Technische Universität Dresden

Fakultät Informatik Institut für Systemarchitektur, Professur für Betriebssysteme

BETRIEBSSYSTEME

Scheduling

https://tud.de/inf/os/studium/vorlesungen/bs

HORST SCHIRMEIER

Inhalt

- Wiederholung
- Abfertigungszustände und Zustandsübergänge
- Klassische CPU-Zuteilungsstrategien
  - FCFS..... einfach
  - RR, VRR..... zeitscheibenbasiert
  - SPN (SJF), SRTF, HRRN.....vorhersagebasiert
  - MLFQ.....prioritätenbasiert
- Bewertungskriterien und Vergleich
- Beispiele
  - UNIX (4.3BSD)
  - NT

Silberschatz, Kap. …
5: Process Scheduling

Tanenbaum, Kap. …
2.5: Scheduling

Es geht um Uniprozessor-Scheduling für den Allgemeinzweckbetrieb. Nicht betrachtet wird:

- **Multiprozessor-Scheduling**
- **Echtzeit-Scheduling**
- **E/A-Scheduling**

Inhalt

- **Wiederholung**
- Abfertigungszustände und Zustandsübergänge
- Klassische CPU-Zuteilungsstrategien
  - FCFS..... einfach
  - RR, VRR..... zeitscheibenbasiert
  - SPN (SJF), SRTF, HRRN.....vorhersagebasiert
  - MLFQ.....prioritätenbasiert
- Bewertungskriterien und Vergleich
- Beispiele
  - UNIX (4.3BSD)
  - NT

Wiederholung

- **Prozesse:** die zentrale Abstraktion für Aktivitäten in heutigen Betriebssystemen
  - konzeptionell: unabhängige sequentielle Kontrollflüsse (Folge von CPU- und E/A-Stößen)
  - Tatsächlich findet ein ***Multiplexing*** der CPU statt.
- UNIX-Systeme stellen diverse *System Calls* zur Verfügung, um Prozesse zu erzeugen, zu verwalten und miteinander zu verknüpfen.
  - Moderne Betriebssysteme unterstützen darüber hinaus auch leicht- und federgewichtige Prozesse.
- Prozesse unterliegen der Kontrolle des Betriebssystems:
  - Betriebsmittel-Zuteilung
  - Betriebsmittel-Entzug

Inhalt

- Wiederholung
- **Abfertigungszustände und Zustandsübergänge**
- Klassische CPU-Zuteilungsstrategien
  - FCFS..... einfach
  - RR, VRR..... zeitscheibenbasiert
  - SPN (SJF), SRTF, HRRN.....vorhersagebasiert
  - MLFQ.....prioritätenbasiert
- Bewertungskriterien und Vergleich
- Beispiele
  - UNIX (4.3BSD)
  - NT

Abfertigungszustand

- Jedem Prozess ist in Abhängigkeit von der Einplanungsebene ein logischer Zustand zugeordnet, der den Abfertigungszustand zu einem Zeitpunkt angibt:
- **kurzfristig** (*short-term scheduling*)
  - bereit, laufend, blockiert
- **mittelfristig** (*medium-term scheduling*)
  - ausgelagert bereit, ausgelagert blockiert
- **langfristig** (*long-term scheduling*)
  - erzeugt, beendet

Kurzfristige Einplanung

- **bereit** (*READY*)
  zur Ausführung durch den Prozessor (die CPU)
  - Prozess ist auf der Warteliste für die CPU-Zuteilung (*ready list*)
  - Listenposition bestimmt sich durch das Einplanungsverfahren
- **laufend** (*RUNNING*)
  Zuteilung des Betriebsmittels „CPU“ ist erfolgt
  - Prozess führt Berechnungen durch, er vollzieht seinen CPU-Stoß
  - Für jeden Prozessor gibt es zu einem Zeitpunkt nur einen laufenden Prozess.
- **blockiert** (*BLOCKED*)
  auf ein bestimmtes Ereignis
  - Prozess führt „Ein-/Ausgabe“ durch, er vollzieht seinen E/A-Stoß
  - Er erwartet die Erfüllung mindestens einer Bedingung.

Mittelfristige Einplanung

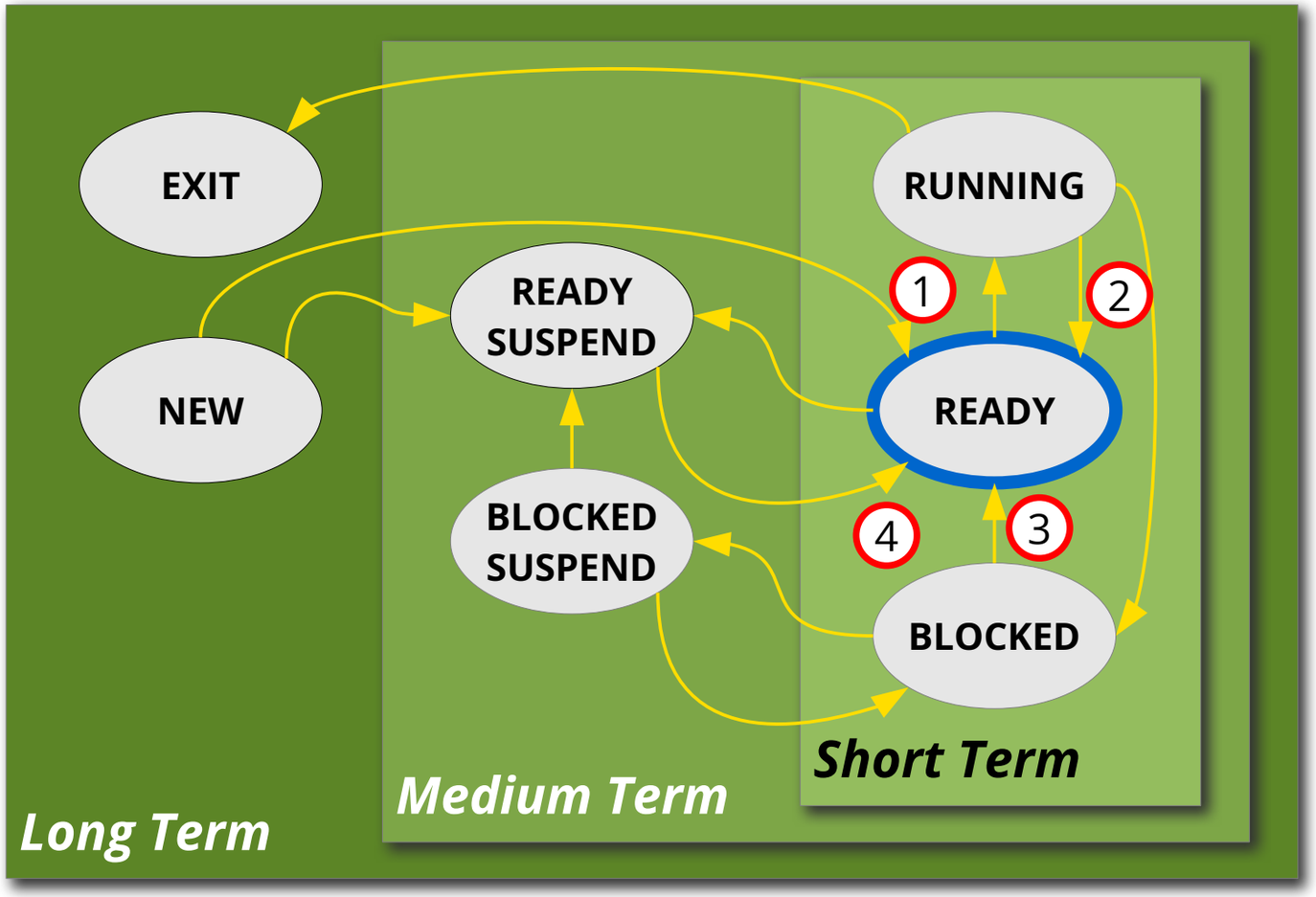
Prozess ist **komplett ausgelagert**, d.h. der Inhalt seines gesamten Adressraums wurde in den Hintergrundspeicher verschoben (*swap-out*) und der von dem Prozess belegte Vordergrundspeicher wurde freigegeben. Die Einlagerung (*swap-in*) des Adressraums ist abzuwarten:

- **ausgelagert bereit** (*READY SUSPEND*)
  - CPU-Zuteilung („bereit“) ist außer Kraft gesetzt
  - Prozess ist auf der Warteliste für die Speicherzuteilung
- **ausgelagert blockiert** (*BLOCKED SUSPEND*)
  - Prozess erwartet weiterhin ein Ereignis („blockiert“)
  - Tritt das Ereignis ein, wird der Prozess „ausgelagert bereit“.

Langfristige Einplanung

- **erzeugt** (*NEW*)
  und fertig zur Programmverarbeitung – **fork** (2)
  - Prozess ist instanziiert, ihm wurde ein Programm zugeordnet
  - Ggf. steht die Zuteilung des Betriebsmittels „Speicher“ noch aus.
- **beendet** (*EXIT*)
  und erwartet die Entsorgung – **exit** (2) / **wait** (2)
  - Prozess ist terminiert, seine Betriebsmittel werden freigegeben
  - Ggf. muss ein anderer Prozess den „Kehraus“ vollenden (wie z.B. unter UNIX).

Zustandsübergänge



Einplanungs- und Auswahlzeitpunkt

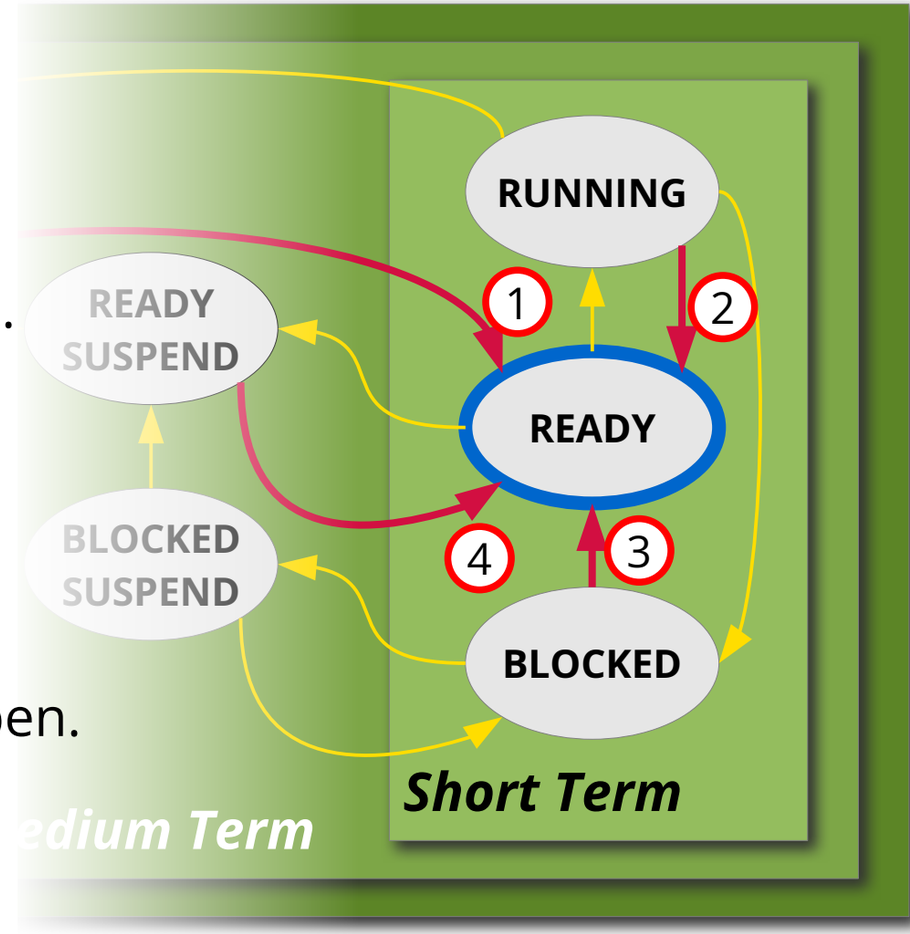
- Jeder Übergang in den Zustand **bereit** (*READY*) aktualisiert die CPU-Warteschlange:
  - Entscheidung über die Einreihung des Prozesskontrollblocks
  - Ergebnis hängt von **CPU-Zuteilungsstrategie** des Systems ab
- Einplanung/Umplanung (*scheduling/rescheduling*) erfolgt, …

  ① nachdem ein Prozess erzeugt worden ist

  ② wenn ein Prozess freiwillig die Kontrolle über die CPU abgibt

  ③ sofern das von einem Prozess erwartete Ereignis eingetreten ist

  ④ sobald ein ausgelagerter Prozess wieder aufgenommen wird
- Ein Prozess kann dazu gedrängt werden, die CPU abzugeben.
  → ***präemptives Scheduling***, auch ②
  - z.B. durch Zeitgeberunterbrechung



Inhalt

- Wiederholung
- Abfertigungszustände und Zustandsübergänge
- **Klassische CPU-Zuteilungsstrategien**
  - FCFS..... einfach
  - RR, VRR..... zeitscheibenbasiert
  - SPN (SJF), SRTF, HRRN.....vorhersagebasiert
  - MLFQ.....prioritätenbasiert
- Bewertungskriterien und Vergleich
- Beispiele
  - UNIX (4.3BSD)
  - NT

First-Come First-Served – FCFS

- Ein einfaches und gerechtes(?) Verfahren: „Wer zuerst kommt, mahlt zuerst.“
  - Einreihungskriterium ist die **Ankunftszeit** eines Prozesses
  - arbeitet nicht-verdrängend und setzt **kooperative Prozesse** voraus
- Beispiel:

AnimOS-Simulation

Prozess	Zeiten					
	Ankunft	**Bedienzeit T_s**	**Start**	**Ende**	**Durchlauf T_r**	**T_r/T_s**
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
Mittelwert					100	26.00

  - Sehr hohe normalisierte Durchlaufzeit (T_r/T_s): Durchlaufzeit von C steht in einem sehr schlechten Verhältnis zur Bedienzeit T_s.

Diskussion: FCFS – „Konvoi-Effekt“

- Problem für kurz laufende E/A-lastige Prozesse, die langen CPU-lastigen Prozessen folgen:
  - Prozesse mit **langen** CPU-Stößen werden **begünstigt**,
  - Prozesse mit **kurzen** CPU-Stößen werden **benachteiligt**.
- FCFS: Minimiert Anzahl der Kontextwechsel, Konvoi-Effekt verursacht …
  - hohe Antwortzeit
  - niedriger E/A-Durchsatz
- FCFS ungeeignet bei Mix von CPU- und E/A-lastigen Prozessen
  - Einsatz typischerweise nur in reinen Stapelverarbeitungssystemen

Round Robin – RR

- Verringert die bei FCFS auftretende Benachteiligung kurzer CPU-Stöße: „Jeder gegen jeden“
  - Die Prozessorzeit wird in **Zeitscheiben** aufgeteilt (***time slicing***).
- Mit Ablauf der Zeitscheibe erfolgt ggf. ein Prozesswechsel:
  - Der unterbrochene Prozess wird ans Ende der Bereitliste verdrängt,
  - der nächste Prozess wird gemäß FCFS der Bereitliste entnommen.
- Basis für CPU-Schutz: Zeitgeber bewirkt **Unterbrechung** am Ende der Zeitscheibe
- **Zeitscheibenlänge** bestimmt maßgeblich Effektivität des Verfahrens
  - zu lang, Degenerierung zu FCFS; zu kurz, sehr hoher Overhead
  - **Faustregel:** etwas länger als die Dauer einer „typischen Interaktion“

Diskussion: RR – Leistungsprobleme

- **E/A-lastige Prozesse** beenden ihren CPU-Stoß innerhalb ihrer Zeitscheibe
  - blockieren und kommen mit Ende ihres E/A-Stoßes in die Bereitliste
- **CPU-lastige Prozesse** schöpfen dagegen ihre Zeitscheibe voll aus
  - werden verdrängt und kommen sofort wieder in die Bereitliste
- Die CPU-Zeit ist zu Gunsten CPU-lastiger Prozesse ungleich verteilt!
  - E/A-lastige Prozesse werden schlecht bedient und dadurch Geräte schlecht ausgelastet,
  - Varianz der Antwortzeit E/A-lastiger Prozesse erhöht sich.

AnimOS-Simulation: FCFS vs. RR

Virtual Round Robin – VRR

- Vermeidet die bei RR mögliche ungleiche Verteilung der CPU-Zeiten
  - Prozesse kommen mit Ende ihrer E/A-Stöße in eine **Vorzugsliste**
  - Diese Liste wird **vor** der Bereitliste abgearbeitet.
- VRR arbeitet mit Zeitscheiben unterschiedlicher Längen
  - Prozesse der Vorzugsliste bekommen keine volle Zeitscheibe zugeteilt:
    Ihnen wird die **Restlaufzeit** ihrer vorher nicht voll genutzten Zeit gewährt.
  - Sollte ihr CPU-Stoß länger dauern, werden sie in die Bereitliste verdrängt.
- Prozessabfertigung ist dadurch im Vergleich zu RR etwas aufwändiger.

AnimOS-Simulation: FCFS vs. RR vs. VRR

Shortest Process Next – SPN

- Verringert die bei FCFS auftretende Benachteiligung kurzer CPU-Stöße:
  „Die Kleinen nach vorne“
  - Grundlage dafür ist die **Kenntnis über die Prozesslaufzeiten**
  - Verdrängung findet nicht statt
- **Hauptproblem:** Vorhersage der Laufzeiten
  - Stapelbetrieb: Programmierer geben das erforderliche *time limit** vor
  - Dialogbetrieb: Schätzung aus früheren Stoßlängen des Prozesses
- Antwortzeiten werden wesentlich verkürzt und die Gesamtleistung steigt.
  - Dafür: Gefahr der **Aushungerung** (***starvation***) CPU-lastiger Prozesse

*Die Zeitdauer, innerhalb der der Job (wahrscheinlich/hoffentlich) beendet wird, bevor er abgebrochen wird.

Diskussion: SPN – CPU-Stoßdauer

- Basis für die **Schätzung** ist die Mittelwertbildung über alle bisherigen 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**: gleiche **Gewichtung** aller CPU-Stöße
  - **Jüngere CPU-Stöße** sind jedoch von größerer Bedeutung als ältere und sollten daher auch mit größerer Gewichtung berücksichtigt werden!
- Ursache ist das **Lokalitätsprinzip**

Diskussion: SPN – Stoßgewichtung

- Die weiter zurückliegenden CPU-Stöße sollen weniger Gewicht erhalten:

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

  - Für den konstanten Gewichtungsfaktor α gilt dabei: $0 < \alpha < 1$
  - Er drückt die relative Gewichtung einzelner CPU-Stöße der Zeitreihe aus.

- Rekursive Einsetzung führt zu …

$$S_{n+1} = \alpha T_n + (1 - \alpha)\alpha T_{n-1} + \cdots + (1 - \alpha)^i \alpha T_{n-i} + \cdots + (1 - \alpha)^n S_1$$
$$S_{n+1} = \alpha \cdot \sum_{i=0}^{n-1} (1 - \alpha)^i T_{n-i} + (1 - \alpha)^n S_1$$

- für α = 0.8:

$$S_{n+1} = 0{,}8T_n + 0{,}16T_{n-1} + 0{,}032T_{n-2} + 0{,}0064T_{n-3} + \ldots$$

Dieses statistische Verfahren nennt man auch exponentielle Glättung.

Shortest Remaining Time First – SRTF

- Erweitert den SPN-Ansatz um **Verdrängung**.
  - Dadurch geeignet für den Dialogbetrieb
  - Führt zu besseren Durchlaufzeiten
- Der laufende Prozess wird verdrängt, wenn gilt: $T_{erw} < T_{rest}$
  - T_{erw} ist die erwartete CPU-Stoßlänge eines eintreffenden Prozesses
    T_{rest} ist die verbleibende CPU-Stoßlänge des laufenden Prozesses
- Anders als RR basiert SRTF **nicht auf Zeitgeberunterbrechungen**, ist aber **präemptiv**.
  - Dafür müssen allerdings Stoßlängen abgeschätzt werden.
- Wie SPN kann auch SRTF Prozesse zum „Verhungern“ bringen.

Highest Response Ratio Next – HRRN

- Vermeidet das bei SRTF mögliche Verhungern von CPU-lastigen Prozessen.
  - Das Altern (*aging*), d.h. die Wartezeit von Prozessen, wird berücksichtigt:

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

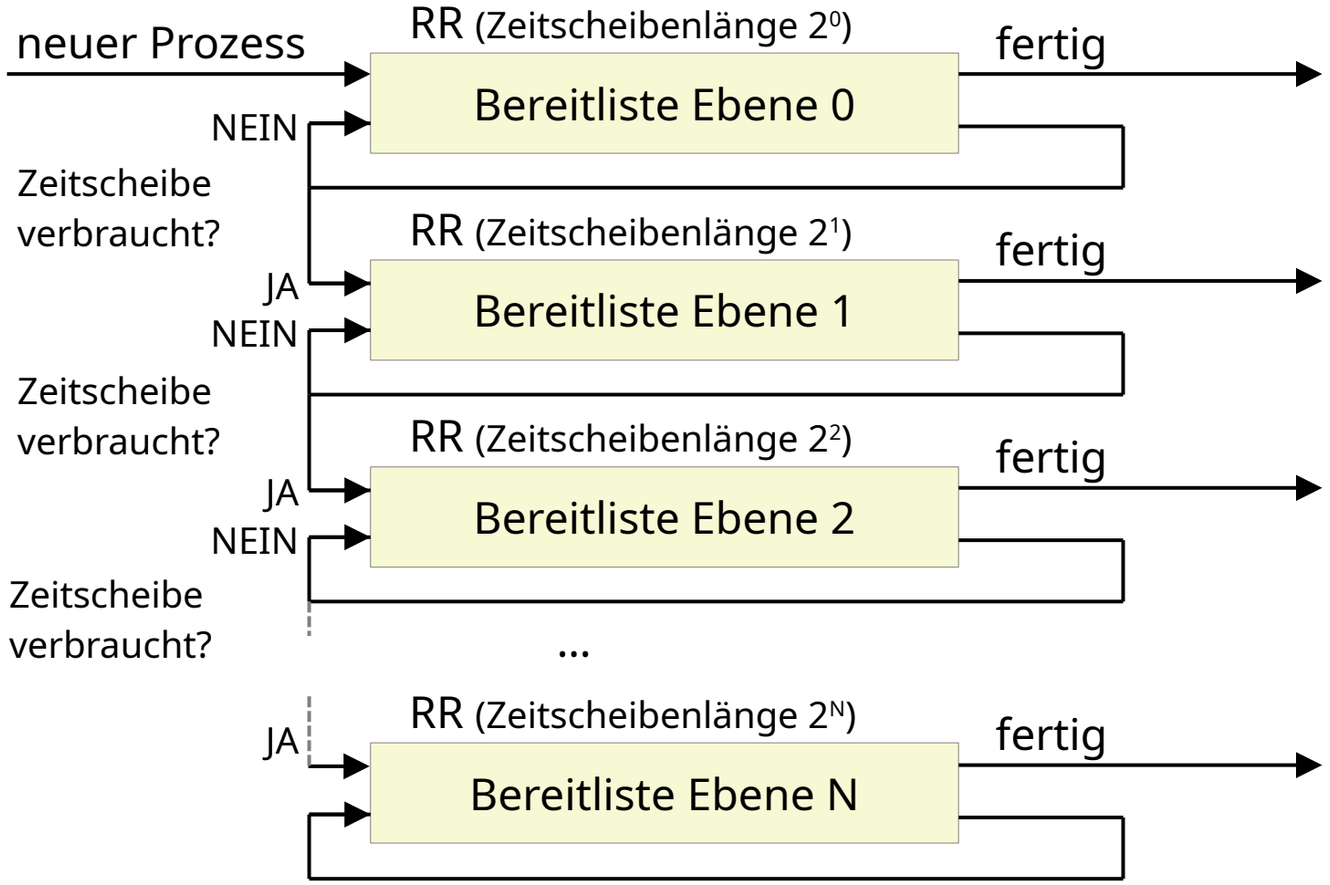
  - w ist die bisherige „Wartezeit des Prozesses“
  - s ist die „erwartete Bedienzeit“

- Ausgewählt wird immer der Prozess mit dem **größten Verhältniswert R**.

Multi-Level Feedback Queue – MLFQ

- Begünstigt kurze Prozesse, ohne die relativen Längen der Prozesse abschätzen zu müssen.
  - Grundlage ist die „Bestrafung“ (*penalization*) von „Langläufern“
  - Prozesse unterliegen dem Verdrängungsprinzip
- Mehrere Bereitlisten, je nach Anzahl von Prioritätsebenen:
  - Prozesse werden **strikt nach Priorität** eingeplant, und nach RR, wenn sie dieselbe Priorität haben
  - Prozess läuft zunächst auf **höchster** Prioritätsebene i=0
  - RR mit kurzen Zeitscheiben für hohe Prioritäten (kleine i), langen Zeitscheiben für niedrige Prioritäten (große i), ohne Zurücksetzen der Zeitmessung bei E/A
  - **Ablauf der Zeitscheibe:** Prozess kommt in Ebene i+1
  - **Boost:** in fixen Intervallen **Anheben aller Prozesse** auf die höchste Prioritätsebene (Fortschrittsgarantie, Berücksichtigung von verändertem Verhalten)

MLFQ – *Scheduling*-Modell



Diskussion: Prioritäten

- Prozess-„Vorrang“, der Zuteilungsentscheidungen maßgeblich beeinflusst
- **Statische Prioritäten** werden zum Zeitpunkt der Prozesserzeugung festgelegt:
  - Wert kann im weiteren Verlauf nicht mehr verändert werden
  - erzwingen **deterministische Ordnung** zwischen Prozessen
- **Dynamische Prioritäten** werden während der Prozesslaufzeit aktualisiert:
  - Aktualisierung erfolgt im Betriebssystem, aber auch vom Benutzer aus
  - SPN, SRTF, HRRN und FB sind Spezialfälle dieses Verfahrens

Inhalt

- Wiederholung
- Abfertigungszustände und Zustandsübergänge
- Klassische CPU-Zuteilungsstrategien
  - FCFS..... einfach
  - RR, VRR..... zeitscheibenbasiert
  - SPN (SJF), SRTF, HRRN.....vorhersagebasiert
  - MLFQ.....prioritätenbasiert
- **Bewertungskriterien und Vergleich**
- Beispiele
  - UNIX (4.3BSD)
  - NT

Ziele = Bewertungskriterien

- Benutzerorientiert
  - **Durchlaufzeit** (auch: Verweilzeit) – Zeit zwischen Eingang und Abschluss eines Prozesses einschließlich der Wartezeit(en) → **Stapelverarbeitung**
  - **Antwortzeit** – Zeit zwischen Benutzereingabe und Antwort → **interaktive Systeme**
  - **Termineinhaltung** – Für die Interaktion mit äußeren physikalischen Prozessen sollten Termine eingehalten werden. → **Echtzeitsysteme**
  - **Vorhersagbarkeit** – Prozesse werden unabhängig von der Last immer gleich bearbeitet. → **harte Echtzeitsysteme**
- Systemorientiert
  - **Durchsatz** – Möglichst viele Prozesse pro Zeiteinheit abarbeiten
  - **CPU-Auslastung** – CPU sollte möglichst immer beschäftigt sein. *Overhead* (*Scheduling*-Entscheidung, Kontextwechsel) vermeiden.
  - **Fairness** – Kein Prozess soll benachteiligt werden (z.B. „Aushungern“).
  - **Lastausgleich** – Auch E/A-Geräte sollen gleichmäßig ausgelastet werden.

Gegenüberstellung – quantitativ

	Prozess	A	B	C	D	E	Mittel-wert
	Start	0	2	4	6	8	
	Bedienzeit T_s	3	6	4	5	2	
FCFS	Ende	3	9	13	18	20	
	Durchlauf T_r	3	7	9	12	12	8.60
	T_r/T_s	1.00	1.17	2.25	2.40	6.00	2.56
RR	Ende	4	18	17	20	15	
q=1	Durchlauf T_r	4	16	13	14	7	10.80
	T_r/T_s	1.33	2.67	3.25	2.80	3.50	2.71
SPN	Ende	3	9	15	20	11	
	Durchlauf T_r	3	7	11	14	3	7.60
	T_r/T_s	1.00	1.17	2.75	2.80	1.50	1.84
SRTF	Ende	3	15	8	20	10	
	Durchlauf T_r	3	13	4	14	2	**7.20**
	T_r/T_s	1.00	2.17	1.00	2.80	1.00	**1.59**
HRRN	Ende	3	9	13	20	15	
	Durchlauf T_r	3	7	9	14	7	8.00
	T_r/T_s	1.00	1.17	2.25	2.80	3.50	2.14

Nach William Stallings, „Betriebssysteme – Prinzipien und Umsetzung“

Gegenüberstellung – qualitativ

Strategie	präemptiv/ kooperativ	Vorhersage nötig	Implementierungs-aufwand	Verhungern möglich	Auswirkung auf Prozesse
FCFS	kooperativ	nein	minimal	nein	Konvoi-Effekt
RR	präemptiv (Zeitgeber)	nein	klein	nein	fair, aber be-nachteiligt E/A-lastige Prozesse
SPN	kooperativ	ja	groß	ja	benachteiligt CPU-lastige Prozesse
SRTF	präemptiv (bei Eingang)	ja	größer	ja	benachteiligt CPU-lastige Prozesse
HRRN	kooperativ	ja	groß	nein	gute Lastverteilung
MLFQ	präemptiv (Zeitgeber)	nein	größer	nein	bevorzugt u.U. E/A-lastige Prozesse

In Anlehnung an William Stallings, „Betriebssysteme – Prinzipien und Umsetzung“

Inhalt

- Wiederholung
- Abfertigungszustände und Zustandsübergänge
- Klassische CPU-Zuteilungsstrategien
  - FCFS..... einfach
  - RR, VRR..... zeitscheibenbasiert
  - SPN (SJF), SRTF, HRRN vorhersagebasiert
  - MLFQ..... prioritätenbasiert
- Bewertungskriterien und Vergleich
- **Beispiele**
  - UNIX (4.3BSD)
  - NT

UNIX

- Zweistufiges präemptives Verfahren mit dem Ziel, Antwortzeiten zu minimieren
- Kein Long-Term Scheduling
- ***high-level***: mittelfristig
  mit Ein-/Auslagerung (*swapping*) arbeitend
- ***low-level***: kurzfristig
  präemptiv, MLFQ, dynamische Prozessprioritäten
  - Einmal pro Sekunde: $prio = cpu_usage + p_nice + base$
  - Jeder „***Tick***“ (1/10 s) verringert das Nutzungsrecht über die CPU durch Erhöhung von *cpu_usage* beim laufenden Prozess
    - hohe *prio*-Zahl = niedrige Priorität
  - Das Maß der CPU-Nutzung (*cpu_usage*) wird über die Zeit gedämpft.
    - Die Dämpfungs-/Glättungsfunktion variiert von UNIX zu UNIX.

UNIX – 4.3 BSD (1)

- Jeden vierten Tick (40ms) erfolgt die Berechnung der Benutzerpriorität:

$$P_{usrpri} = \min(PUSER + \frac{P_{cpu}}{4} + 2 \cdot P_{nice}, 127)$$

- P_{cpu} nimmt mit jedem Tick um 1 zu und wird einmal pro Sekunde geglättet:

$$P_{cpu} \Leftarrow \frac{2 \cdot load}{2 \cdot load + 1} \cdot P_{cpu} + P_{nice}$$

- Glättung für erwachte Prozesse, die länger als eine Sekunde blockiert waren:

$$P_{cpu} \Leftarrow \left(\frac{2 \cdot load}{2 \cdot load + 1}\right)^{P_{slptime}} \cdot P_{cpu}$$

UNIX – 4.3 BSD (2)

- Glättung (*decay filter*) bei einer angenommenen mittleren Auslastung (*load*) von 1 gilt $P_{cpu} := 0{,}66 \cdot P_{cpu} + P_{nice}$. Ferner sei angenommen, ein Prozess sammelt T_i Ticks im Zeitintervall i an und $P_{nice} = 0$

$$
\begin{aligned}
P_{cpu_1} &= 0{,}66T_0 \\
P_{cpu_2} &= 0{,}66(T_1 + 0{,}66T_0) = 0{,}66T_1 + 0{,}44T_0 \\
P_{cpu_3} &= 0{,}66T_2 + 0{,}44T_1 + 0{,}30T_0 \\
P_{cpu_4} &= 0{,}66T_3 + \cdots + 0{,}20T_0 \\
P_{cpu_5} &= 0{,}66T_4 + \cdots + 0{,}13T_0
\end{aligned}
$$

- Nach 5 Sekunden geht nur noch 13% „alte“ Auslastung ein.

Windows NT – Prioritätsklassen

- Präemptive, prioritäts- und zeitscheibenbasierte Einplanung von ***Threads***
  - Verdrängung erfolgt auch dann, wenn der Thread sich im Kern befindet
    → nicht so bei UNIX & Co.
  - RR bei gleicher Priorität: 0 reserviert, 1–15 variabel, 16–31 Echtzeit
- Die Art des Threads (Vorder-/Hintergrund) bestimmt das Zeitquantum
  → ***Quantum Stretching***
  - **Quantum** (zwischen 6 und 36) vermindert sich mit jedem ***Tick*** (10 bzw. 15ms) um 3 oder um 1, falls der Thread in den Wartezustand geht
  - Die Zeitscheibenlänge variiert mit den Prozessen: 20 – 180ms
    - Vordergrund/Hintergrund, *Server*/*Desktop*-Konfiguration
- Zudem variable Priorität: *process_priority_class* + *relative_thread_priority* + ***boost***

NT – Prioritätsanpassung

- Threadprioritäten werden in bestimmten Situationen dynamisch angehoben:
  Dynamic Boost
  - Abschluss von Ein-/Ausgabe (Festplatten) +1
  - Mausbewegung, Tastatureingabe +6
  - Deblockierung, Betriebsmittelfreigabe (Semaphor, *Event*, *Mutex*) +1
  - Andere Ereignisse (Netzwerk, *Pipe*, . . .) +2
  - Ereignis im Vordergrundprozess +2
- Die *dynamic boosts* werden mit jedem *Tick* wieder verbraucht
- **Fortschrittsgarantie**
  - Verhindert das Aushungern von *Threads*
  - Alle 3–4 s erhalten bis zu 10 „benachteiligte“ Threads für zwei Zeitscheiben die Priorität 15.

Zusammenfassung

- Betriebssysteme treffen CPU-Zuteilungsentscheidungen auf drei Ebenen:
  - ***Long-Term Scheduling:*** Zulassung von Prozessen zum System
  - ***Medium-Term Scheduling:*** Aus- und Einlagerung von Prozessen
  - ***Short-Term Scheduling:*** kurzfristige CPU-Zuteilung
- Alle hier betrachteten Verfahren werden dem *Short-Term Scheduling* zugerechnet.
  - Es gibt diverse benutzer- und systemorientierte Kriterien für die Beurteilung der Eigenschaften eines CPU-Zuteilungsverfahrens.
  - Die Auswahl kommt einer Gratwanderung gleich; das „beste“ Verfahren lässt sich nur nach einer Analyse des typischen Anwendungsprofils und aller Randbedingungen finden.