5. Foliensatz
Betriebssysteme und Rechnernetze

Prof. Dr. Christian Baun

Frankfurt University of Applied Sciences
(1971–2014: Fachhochschule Frankfurt am Main)
Fachbereich Informatik und Ingenieurwissenschaften
christianbaun@fb2.fra-uas.de

Lernziele dieses Foliensatzes

- Am Ende dieses Foliensatzes kennen/verstehen Sie...
  - welche Schritte der **Dispatcher** (Prozessumschalter) beim Prozesswechsel durchführt
  - was **Scheduling** ist
    - wie **präemptives Scheduling** und **nicht-präemptives Scheduling** funktioniert
    - die Arbeitsweise verschiedener **Scheduling-Verfahren**
    - warum **moderne Betriebssysteme** nicht nur ein einziges Scheduling-Verfahren verwenden
    - wie das **Scheduling moderner Betriebssysteme** im Detail funktioniert

Übungsblatt 5 wiederholt die für die Lernziele relevanten Inhalte dieses Foliensatzes

Prozesswechsel – Der Dispatcher (1/2)

- Aufgaben von Multitasking-Betriebssystemen sind u.a.:
  - **Dispatching**: Umschalten des Prozessors bei einem Prozesswechsel
  - **Scheduling**: Festlegen des Zeitpunkts des Prozesswechsels und der Ausführungsreihenfolge der Prozesse
- Der **Dispatcher** (Prozessumschalter) führt die Zustandsübergänge der Prozesse durch

Wir wissen bereits...

- Beim Prozesswechsel entzieht der Dispatcher dem rechnenden Prozess die CPU und teilt sie dem Prozess zu, der in der Warteschlange an erster Stelle steht
- Bei Übergängen zwischen den Zuständen `bereit` und `blockiert` werden vom Dispatcher die entsprechenden Prozesskontrollblöcke aus den Zustandslisten entfernt neu eingefügt
- Übergänge aus oder in den Zustand `rechnend` bedeuten immer einen Wechsel des aktuell rechnenden Prozesses auf der CPU

Beim Prozesswechsel in oder aus dem Zustand `rechnend`, muss der Dispatcher...

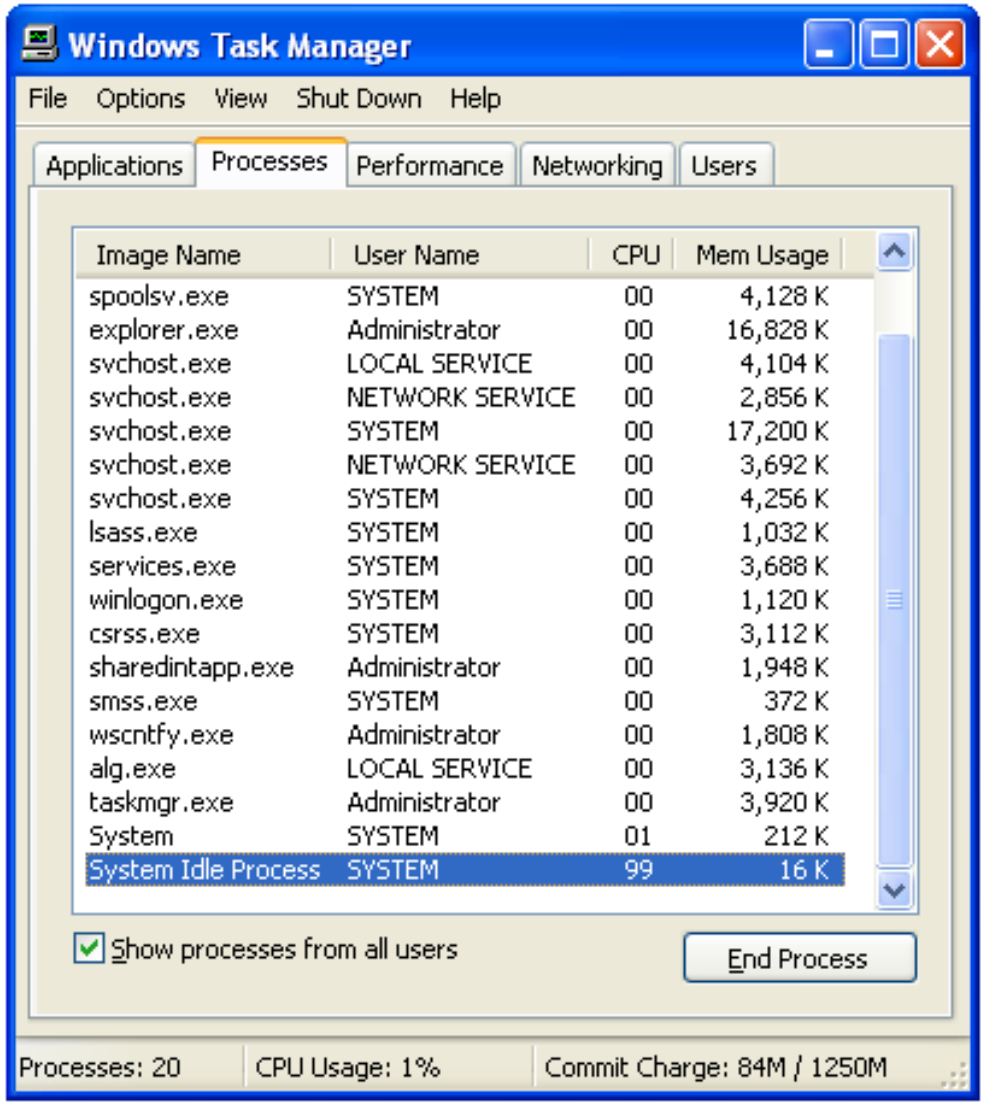
- den Kontext, also die Registerinhalte des aktuell ausgeführten Prozesses im Prozesskontrollblock speichern (retten)
- den Prozessor einem anderen Prozess zuteilen
- den Kontext (Registerinhalte) des jetzt auszuführenden Prozesses aus seinem Prozesskontrollblock wieder herstellen

Prozesswechsel – Der Dispatcher (2/2)

Bildquelle: Wikipedia

Der Leerlaufprozess (*System Idle Process*)

- Bei Windows-Betriebssystemen seit Windows NT erhält die CPU zu jedem Zeitpunkt einen Prozess
- Ist kein Prozess im Zustand `bereit`, kommt der **Leerlaufprozess** zum Zug
- Der Leerlaufprozess ist immer aktiv und hat die niedrigste Priorität
- Durch den Leerlaufprozesses muss der Scheduler nie den Fall berücksichtigen, dass kein aktiver Prozess existiert
- Seit Windows 2000 versetzt der Leerlaufprozess die CPU in einen stromsparenden Modus



Scheduling-Kriterien und Scheduling-Strategien

- Beim Scheduling legt des Betriebssystem die Ausführungsreihenfolge der Prozesse im Zustand `bereit` fest
- **Keine Scheduling-Strategie...**
  - **ist für jedes System optimal geeignet**
  - **kann alle Scheduling-Kriterien optimal berücksichtigen**
    - Scheduling-Kriterien sind u.a. CPU-Auslastung, Antwortzeit (Latenz), Durchlaufzeit (*Turnaround*), Durchsatz, Effizienz, Echtzeitverhalten (Termineinhaltung), Wartezeit, Overhead, Fairness, Berücksichtigen von Prioritäten, Gleichmäßige Ressourcenauslastung...
- Bei der Auswahl einer Scheduling-Strategie muss immer ein **Kompromiss** zwischen den Scheduling-Kriterien gefunden werden

Nicht-präemptives und präemptives Scheduling

- 2 Klassen von Schedulingverfahren existieren:
  1. **Nicht-präemptives Scheduling** bzw. **Kooperatives Scheduling** (nicht-verdrängendes Scheduling)
     - Ein Prozess, der vom Scheduler die CPU zugewiesen bekommen hat, behält die Kontrolle über diese bis zu seiner vollständigen Fertigstellung oder bis er die Kontrolle freiwillig wieder abgibt
     - Problematisch: Ein Prozess kann die CPU so lange belegen wie er will

Beispiele: Windows 3.x und MacOS 8/9

  2. **Präemptives Scheduling** (verdrängendes Scheduling)
     - Einem Prozess kann die CPU vor seiner Fertigstellung entzogen werden
     - Wird einem Prozess die CPU entzogen, pausiert er so lange in seinem aktuellen Zustand, bis der Scheduler ihm erneut die CPU zuteilt
     - Nachteil: Höherer Overhead als nicht-präemptives Scheduling
     - Die Vorteile von präemptivem Scheduling, besonders die Beachtung von Prozessprioritäten, überwiegen die Nachteile

Beispiele: Linux, MacOS X, Windows 95 und neuere Versionen

Einfluss auf die Gesamtleistung eines Computers

- Wie groß der Einfluss des verwendeten Schedulingverfahrens auf die Gesamtleistung eines Computers sein kann, zeigt dieses Beispiel
  - Die Prozesse P_A und P_B sollen nacheinander ausgeführt werden

Prozess	**CPU-Laufzeit**
A	24 ms
B	2 ms

- Läuft ein Prozess mit kurzer Laufzeit vor einem Prozess mit langer Laufzeit, **verschlechtern** sich Laufzeit und Wartezeit des langen Prozesses **wenig**
- Läuft ein Prozess mit langer Laufzeit vor einem Prozess mit kurzer Laufzeit, **verschlechtern** sich Laufzeit und Wartezeit des kurzen Prozesses **stark**

Reihenfolge	**Laufzeit**		**Durchschnittliche Laufzeit**	**Wartezeit**		**Durchschnittliche Wartezeit**
	A	**B**		**A**	**B**	
P_A, P_B	24 ms	26 ms	$\frac{24+26}{2} = 25$ ms	0 ms	24 ms	$\frac{0+24}{2} = 12$ ms
P_B, P_A	26 ms	2 ms	$\frac{2+26}{2} = 14$ ms	2 ms	0 ms	$\frac{0+2}{2} = 1$ ms

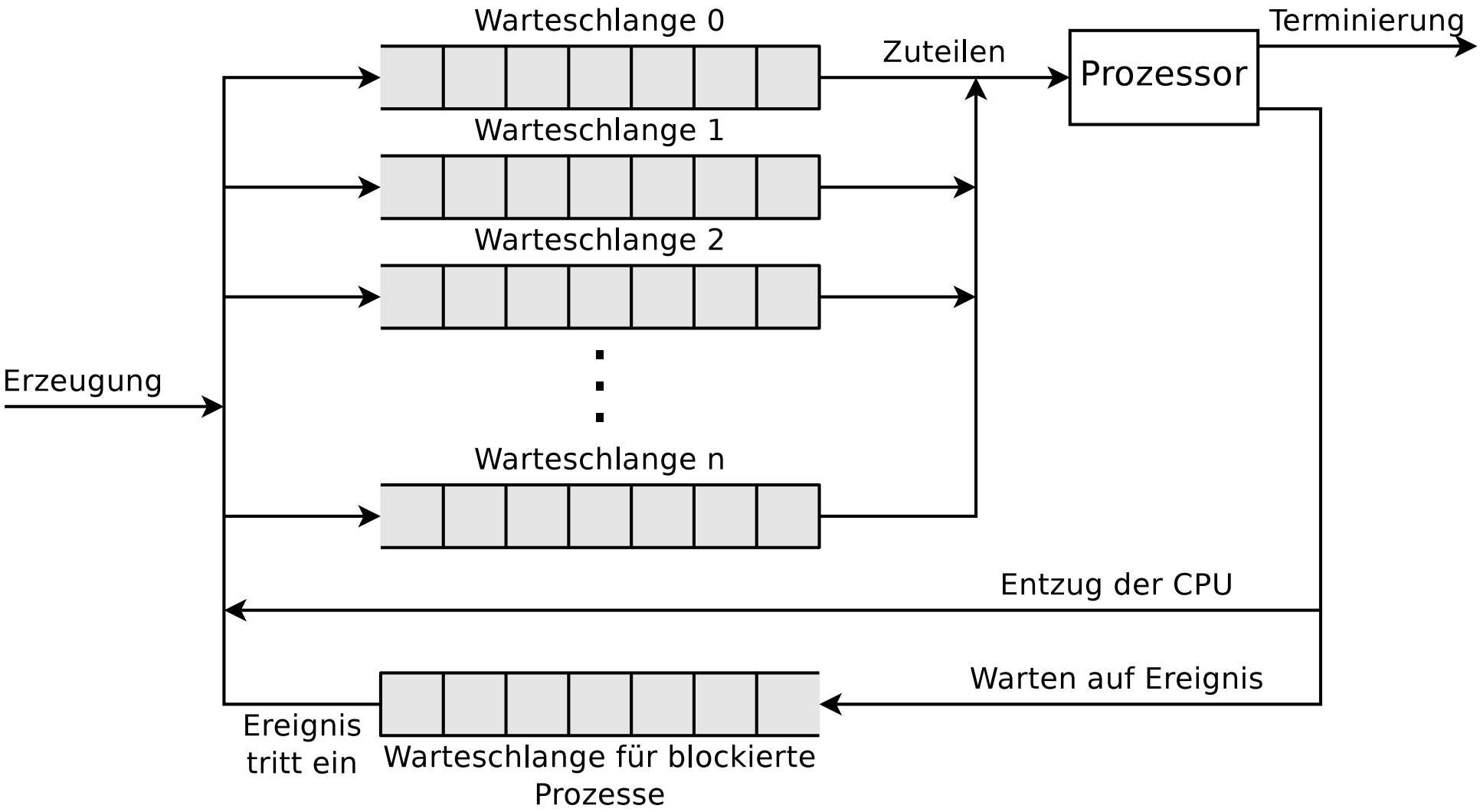
Scheduling-Verfahren

- Zahlreiche Scheduling-Verfahren (Algorithmen) existieren
- Jedes Scheduling-Verfahren versucht unterschiedlich stark, die bekannten Scheduling-Kriterien und -Grundsätze einzuhalten
- Bekannte Scheduling-Verfahren:
  - **Prioritätengesteuertes Scheduling**
  - **First Come First Served** (FCFS) bzw. **First In First Out** (FIFO)
  - ~~**Last Come First Served** (LCFS)~~
  - **Round Robin** (RR) mit Zeitquantum
  - **Shortest Job First** (SJF) ~~und **Longest Job First** (LJF)~~
  - **Shortest Remaining Time First** (SRTF)
  - ~~**Longest Remaining Time First** (LRTF)~~
  - **Highest Response Ratio Next** (HRRN)
  - ~~**Earliest Deadline First** (EDF)~~
  - ~~**Fair-Share-Scheduling**~~
  - ~~**Statisches Multilevel-Scheduling**~~
  - **Multilevel-Feedback-Scheduling**

Prioritätengesteuertes Scheduling

- Prozesse werden nach ihrer Priorität (= Wichtigkeit bzw. Dringlichkeit) abgearbeitet
- Es wird immer dem Prozess im Zustand `bereit` die CPU zugewiesen, der die höchste Priorität hat
  - Die Priorität kann von verschiedenen Kriterien abhängen, z.B. benötigte Ressourcen, Rang des Benutzers, geforderte Echtzeitkriterien, usw.
- Kann **präemptiv** (verdrängend) und **nicht-präemptiv** (nicht-verdrängend) sein
- Die Prioritätenvergabe kann **statisch** oder **dynamisch** sein
  - Statische Prioritäten ändern sich während der gesamten Lebensdauer eines Prozesses nicht und werden häufig in Echtzeitsystemen verwendet
  - Dynamische Prioritäten werden von Zeit zu Zeit angepasst
    $\Longrightarrow$ **Multilevel-Feedback Scheduling** (siehe Folie 22)
- Gefahr beim (statischen) prioritätengesteuertem Scheduling: Prozesse mit niedriger Priorität können verhungern ($\Longrightarrow$ **nicht fair**)
- Prioritätengesteuertes Scheduling eignet sich für interaktive Systeme

Prioritätengesteuertes Scheduling



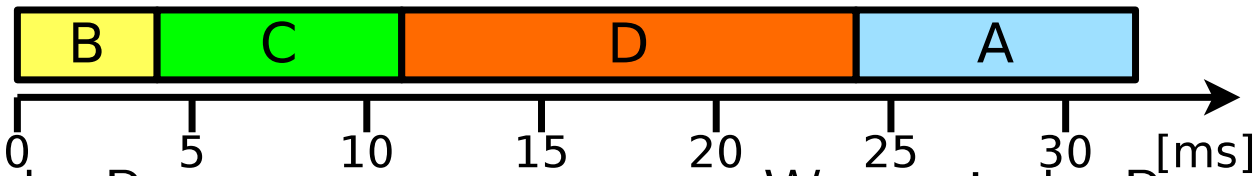
Quelle: William Stallings. Betriebssysteme. Pearson Studium. 2003

Beispiel zum Prioritätengesteuerten Scheduling

- Auf einem Einprozessorrechner sollen vier Prozesse verarbeitet werden
- Alle Prozesse sind zum Zeitpunkt 0 im Zustand `bereit`

Prozess	CPU-Laufzeit	Priorität
A	8 ms	3
B	4 ms	15
C	7 ms	8
D	13 ms	4

- Ausführungsreihenfolge der Prozesse als Gantt-Diagramm (Zeitleiste)



- Laufzeit der Prozesse

Prozess	A	B	C	D
Laufzeit	32	4	11	24

$\frac{32+4+11+24}{4} = 17,75$ ms

- Wartezeit der Prozesse

Prozess	A	B	C	D
Wartezeit	24	0	4	11

$\frac{24+0+4+11}{4} = 9,75$ ms

First Come First Served (FCFS)

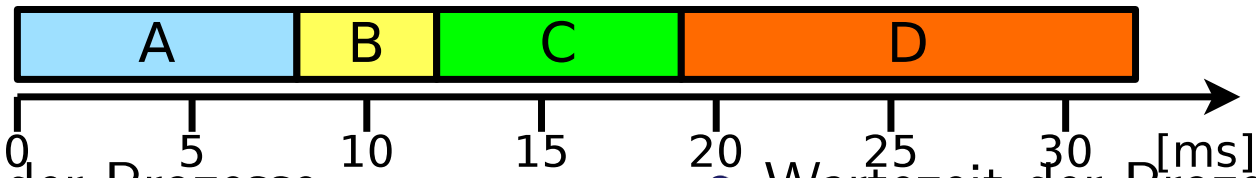
- Funktioniert nach dem Prinzip **First In First Out** (FIFO)
- Die Prozesse bekommen die CPU entsprechend ihrer Ankunftsreihenfolge zugewiesen
- Dieses Scheduling-Verfahren ist vergleichbar mit einer Warteschlange von Kunden in einem Geschäft
- Laufende Prozesse werden nicht unterbrochen
  - Es handelt sich um **nicht-präemptives** (nicht-verdrängendes) Scheduling
- FCFS ist **fair**
  - Alle Prozesse werden berücksichtigt
- Die **mittlere Wartezeit kann unter Umständen sehr hoch sein**
  - Prozesse mit kurzer Abarbeitungszeit müssen eventuell lange warten, wenn vor ihren Prozesse mit langer Abarbeitungszeit eingetroffen sind
- FCFS/FIFO eignet sich für Stapelverarbeitung ($\Longrightarrow$ Foliensatz 1)

Beispiel zu First Come First Served

- Auf einem Einprozessorrechner sollen vier Prozesse verarbeitet werden

Prozess	CPU-Laufzeit	Ankunftszeit
A	8 ms	0 ms
B	4 ms	1 ms
C	7 ms	3 ms
D	13 ms	5 ms

- Ausführungsreihenfolge der Prozesse als Gantt-Diagramm (Zeitleiste)



- Laufzeit der Prozesse

Prozess	A	B	C	D
Laufzeit	8	11	16	27

$\frac{8+11+16+27}{4} = 15,5$ ms

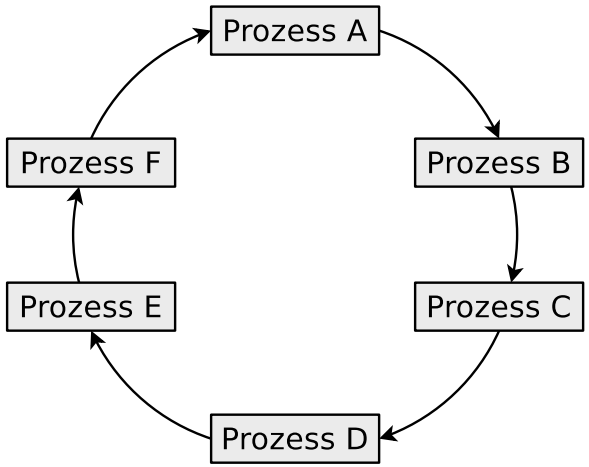
- Wartezeit der Prozesse

Prozess	A	B	C	D
Wartezeit	0	7	9	14

$\frac{0+7+9+14}{4} = 7,5$ ms

Round Robin (RR) – Zeitscheibenverfahren (1/2)

- Es werden Zeitscheiben (*Time Slices*) mit einer festen Dauer festgelegt
- Die Prozesse werden in einer zyklischen Warteschlange nach dem FIFO-Prinzip eingereiht
  - Der erste Prozess der Warteschlange erhält für die Dauer einer Zeitscheibe Zugriff auf die CPU
  - Nach dem Ablauf der Zeitscheibe wird diesem der Zugriff auf die CPU wieder entzogen und er wird am Ende der Warteschlange eingereiht
  - Wird ein Prozess erfolgreich beendet, wird er aus der Warteschlange entfernt
    - Neue Prozesse werden am Ende der Warteschlange eingereiht



- Die Zugriffszeit auf die CPU wird **fair** auf die Prozesse aufgeteilt
- RR mit Zeitscheibengröße ∞ verhält sich wie FCFS

Round Robin (RR) – Zeitscheibenverfahren (2/2)

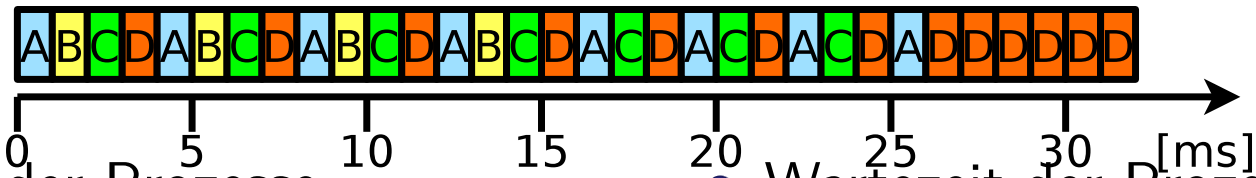
- Je länger die Bearbeitungsdauer eines Prozesses ist, desto mehr Runden sind für seine vollständige Ausführung nötig
- Die Größe der Zeitschlitze ist wichtig für die Systemgeschwindigkeit
  - Je kürzer sie sind, desto mehr Prozesswechsel müssen stattfinden
    $\Longrightarrow$ Hoher Overhead
  - Je länger sie sind, desto mehr geht die Gleichzeitigkeit verloren
    $\Longrightarrow$ Das System hängt/*ruckelt*
- Die Größe der Zeitschlitze liegt üblicherweise im ein- oder zweistelligen Millisekundenbereich
- **Bevorzugt Prozesse, die eine kurze Abarbeitungszeit haben**
- **Präemptives (verdrängendes) Scheduling-Verfahren**
- Round Robin Scheduling eignet sich für interaktive Systeme

Beispiel zu Round Robin

- Auf einem Einprozessorrechner sollen vier Prozesse verarbeitet werden
- Alle Prozesse sind zum Zeitpunkt 0 im Zustand `bereit`
- Zeitquantum $q = 1$ ms

Prozess	CPU-Laufzeit
A	8 ms
B	4 ms
C	7 ms
D	13 ms

- Ausführungsreihenfolge der Prozesse als Gantt-Diagramm (Zeitleiste)



- Laufzeit der Prozesse

Prozess	A	B	C	D
Laufzeit	26	14	24	32

$\frac{26+14+24+32}{4} = 24$ ms

- Wartezeit der Prozesse

Prozess	A	B	C	D
Wartezeit	18	10	17	19

$\frac{18+10+17+19}{4} = 16$ ms

Shortest Job First (SJF) / Shortest Process Next (SPN)

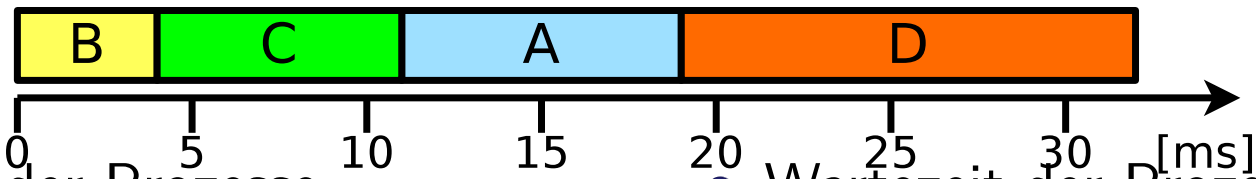
- Der Prozess mit der kürzesten Abarbeitungszeit erhält als erster Zugriff auf die CPU
- **Nicht-präemptives (nicht-verdrängendes) Scheduling**
- Hauptproblem:
  - Für jeden Prozess muss bekannt sein, wie lange er bis zu seiner Terminierung braucht, also wie lange seine Abarbeitungszeit ist
    - Ist in der Realität praktisch nie der Fall ($\Longrightarrow$ **unrealistisch**)
- Lösung:
  - Die Abarbeitungszeit der Prozesse wird abgeschätzt, indem die Abarbeitungszeit vorheriger Prozesse erfasst und analysiert wird
- SJF ist **nicht fair**
  - **Prozesse mit kurzer Abarbeitungszeit werden bevorzugt**
  - Prozesse mit langer Abarbeitungszeit erhalten eventuell erst nach sehr langer Wartezeit oder **verhungern**
- Wenn die Abarbeitungszeit der Prozesse abgeschätzt werden kann, eignet sich SJF für Stapelverarbeitung ($\Longrightarrow$ Foliensatz 1)

Beispiel zu Shortest Job First

- Auf einem Einprozessorrechner sollen vier Prozesse verarbeitet werden
- Alle Prozesse sind zum Zeitpunkt 0 im Zustand `bereit`

Prozess	CPU-Laufzeit
A	8 ms
B	4 ms
C	7 ms
D	13 ms

- Ausführungsreihenfolge der Prozesse als Gantt-Diagramm (Zeitleiste)



- Laufzeit der Prozesse

Prozess	A	B	C	D
Laufzeit	19	4	11	32

$\frac{19+4+11+32}{4} = 16,5$ ms

- Wartezeit der Prozesse

Prozess	A	B	C	D
Wartezeit	11	0	4	19

$\frac{11+0+4+19}{4} = 8,5$ ms

Shortest Remaining Time First (SRTF)

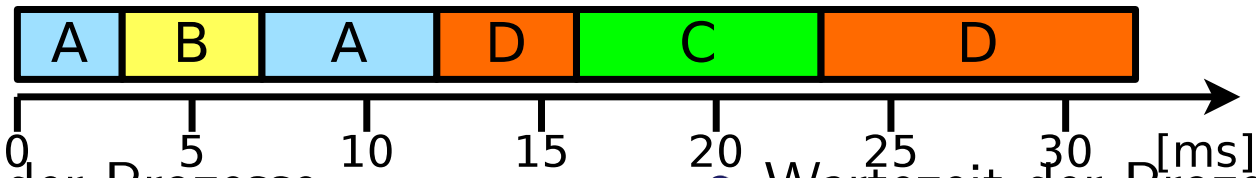
- **Präemptives** SJF heißt **Shortest Remaining Time First** (SRTF)
- Trifft ein neuer Prozess ein, wird die Restlaufzeit des aktuell rechnenden Prozesses mit jedem Prozess in der Liste der wartenden Prozesse verglichen
  - Hat der aktuell rechnende Prozesses die kürzeste Restlaufzeit, darf er weiter rechnen
  - Haben ein oder mehr Prozesse in der Liste der wartenden Prozesse eine kürzere Abarbeitungszeit bzw. Restlaufzeit, erhält der Prozess mit der kürzesten Restlaufzeit Zugriff auf die CPU
- Hauptproblem: Die Restlaufzeit muss bekannt sein ($\Longrightarrow$ **unrealistisch**)
- **Solange kein neuer Prozess eintrifft, wird kein rechnender Prozess unterbrochen**
  - Die Prozesse in der Liste der wartenden Prozesse werden nur dann mit dem aktuell rechnenden Prozess verglichen, wenn ein neuer Prozess eintrifft!
- Prozesse mit langer Laufzeit können **verhungern** ($\Longrightarrow$ **nicht fair**)

Beispiel zu Shortest Remaining Time First

- Auf einem Einprozessorrechner sollen vier Prozesse verarbeitet werden

Prozess	CPU-Laufzeit	Ankunftszeit
A	8 ms	0 ms
B	4 ms	3 ms
C	7 ms	16 ms
D	13 ms	11 ms

- Ausführungsreihenfolge der Prozesse als Gantt-Diagramm (Zeitleiste)



- Laufzeit der Prozesse

Prozess	A	B	C	D
Laufzeit	12	4	7	21

$\frac{12+4+7+21}{4} = 11$ ms

- Wartezeit der Prozesse

Prozess	A	B	C	D
Wartezeit	4	0	0	8

$\frac{4+0+0+8}{4} = 3$ ms

Highest Response Ratio Next (HRRN)

- Faire Variante von SJF/SRTF
  - Berücksichtigt das Alter der Prozesse um **Verhungern zu vermeiden**
- **Antwortquotient** (Response Ratio) wird für jeden Prozess berechnet

$$\text{Antwortquotient} = \frac{\text{geschätzte Rechenzeit} + \text{Wartezeit}}{\text{geschätzte Rechenzeit}}$$

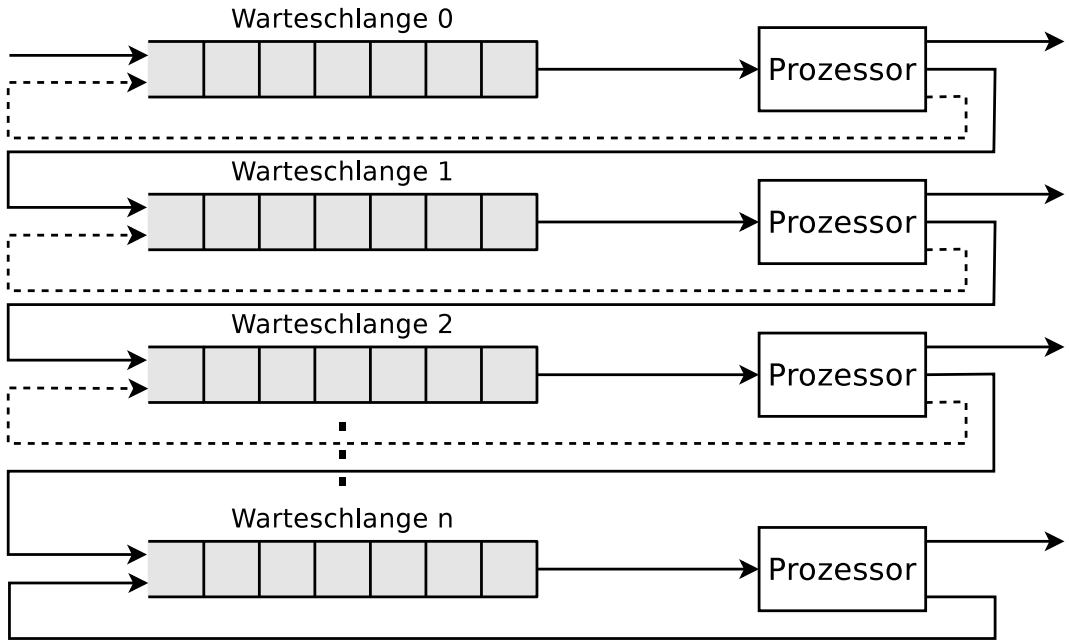
- Wert des Antwortquotienten bei der Erzeugung eines Prozesses: 1.0
  - Der Wert steigt bei kurzen Prozessen schnell an
  - Ziel: Der Antwortquotient soll für alle Prozesse möglichst gering sein
    - Dann arbeitet das Scheduling effizient
- Nach Beendigung oder bei Blockade eines Prozesses, bekommt der Prozess mit dem höchsten Antwortquotient die CPU zugewiesen
- Wie bei SJF/SRTF müssen die Laufzeiten der Prozesse durch statistische Erfassung aus der Vergangenheit **abgeschätzt** werden
- Es ist unmöglich, dass Prozesse verhungern $\Longrightarrow$ HRRN ist **fair**

Multilevel-Feedback-Scheduling (1/2)

- Es ist **unmöglich, die Rechenzeit verlässlich im voraus zu kalkulieren**
  - Lösung: Prozesse, die schon länger aktiv sind, werden **bestraft**
- **Multilevel-Feedback-Scheduling** arbeitet mit mehreren Warteschlangen
  - Jede Warteschlange hat eine andere Priorität oder Zeitmultiplex (z.B. 70%:15%:10%:5%)
- Jeder neue Prozess kommt in die oberste Warteschlange
  - Damit hat er die höchste Priorität
- Innerhalb jeder Warteschlange wird Round Robin eingesetzt
  - Gibt ein Prozess die CPU freiwillig wieder ab, wird er wieder in die selbe Warteschlange eingereiht
  - Hat ein Prozess seine volle Zeitscheibe genutzt, kommt er in die nächst tiefere Warteschlange mit einer niedrigeren Priorität
    - Die Prioritäten werden bei diesem Verfahren also **dynamisch** vergeben
- Multilevel-Feedback-Scheduling ist **unterbrechendes Scheduling**

Multilevel-Feedback-Scheduling (2/2)

- Vorteil:
  - **Keine komplizierten Abschätzungen**!
    - Neue Prozesse werden schnell in eine Prioritätsklasse eingeordnet
- **Bevorzugt neue Prozesse gegenüber älteren** (länger laufenden) Prozessen



Quelle: William Stallings. Betriebssysteme. Pearson Studium. 2003

- Prozesse mit vielen Ein-/Ausgabeoperationen werden bevorzugt, weil sie nach einer freiwilligen Abgabe der CPU wieder in die ursprüngliche Warteliste eingeordnet werden $\Longrightarrow$ Dadurch behalten Sie ihre Priorität
- Ältere, länger laufende Prozesse werden verzögert

Moderne Betriebssysteme (z.B. Linux, Mac OS X und Microsoft Windows) verwenden für das Scheduling der Prozesse Varianten des Multilevel-Feedback-Scheduling

Klassische und moderne Scheduling-Verfahren

	Scheduling NP	Scheduling P	Fair	CPU-Laufzeit muss bekannt sein	Berücksichtigt Prioritäten
Prioritätengesteuertes Scheduling	X	X	nein	nein	ja
First Come First Served	X		ja	nein	nein
~~Last Come First Served~~	~~X~~	~~X~~	~~nein~~	~~nein~~	~~nein~~
Round Robin		X	ja	nein	nein
Shortest Job First	X		nein	ja	nein
~~Longest Job First~~	~~X~~		~~nein~~	~~ja~~	~~nein~~
Shortest Remaining Time First		X	nein	ja	nein
~~Longest Remaining Time First~~		~~X~~	~~nein~~	~~ja~~	~~nein~~
Highest Response Ratio Next	X		ja	ja	nein
~~Earliest Deadline First~~	~~X~~	~~X~~	~~ja~~	~~nein~~	~~nein~~
~~Fair-Share~~		~~X~~	~~ja~~	~~nein~~	~~nein~~
~~Statisches Multilevel-Scheduling~~		~~X~~	~~nein~~	~~nein~~	~~ja (statisch)~~
Multilevel-Feedback-Scheduling		X	ja	nein	ja (dynamisch)

- NP = Nicht-präemptives Scheduling, P = Präemptives Scheduling
- Ein Schedulingverfahren ist „fair“, wenn jeder Prozess irgendwann Zugriff auf die CPU erhält
- Es ist unmöglich, die Rechenzeit verlässlich im voraus zu kalkulieren

Einfaches Beispiel zum Scheduling

Prozess	CPU-Laufzeit	Priorität
A	5 ms	15
B	10 ms	5
C	3 ms	4
D	6 ms	12
E	8 ms	7

- Auf einem Einprozessorrechner sollen 5 Prozesse verarbeitet werden
- Alle Prozesse sind zum Zeitpunkt 0 im Zustand `bereit`
- Hohe Prioritäten sind durch hohe Zahlen gekennzeichnet

- Skizzieren Sie die Ausführungsreihenfolge der Prozesse mit einem Gantt-Diagramm (Zeitleiste) für **Round Robin** (Zeitquantum $q = 1\,\text{ms}$), **FCFS**, **SJF** und **Prioritätengesteuertes Scheduling**
- Berechnen Sie die mittleren Laufzeiten und Wartezeiten der Prozesse
  - Laufzeit = Zeit von der Ankunft bis zur Terminierung
  - Wartezeit = Laufzeit - Rechenzeit

Im Zweifelsfall immer FIFO anwenden

Das heißt im Detail: Wenn das Entscheidungskriterium des verwendeten Scheduling-Verfahrens auf mehrere Prozesse zutrifft, dann nehmen Sie den ältesten Prozess $\Longrightarrow$ FIFO

Einfaches Beispiel zum Scheduling

Prozess	CPU-Laufzeit	Priorität
A	5 ms	15
B	10 ms	5
C	3 ms	4
D	6 ms	12
E	8 ms	7

Laufzeit	A	B	C	D	E
RR					
FCFS					
SJF					
PS*					

* Prioritätengesteuertes Scheduling

Round Robin (Zeitquantum = 1)

First Come First Served

Shortest Job First

Prioritätengesteuertes Scheduling

Wartezeit	A	B	C	D	E
RR					
FCFS					
SJF					
PS*					

* Prioritätengesteuertes Scheduling

- Die Wartezeit ist die Zeit in der `bereit`-Liste

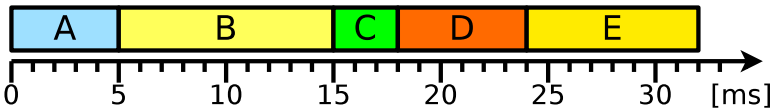
Das Beispiel ist ein „einfaches Beispiel“, weil es keine unterschiedlichen Ankunftszeiten gibt. Alle Prozesse sind zum Zeitpunkt 0 im Zustand `bereit`

Lösung – Gantt-Diagramm + Laufzeit + Wartezeit

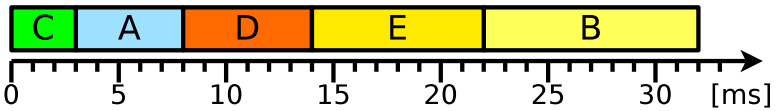
Prozess	CPU-Laufzeit	Priorität
A	5 ms	15
B	10 ms	5
C	3 ms	4
D	6 ms	12
E	8 ms	7

Round Robin (Zeitquantum = 1)

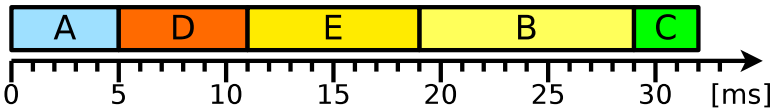
First Come First Served



Shortest Job First



Prioritätengesteuertes Scheduling



Laufzeit	A	B	C	D	E
RR	20	32	13	25	30
FCFS	5	15	18	24	32
SJF	8	32	3	14	22
PS*	5	29	32	11	19

* Prioritätengesteuertes Scheduling

RR $(20 + 32 + 13 + 25 + 30) \;/\; 5 = 24$ ms
FCFS $(5 + 15 + 18 + 24 + 32) \;/\; 5 = 18,8$ ms
SJF $(8 + 32 + 3 + 14 + 22) \;/\; 5 = 15,8$ ms
PS $(5 + 29 + 32 + 11 + 19) \;/\; 5 = 19,2$ ms

Wartezeit	A	B	C	D	E
RR	15	22	10	19	22
FCFS	0	5	15	18	24
SJF	3	22	0	8	14
PS*	0	19	29	5	11

* Prioritätengesteuertes Scheduling

RR $(15 + 22 + 10 + 19 + 22) \;/\; 5 = 17,6$ ms
FCFS $(0 + 5 + 15 + 18 + 24) \;/\; 5 = 12,4$ ms
SJF $(3 + 22 + 0 + 8 + 14) \;/\; 5 = 9,4$ ms
PS $(0 + 19 + 29 + 5 + 11) \;/\; 5 = 12,8$ ms

Fazit

- Von den untersuchten Scheduling-Verfahren hat/haben...
  - **SJF** die beste mittlere Laufzeit und kürzeste mittlere Wartezeit
  - **RR** die schlechteste mittlere Laufzeit und mittlere Wartezeit
- **RR** verursacht häufige Prozesswechsel
  - Der dadurch entstehende **Overhead** wirkt sich zusätzlich negativ auf die Systemleistung aus
- Die Größe des Overhead hängt von der Größe der Zeitscheiben ab
  - **Kurze Zeitscheiben** $\Longrightarrow$ hoher Overhead
  - **Lange Zeitscheiben** $\Longrightarrow$ Antwortzeiten sind eventuell zu lang für interaktive Prozesse