutz

Prozesse werden nach ihrer **Ankunftszeit** ein- und in regelmäßigen Zeitabständen (periodisch) umgeplant

- ▶ verdrängendes Verfahren, nutzt **periodische Unterbrechungen**
  - ▶ Zeitgeber (engl. *timer*) liefert asynchrone Programmunterbrechungen
- ▶ jeder Prozess erhält eine **Zeitscheibe** (engl. *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“



## RR (Forts.)

## Leistungsprobleme bei einem Mix von Prozessen

**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

☞ **Konvoi(d)effekt:** mehrere kurze CPU-Stöße folgen einem langen. . .

CPU-Zeit ist zu Gunsten CPU-intensiver Prozesse ungleich verteilt

- ▶ E/A-intensive Prozesse werden schlechter bedient
- ▶ E/A-Geräte sind schlecht ausgelastet

☞ **Varianz der Antwortzeit** E/A-intensiver Prozesse ist groß

VRR (engl. *virtual round robin*)

RR mit Vorzugswarteschlange und variablen Zeitscheiben

Prozesse werden mit Beendigung ihres E/A-Stoßes **bevorzugt eingeplant**, jedoch nicht (zwingend) bevorzugt/sofort eingelastet

- ▶ Einreihung in eine der Bereitliste vorgeschalteten **Vorzugsliste**
  - ▶ FIFO  $\rightsquigarrow$  evtl. Benachteiligung hoch-interaktiver Prozesse; daher. . .
  - ▶ aufsteigend sortiert nach dem **Zeitscheibenrest** eines Prozesses
- ▶ **Umplanung** erfolgt bei Beendigung des jeweils laufenden CPU-Stoßes
  - ▶ 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 Bereitsliste eingereiht

Vermeidung der bei RR möglichen ungleichen Verteilung von CPU-Zeiten

- ▶ bevorzugt werden interaktive Prozessen mit kurzen CPU-Stößen
- ▶ erreicht durch strukturelle Maßnahmen. . .

## SPN (engl. *shortest process next*)

Zeitreihen bilden, analysieren und verwerten

Prozesse werden nach ihrer **erwarteten Bedienzeit** eingeplant

- ▶ Grundlage dafür ist *à priori* Wissen über die **Prozesslaufzeiten**:
  - Stapelbetrieb** Programmierer setzen eine **Frist** (engl. *time limit*)
  - Produktionsbetrieb** Erstellung einer **Statistik** durch Probeläufe
  - Dialogbetrieb** **Abschätzung** von CPU-Stoßlängen zur Laufzeit
- ▶ Abarbeitung einer aufsteigend nach (vor/zur Laufzeit abgeschätzten) Prozesslaufzeiten sortierten Bereitsliste
  - ▶ statische oder dynamische Verfahrensweise

Verkürzung von Antwortzeiten und Steigerung der Gesamtleistung des Systems auf Kosten länger laufender Prozess

- ▶ ein **Verhungern** (engl. *starvation*) dieser Prozesse ist möglich

## SPN (Forts.)

Abschätzung der Dauer eines CPU-Stoßes

**Mittelwertbildung** über alle CPU-Stoßlängen eines Prozesses:

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

- ▶ Problem dieser Berechnung ist die **gleiche Wichtung** aller CPU-Stöße
- ▶ jüngere CPU-Stöße sollten mit größerer Wichtung eingehen: **Lokalität**

**Messung** der Dauer eines CPU-Stoßes geschieht bei Prozesseinlastung:

- ▶ Stoppzeit  $T_2$  von  $P_x$  entspricht (in etwa) der Startzeit  $T_1$  von  $P_y$ 
  - ▶ gemessen in **Uhrzeit** (engl. *clock time*) oder **Uhrtick** (engl. *clock tick*)
- ▶ Akkumulation der Differenzen  $T_2 - T_1$  für jeden Prozess  $P_i$

## SPN (Forts.)

Wichtung der CPU-Stöße, Dämpfungsfiler (engl. *decay filter*) einsetzen

**Dämpfung** (engl. *decay*) der am weitesten zurückliegenden CPU-Stöße:

$$S_{n+1} = \alpha \cdot T_n + (1 - \alpha) \cdot S_n$$

- ▶ für den konstanten Wichtungsfaktor  $\alpha$  gilt dabei:  $0 < \alpha < 1$
- ▶ er drückt die **relative Wichtung** einzelner CPU-Stöße der Zeitreihe aus
- ▶ teilweise Expansion der Gleichung führt zu:
  - ▶  $S_{n+1} = \alpha T_n + (1 - \alpha)\alpha T_{n-1} + \dots + (1 - \alpha)^i \alpha T_{n-1} + \dots + (1 - \alpha)^n S_1$
- ▶ Beispiel der Entwicklung für  $\alpha = 0.8$ :
  - ▶  $S_{n+1} = 0.8 T_n + 0.16 T_{n-1} + 0.032 T_{n-2} + 0.0064 T_{n-3} + \dots$

## SRTF (engl. *shortest remaining time first*)

Verdrängendes SPN, Verhungerungsgefahr, Effektivität von VRR

Prozesse werden nach ihrer **erwarteten Bedienzeit** eingeplant und in unregelmäßigen Zeitabständen **sporadisch** 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:  $T_{et} < T_{rt}$

**Umplanung** erfolgt ereignisbedingt und (ggf. voll) verdrängend

- ▶ z.B. bei Beendigung des E/A-Stoßes eines darauf wartenden Prozesses
- ▶ allgemein: bei Aufhebung der Wartebedingung für einen Prozess

**Verdrängung** führt zu besseren Antwort- und Durchlaufzeiten:

- ▶ gegenüber VRR steht der *Overhead* zur CPU-Stoßlängenabschätzung

## HRRN (engl. *highest response ratio next*)

SRTF ohne Verhungern der Prozesse

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

- ▶ **periodische Aktualisierung** aller Einträge in der Bereitliste
- ▶ ausgewählt wird der Prozess mit dem größten Verhältniswert  $R$

**Alterung** (engl. *aging*) von Prozessen meint einen Anstieg der Wartezeit

- ▶ der **Alterung entgegenwirken** (engl. *anti-aging*) beugt Verhungern vor

## MLQ (engl. *multilevel queue*)

Unterstützt Mischbetrieb: Vorder- und Hintergrundbetrieb

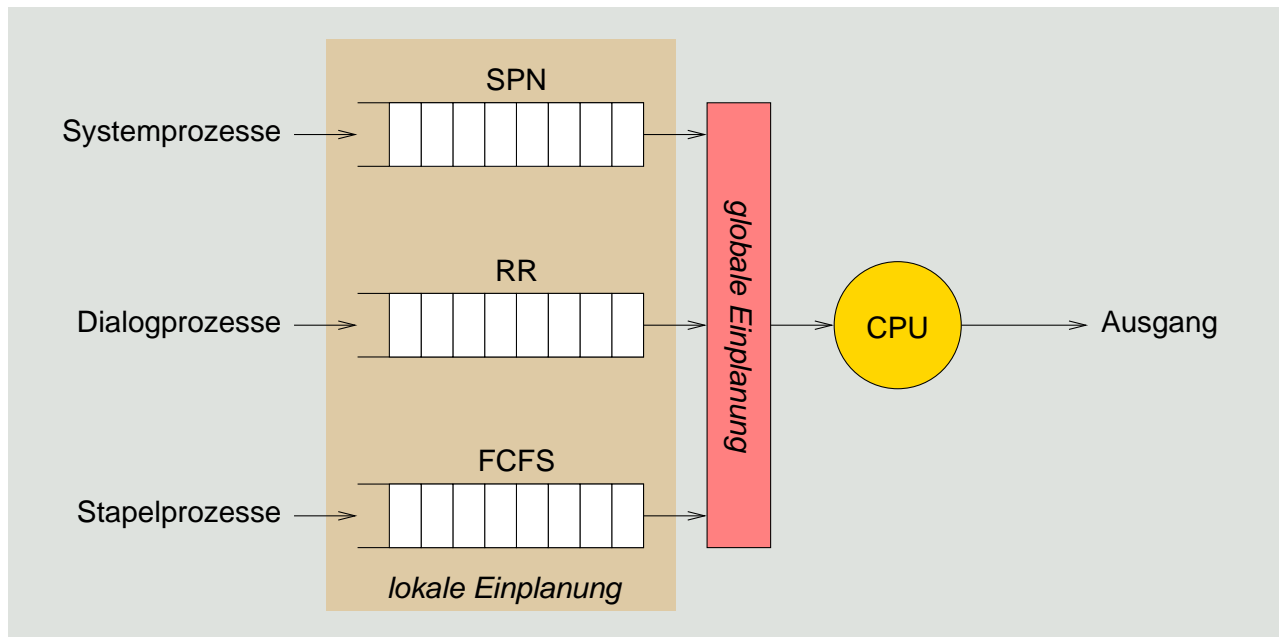
Prozesse werden nach ihrem **Typ** (d.h., nach den für sie zutreffend geglaubten Eigenschaften) eingeplant

- ▶ 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** eine Liste einer bestimmten Prioritätsebene fest zuordnen
    - ▶ Verhungerungsgefahr für Prozesse tiefer liegender Listen
  - dynamisch** die Listen im Zeitmultiplexverfahren wechseln
    - ▶ z.B. 40 % System-, 40 % Dialog- und 20 % Stapelprozesse

Prozessen Typen zuordnen ist eine statische Entscheidung: sie wird zum Zeitpunkt der Prozesserzeugung getroffen

## MLQ (Forts.)

## Mischbetrieb mit System-, Dialog- und Stapelprozessen

FB (engl. *feedback*)

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

Prozesse werden nach ihrer **Ankunftszeit** ein- und in regelmäßigen Zeitabständen (periodisch) umgeplant

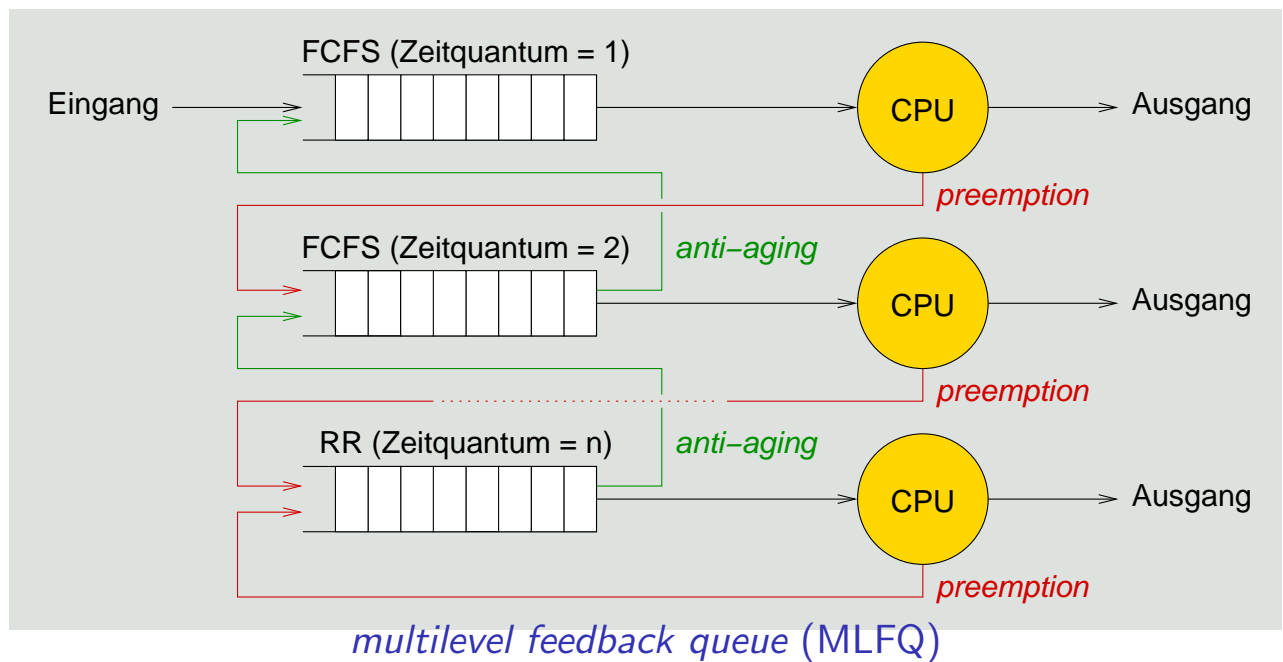
- ▶ Hierarchie von Bereitlisten, je nach Anzahl der **Prioritätsebenen**
  - ▶ erstmalig eintreffende Prozesse steigen oben ein
  - ▶ Zeitscheibenablauf drückt den laufenden Prozess weiter nach unten
- ▶ je nach Ebene verschiedene Einreihungsstrategien und -parameter
  - ▶ die unterste Ebene arbeitet nach RR, alle anderen (höheren) nach FCFS
  - ▶ die Zeitscheibengrößen nehmen von oben nach unten zu

**Bestrafung** (engl. *penalization*) von Prozessen mit langen CPU-Stößen

- ▶ Prozesse mit kurzen CPU-Stößen laufen relativ schnell durch
- ▶ Prozesse mit langen CPU-Stößen fallen nach unten durch
  - ▶ ggf. wird der Alterung entgegengewirkt: Prozesse wieder anheben

## FB (Forts.)

Bestrafung lange laufender Prozesse und Bewahrung lange wartender 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** zum Zeitpunkt der **Prozesserzeugung**  $\rightsquigarrow$  Laufzeitkonstante

- ▶ wird im weiteren Verlauf nicht mehr verändert
- ▶ erzwingt eine deterministische Ordnung zw. Prozessen

**dynamisch** zum Zeitpunkt der **Prozessausführung**  $\rightsquigarrow$  Laufzeitvariable

- ▶ die Berechnung erfolgt durch das Betriebssystem
  - ▶ ggf. in Kooperation mit den Anwendungsprogrammen
- ▶ erzwingt keine deterministische Ordnung zw. Prozessen

**Echtzeitverarbeitung** bedingt Prioritäten setzende Verfahren

- ▶ jedoch nicht jedes solcher Verfahren eignet sich zum Echtzeitbetrieb
- ▶ Einplanung muss ein **deterministisches Laufzeitverhalten** liefern
  - ▶ entsprechend der jeweiligen Anforderungen der Anwendungsdomäne

# Gegenüberstellung von Strategien und Verfahrensweisen

kooperativ/verdrängend vs. probabilistisch/deterministisch

	FCFS	RR	VRR	SPN	SRTF	HRRN	FB
kooperativ	✓			✓			
verdrängend		✓	✓		✓	✓	✓
probabilistisch				✓	✓	✓	
deterministisch							

keine bzw. nicht von sich aus allein

MLQ umfasst die Eigenschaften der in dem Verfahren vereinten Strategien

- ▶ Priorisierung von Strategien liefert Nuancen im Laufzeitverhalten
- ▶ speziellen Anwendungsanforderungen (teilweise) entgegenkommen:
  - ▶ z.B. FCFS priorisieren  $\leadsto$  „*number crunching*“ fördern

## UNIX klassisch

Zweistufiges Verfahren, Antwortzeiten minimierend, Interaktivität fördernd

*low-level* kurzfristig; präemptiv, MLFQ, **dynamische Prozessprioritäten**

- ▶ einmal pro Sekunde:  $prio = cpu\_usage + p\_nice + base$
- ▶ CPU-Nutzungsrecht mit jedem „Tick“ (1/10 s) verringert
  - ▶ Prioritätswert kontinuierlich um „Tickstand“ erhöhen
  - ▶ je höher der Wert, desto niedriger die Priorität
- ▶ über die Zeit gedämpftes CPU-Nutzungsmaß: *cpu\_usage*
  - ▶ der Dämpfungsfaktor variiert von UNIX zu UNIX

*high-level* mittelfristig; mit **Umlagerung** arbeitend


Prozesse können relativ zügig den Betriebssystemkern verlassen

- ▶ gesteuert über die beim Schlafenlegen einstellbare **Aufweckpriorität**

## UNIX 4.3 BSD

MLFQ (32 Warteschlangen, RR), dynamische Prioritäten (0–127)

**Berechnung** der **Benutzerpriorität** bei jedem vierten Tick (40 ms)

- ▶  $p\_usrpri = PUSER + \left\lceil \frac{p\_cpu}{4} \right\rceil + 2 \cdot p\_nice$ 
  - ▶ mit  $p\_cpu = p\_cpu + 1$  bei jedem Tick (10 ms)
  - ▶ **Gewichtungsfaktor**  $-20 \leq p\_nice \leq 20$   nice(2)
- ▶ Prozess mit Priorität  $P$  kommt in Warteschlange  $P/4$

**Glättung** des Wertes der **Prozessornutzung** ( $p\_cpu$ ) jede Sekunde

- ▶  $p\_cpu = \frac{2 \cdot load}{2 \cdot load + 1} \cdot p\_cpu + p\_nice$
- ▶ **Sonderfall**: Prozesse schließen länger als eine Sekunde
  - ▶  $p\_cpu = \left[ \frac{2 \cdot load}{2 \cdot load + 1} \right]^{p\_slptime} \cdot p\_cpu$


## UNIX 4.3 BSD (Forts.)

Glättung durch Dämpfungsfiter (engl. *decay filter*)

**Annahme 1**: mittlere Auslastung ( $load$ ) sei 1:  $p\_cpu = 0.66 \cdot p\_cpu + p\_nice$

**Annahme 2**: Prozess sammelt  $T_i$  Ticks im Zeitintervall  $i$  an,  $p\_nice = 0$ :

$$\begin{aligned}
 p\_cpu &= 0.66 \cdot T_0 \\
 &= 0.66 \cdot (T_1 + 0.66 \cdot T_0) = 0.66 \cdot T_1 + 0.44 \cdot T_0 \\
 &= 0.66 \cdot T_2 + 0.44 \cdot T_1 + 0.30 \cdot T_0 \\
 &= 0.66 \cdot T_3 + \dots + 0.20 \cdot T_0 \\
 &= 0.66 \cdot T_4 + \dots + 0.13 \cdot T_0
 \end{aligned}$$

 nach fünf Sekunden gehen nur noch etwa 13% der „Altlast“ ein



# UNIX Solaris

MLQ (4 Klassen) und MLFQ (60 Ebenen, Tabellensteuerung)

quantum	tqexp	slprt	maxwait	lwait	Ebene
200	0	50	0	50	0
200	0	50	0	50	1
...					
40	34	55	0	55	44
40	35	56	0	56	45
40	36	57	0	57	46
40	37	58	0	58	47
40	38	58	0	58	48
40	39	58	0	59	49
40	40	58	0	59	50
40	41	58	0	59	51
40	42	58	0	59	52
40	43	58	0	59	53
40	44	58	0	59	54
40	45	58	0	59	55
40	46	58	0	59	56
40	47	58	0	59	57
40	48	58	0	59	58
20	49	59	32000	59	59

/usr/sbin/dispatchadmin -c TS -g

MLQ (Klasse)		Priorität
time-sharing	TS	0-59
interactive	IA	0-59
system	SYS	60-99
real time	RT	100-109

MLFQ in Klasse TS bzw. IA:

- quantum Zeitscheibe (ms)
- tqexp Ebene bei Bestrafung
- slprt Ebene nach Deblokkierung
- maxwait ohne Bedienung (s)
- lwait Ebene bei Bewährung

**Besonderheit:** dispatch table (TS, IA) kapselt sämtliche Entscheidungen

- ▶ kunden-/problemspezifische Lösungen durch verschiedene Tabellen

# UNIX Solaris (Forts.)

Bestrafung vs. Bewährung nach Verdrängung

Beispiel:

- ▶ 1 × CPU-Stoß à 1000 ms
- ▶ 5 × E/A-Stoß → CPU-Stoß à 1 ms

#	Ebene	CPU-Stoß	Ereignis
1	59	20	Zeitscheibe
2	49	40	Zeitscheibe
3	39	80	Zeitscheibe
4	29	120	Zeitscheibe
5	19	160	Zeitscheibe
6	9	200	Zeitscheibe
7	0	200	Zeitscheibe
8	0	180	E/A-Stoß
9	50	1	E/A-Stoß
10	58	1	E/A-Stoß
11	58	1	E/A-Stoß
12	58	1	E/A-Stoß

Variante: nach 640 ms...

- ▶ der Prozess wird verdrängt und muss auf die erneute Einlastung warten
- ▶ der Alterung des wartenden Prozesses wird durch Anhebung seiner Priorität entgegengewirkt (*anti-aging*)
- ▶ die höhere Ebene erreicht, steigt der Prozess im weiteren Verlauf wieder ab


...			
7	0	20	<i>anti-aging</i>
8	50	40	Zeitscheibe
9	40	40	Zeitscheibe
10	30	80	Zeitscheibe
11	20	120	Zeitscheibe
12	10	80	E/A-Stoß
13	50	1	E/A-Stoß
...			

## Linux 2.4

## Epochen und Zeitquanten

Prozessen zugewiesene Prozessorzeit ist in **Epochen** unterteilt  
**beginnen** alle lauffähige Prozess haben ihr Zeitquantum erhalten  
**enden** alle lauffähigen Prozesse haben ihr Zeitquantum verbraucht

**Zeitquanten** (Zeitscheiben) variieren mit den Prozessen und Epochen


- ▶ jeder Prozess besitzt eine einstellbare **Zeitquantumbasis**  `nice(2)`
  - ▶ 20 Ticks  $\approx$  210 ms
  - ▶ das Zeitquantum eines Prozesses nimmt periodisch (Tick) ab
- ▶ beide Werte addiert liefert die **dynamische Priorität** eines Prozesses
  - ▶ dynamische Anpassung:  $quantum = quantum/2 + (20 - nice)/4 + 1$

**Echtzeitprozesse** (schwache EZ) besitzen **statische Prioritäten**: 1–99


## Linux 2.4 (Forts.)

## Einplanungsklassen und Gütefunktion

Prozesseinplanung unterscheidet zwischen drei **Scheduling-Klassen**:

<b>FIFO</b>	verdrängbare, kooperative Echtzeitprozesse	}	 <b>eine Bereitliste</b>
<b>RR</b>	Echtzeitprozesse derselben Priorität		
<b>other</b>	konventionelle („ <i>time-shared</i> “) Prozesse		

Prozessauswahl greift auf eine **Gütefunktion** zurück:

$v = -1000$	der Prozess ist <i>Init</i>	 $O(n)$	–
$v = 0$	der Prozess hat sein Zeitquantum verbraucht		–
$0 < v < 1000$	der Prozess hat sein Zeitquantum nicht verbraucht		+
$v \geq 1000$	der Prozess ist ein Echtzeitprozess		++

Prozesse können bei der Auswahl einen **Bonus** („*boost*“) erhalten

- ▶ sofern sie sich mit dem Vorgänger den Adressraum teilen

## Linux 2.5

Deterministische Prozesseinplanung:  $O(1)$

Einplanung von Prozessen hat **konstante Berechnungskomplexität**:

**Prioritätsfelder** zwei Tabellen pro CPU: *active*, *expired*

**Prioritätsebenen** 140 Ebenen pro Tabelle

- ▶ 1–100 für Echtzeit-, 101–140 für sonstige Prozesse
- ▶ pro Ebene eine (doppelt verkettete) Bereitliste

**Prioritäten** gewöhnlicher Prozesse skalieren je nach Grad der Interaktivität

- ▶ **Bonus** (−5) für interaktive Prozesse, **Strafe** (+5) für rechenintensive
- ▶ berechnet am Zeitscheibenende:  $prio = MAX\_RT\_PRIO + nice + 20$

Ablauf des Zeitquantums befördert den aktiven Prozess ins „*expired*“-Feld

- ▶ zum Epochenwechsel werden die Tabellen ausgetauscht
  - ▶ `void *aux = active; active = expired; expired = aux;`

## Einplanung $\rightsquigarrow$ Einlastungsreihenfolge von Prozessen

Zuteilung von Betriebsmitteln an konkurrierende Prozesse

- ▶ Betriebssysteme treffen **Zuteilungsentscheidungen** auf drei Ebenen:
  - long-term scheduling* Lastkontrolle des Systems
  - medium-term scheduling* Umlagerung von Programmen
  - short-term scheduling* Einlastungsreihenfolge von Prozessen
- ▶ die Entscheidungskriterien haben verschiedene Dimensionen:
  - Benutzer** Antwort-/Durchlaufzeit, Termine, Vorhersagbarkeit
  - System** Durchsatz, Auslastung, Gerechtigkeit, Dringlichkeit, Lastausgleich
- ▶ Prozesseinplanung kennt z.T. sehr unterschiedlich **Verfahrensweisen**
  - ▶ kooperativ/verdrängend, deterministisch/probabilistisch
  - ▶ entkoppelt/gekoppelt, asymmetrisch/symmetrisch
- ▶ Dimension, Kriterium und Verfahrensweise  $\rightsquigarrow$  **Einplanungsstrategien**
  - ▶ FCFS, RR, VRR, SPN, SRTF, HRRN, MLQ, FB (MLFQ)