FH Bielefeld
University of
Applied Sciences

Vorlesung

Betriebssysteme

Teil 3
Prozesse

Dozent
Prof. Dr.-Ing.
Martin Hoffmann
martin.hoffmann@fh-bielefeld.de

FH Bielefeld
University of
Applied Sciences

Inhalt

- Prozesse und Lebenszyklus von Prozessen
- Threads
- Scheduling

FH Bielefeld
University of
Applied Sciences

Ziele der heutigen Vorlesung

- Das Prozess- und das Threadmodell verstehen und erläutern können
- Den Lebenszyklus von Prozessen und Threads innerhalb eines Betriebssystems verstehen und erläutern können
- Scheduling Mechanismen erklären können

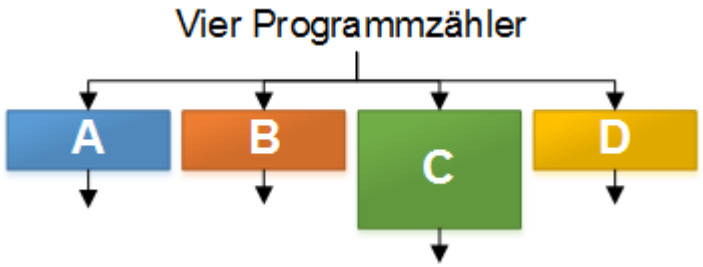
FH Bielefeld
University of
Applied Sciences

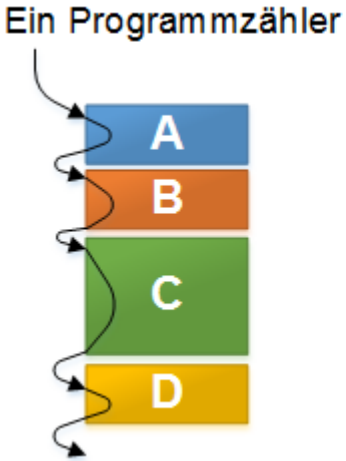
Prozesse

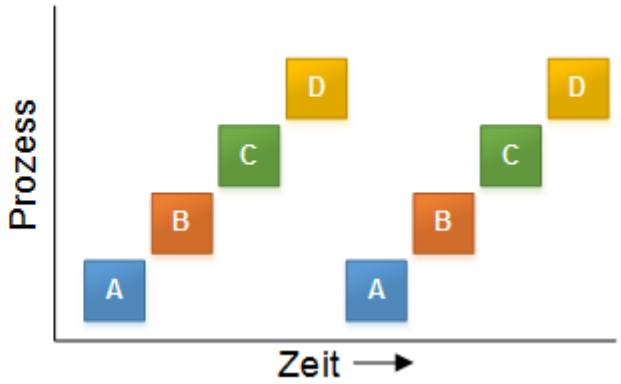
- Programm vs. Prozess
- In den meisten Betriebssystemen
  - laufen **mehrere Programme** auf einem Rechner (Mehrprogrammbetrieb, ***multi-tasking***) simultan und
  - **mehrere Nutzer** teilen sich den Rechner (Mehrbenutzerbetrieb, ***multi-user***).
- Die einzelnen Programme werden vom Betriebssystem verwaltet und **quasi-parallel abgearbeitet** (bzw. echt parallel bei Multiprozessorsystemen).

FH Bielefeld
University of
Applied Sciences

Programmzähler



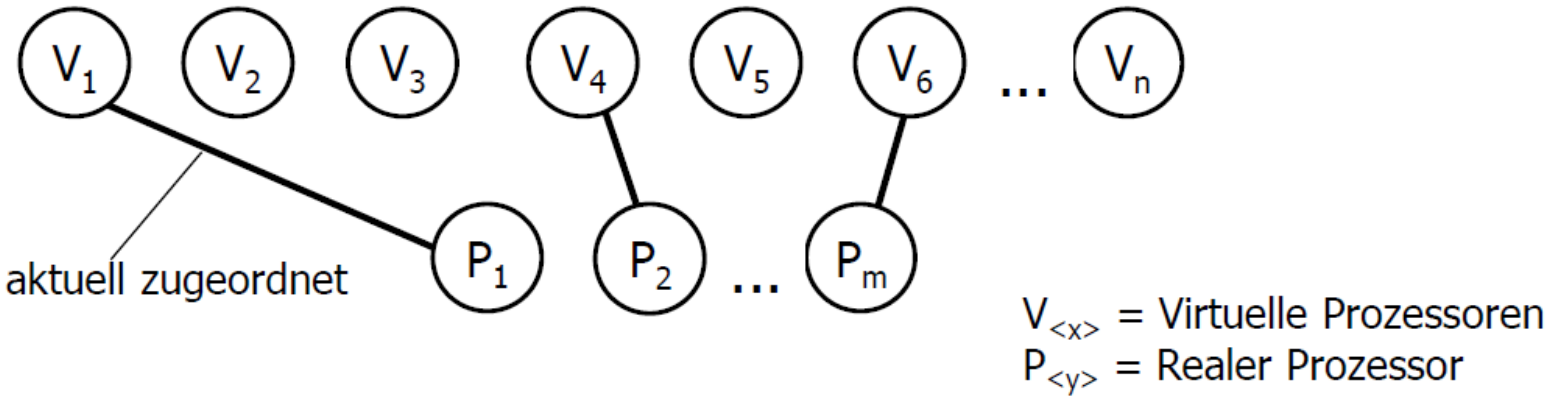




FH Bielefeld
University of
Applied Sciences

Virtuelle Prozessoren

- Das Betriebssystem ordnet im Multiprogramming jedem Prozess einen virtuellen Prozessor zu
- Echte Parallelarbeit, falls jedem virtuellen Prozessor ein realer Prozessor bzw. Rechnerkern zugeordnet wird
- **Quasi parallel**: Jeder reale Prozessor ist zu einer Zeit immer nur einem virtuellen Prozessor zugeordnet und es gibt Prozess-Umschaltungen



FH Bielefeld
University of
Applied Sciences

Prozesse: Definition *Prozess*

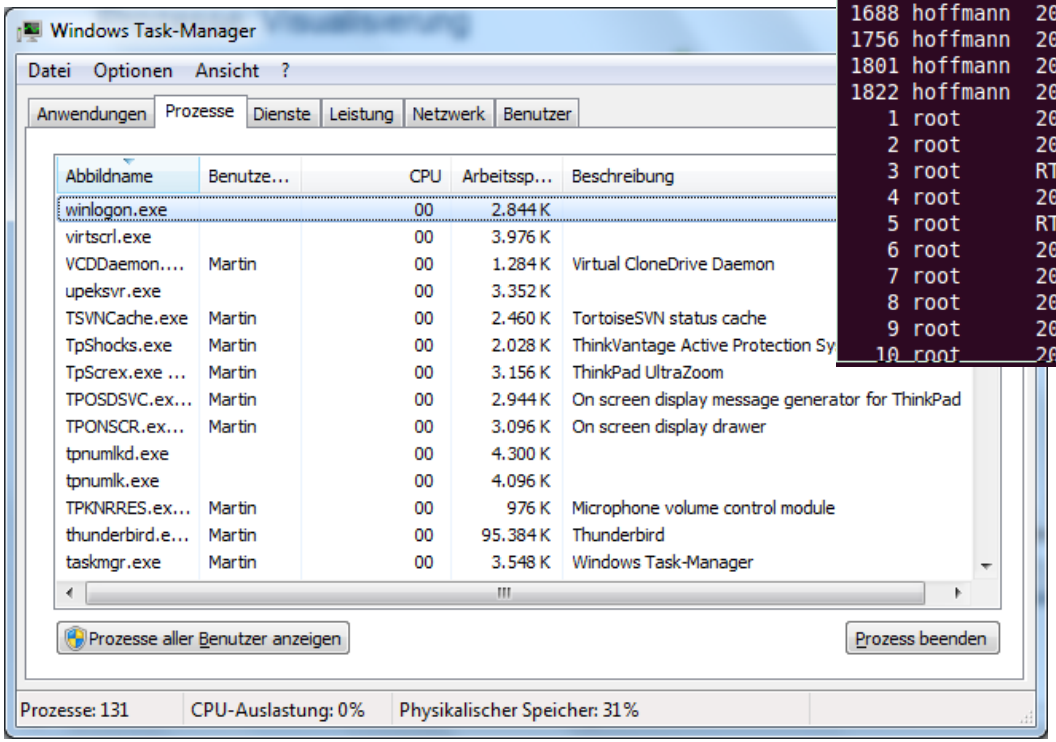
- Ein **Prozess** (*process*, *task*) ist
  - eine durch ein **Programm** spezifizierte Folge von Aktionen,
  - deren erste begonnen, deren letzte aber noch nicht abgeschlossen ist. (Prozess = *Programm in Ausführung*)
- Ein Prozess hat einen **Ausführungskontext** und einen **Zustand**.
- Ein Prozess benötigt **Betriebsmittel** (CPU, Speicher, Dateien, ...) und ist selbst ein Betriebsmittel, das vom Betriebssystem verwaltet wird (Erzeugung, Terminierung, Scheduling, ...).
- Das **Betriebssystem** (*Scheduler*) entscheidet, welcher Prozess zu welchem Zeitpunkt ausgeführt wird.
- Ein **Prozessorkern** führt in jeder Zeiteinheit maximal einen Prozess aus. Laufen mehrere Prozesse auf einem Rechner, finden Prozesswechsel statt.
- Prozesse sind gegeneinander **isoliert**:
  - Jeder Prozess besitzt (virtuell) seine eigenen Betriebsmittel wie etwa den Adressraum.
  - Das Betriebssystem sorgt für die Abschottung der Prozesse gegeneinander

FH Bielefeld
University of
Applied Sciences

Prozesse: Visualisierung

```
top - 19:06:37 up 5 min,  2 users,  load average: 0.48, 0.23, 0.09
Tasks: 144 total,   1 running, 143 sleeping,   0 stopped,   0 zombie
Cpu(s):  1.0%us,  0.3%sy,  0.0%ni, 98.0%id,  0.7%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   2061692k total,   385296k used,  1676396k free,    42700k buffers
Swap:  1340408k total,        0k used,  1340408k free,   181764k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1011 root      20   0 47352  19m 8348 S  0.7  1.0   0:03.35 Xorg
 1421 root      20   0 15956 2864 2332 S  0.3  0.1   0:00.32 vmtoolsd
 1473 root      20   0  3636 1232 1052 S  0.3  0.1   0:00.10 hald-addon-stor
 1688 hoffmann  20   0 46624  18m  14m S  0.3  0.9   0:01.57 vmware-user-loa
 1756 hoffmann  20   0 40588  12m 9.9m S  0.3  0.6   0:00.78 wnck-applet
 1801 hoffmann  20   0 46096  12m 9856 S  0.3  0.6   0:00.61 gnome-terminal
 1822 hoffmann  20   0  2568 1256  952 R  0.3  0.1   0:00.29 top
    1 root      20   0  2828 1656 1208 S  0.0  0.1   0:01.34 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    6 root      20   0     0    0    0 S  0.0  0.0   0:00.02 events/0
    7 root      20   0     0    0    0 S  0.0  0.0   0:00.00 cpuset
    8 root      20   0     0    0    0 S  0.0  0.0   0:00.00 khelper
    9 root      20   0     0    0    0 S  0.0  0.0   0:00.00 netns
   10 root      20   0     0    0    0 S  0.0  0.0   0:00.00 async/mgr
```

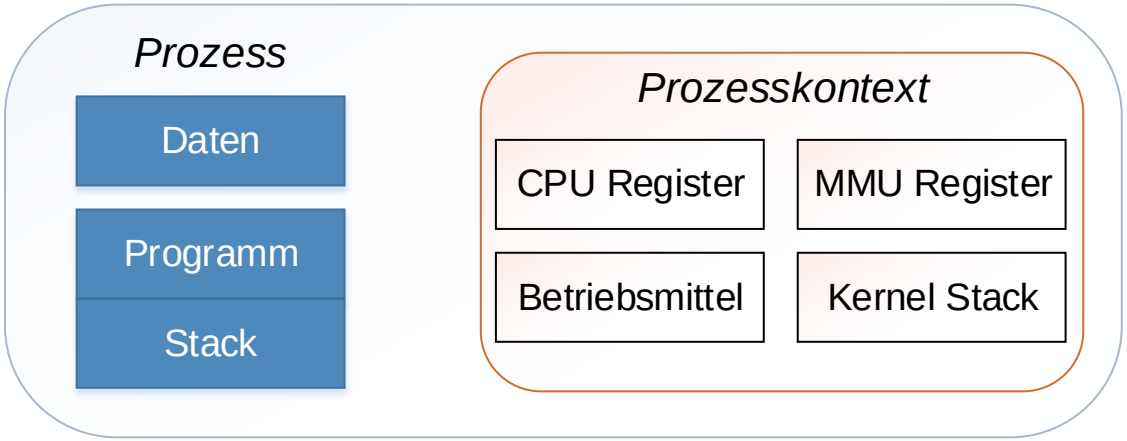


FH Bielefeld
University of
Applied Sciences

Prozesse: Eigenschaften

Ein Prozess wird beschrieben durch:

- Seine Folge von **Maschinenbefehlen** (*program code*, *text section*).
- Seinen augenblicklichen **Zustand** (*program counter*, CPU Register, …)
- Den Inhalt seines **Stapelspeichers** (Keller, *stack*)
- Seine globalen **Daten** (*data section*)
- Seine allozierten **Betriebsmittel** (geöffnete Dateien, …)



FH Bielefeld
University of
Applied Sciences

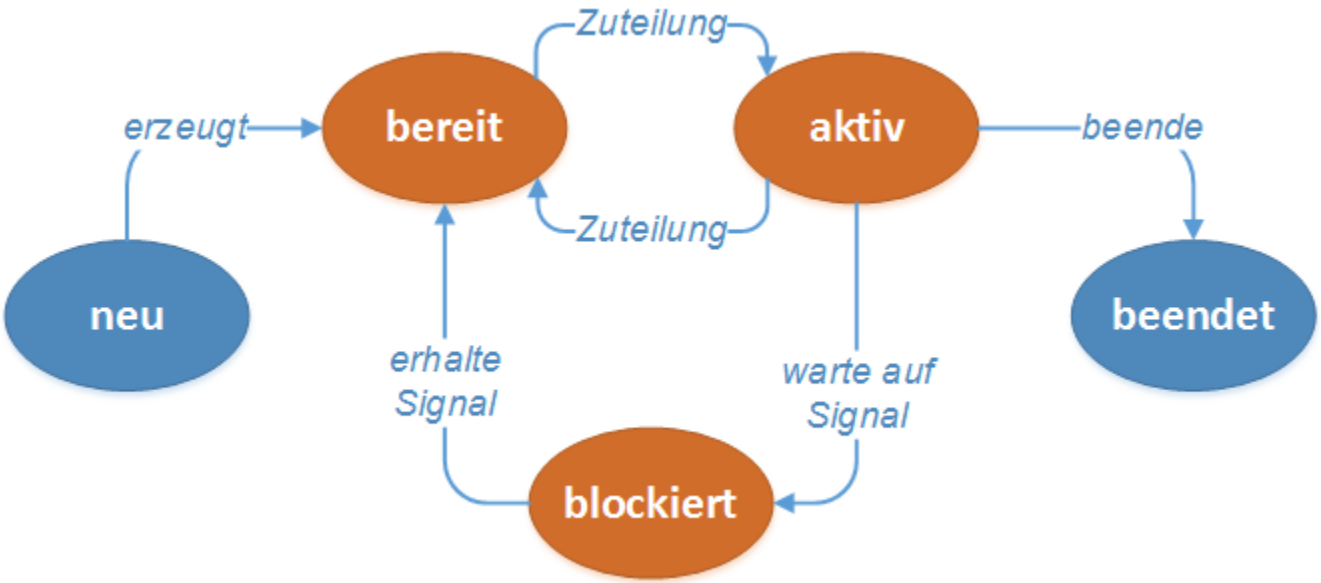
Prozesse: Zustände

Ein Prozess kann mehrere Zustände annehmen:

- **Aktiv** (*running*): Der Prozess belegt gerade das Betriebsmittel CPU und wird ausgeführt.
- **Bereit** (*ready*): Der Prozess wartet darauf, die CPU zu erhalten.
- **Blockiert** (*waiting*): Der Prozess wartet
  - auf ein E/A Gerät,
  - eine Nachricht von einem anderen Prozess,
  - ein Zeitgebersignal oder ähnliches.
  - Selbst wenn die CPU zur Verfügung steht, kann der Prozess nicht aktiv werden.
- **Neu** (*new*): Ein neuer Prozess wird erzeugt.
- **Beendet** (*terminated*): Der Prozess ist beendet.

FH Bielefeld
University of
Applied Sciences

Prozesse: Zustandsübergänge



Prozesse: Erzeugung

- Betriebssystem erzeugt **ersten Prozess**
  - Unix System V Systemstart: Starten von **/sbin/init**
- Existierender Prozess kann **neue Prozesse** erzeugen.
- Beim *Portable Operating System Interface* (POSIX): Systemaufruf **fork()**
  - Neu erzeugter Prozess (Child) ist eine **echte Kopie** des erzeugenden Prozesses (Parent), besitzt aber eine neue PID (*Process ID*).
  - Rückgabewert von **fork()** für beide Prozesse unterschiedlich:
- *Child* (Rückgabewert: 0) und
- *Parent* (Rückgabewert: PId des Childs) können unterschieden werden.
  - Child und Parent führen nach **fork()** die gleichen Instruktionen aus.
- (Child-)Prozess kann sich selbst mit Hilfe des **execv()** POSIX-Systemaufrufs durch Instruktionen und Daten aus einer anderen Programm-Datei ersetzen.
  - Wird z.B. von der Shell benutzt, um andere Programme zu starten.

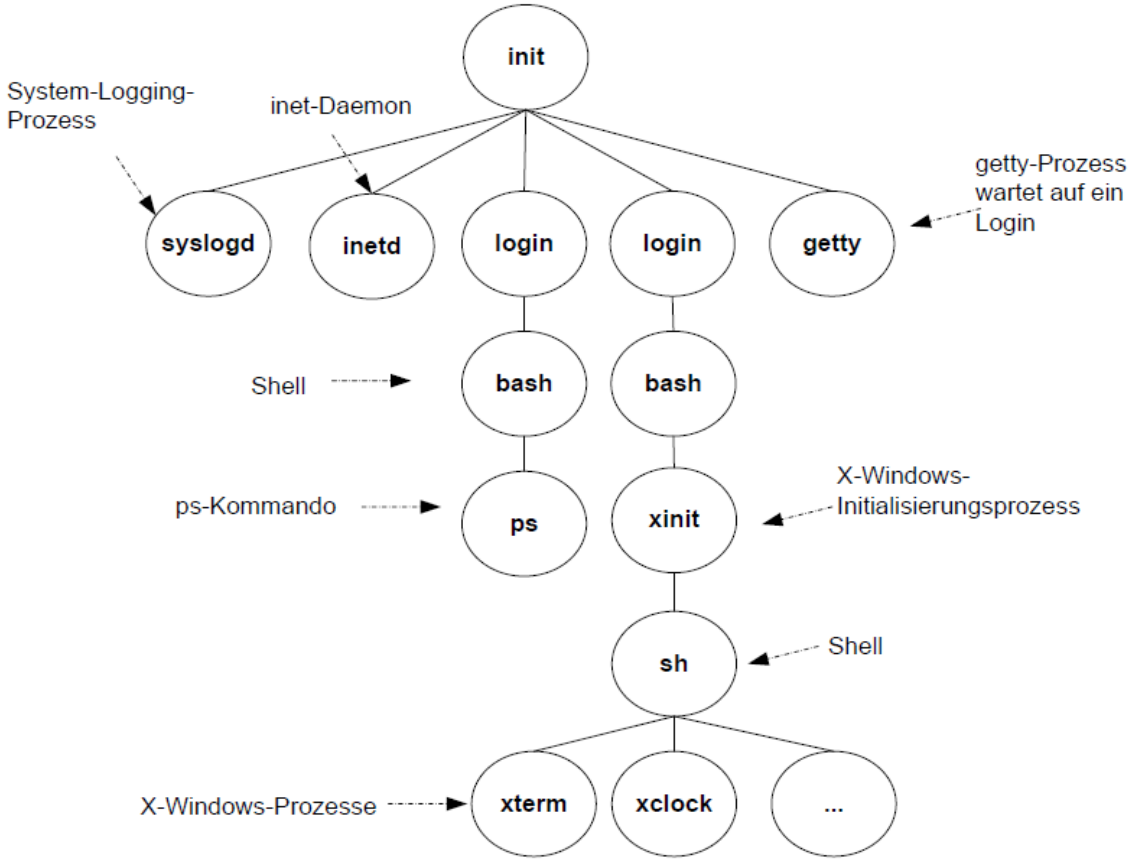
Prozesse: Erzeugung (Forts.)

Beispiel: einfacher *Kommandointerpreter* unter UNIX:

```
while (1)
{                       /* repeat forever                    */
  type_prompt();        /* display prompt on screen          */
  read_command();       /* read input from the terminal      */
  pid = fork();         /* create a new process              */
  if (pid < 0)
  {                     /* repeat if system call failed      */
    perror("fork");
    continue;
  }
  if (pid != 0)
  {                     /* parent process                    */
    waitpid(pid, &status, 0);   /* wait for child            */
  }
  else
  {                     /* child process                     */
    execve(command, params, 0); /* execute command           */
  }
}
```

FH Bielefeld
University of
Applied Sciences

Unix* Prozessbaum



xterm = Standard-Terminalemulator unter Unix/Linux

FH Bielefeld
University of
Applied Sciences

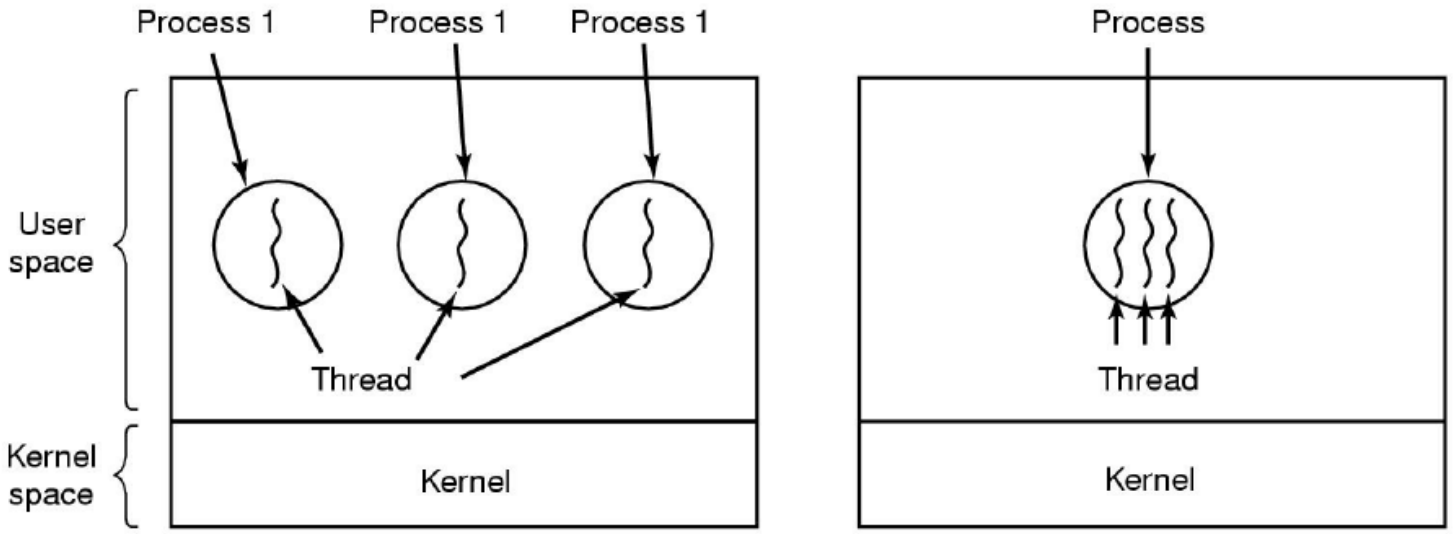
Inhalt

- Prozesse und Lebenszyklus von Prozessen
- Threads
- Scheduling

FH Bielefeld
University of
Applied Sciences

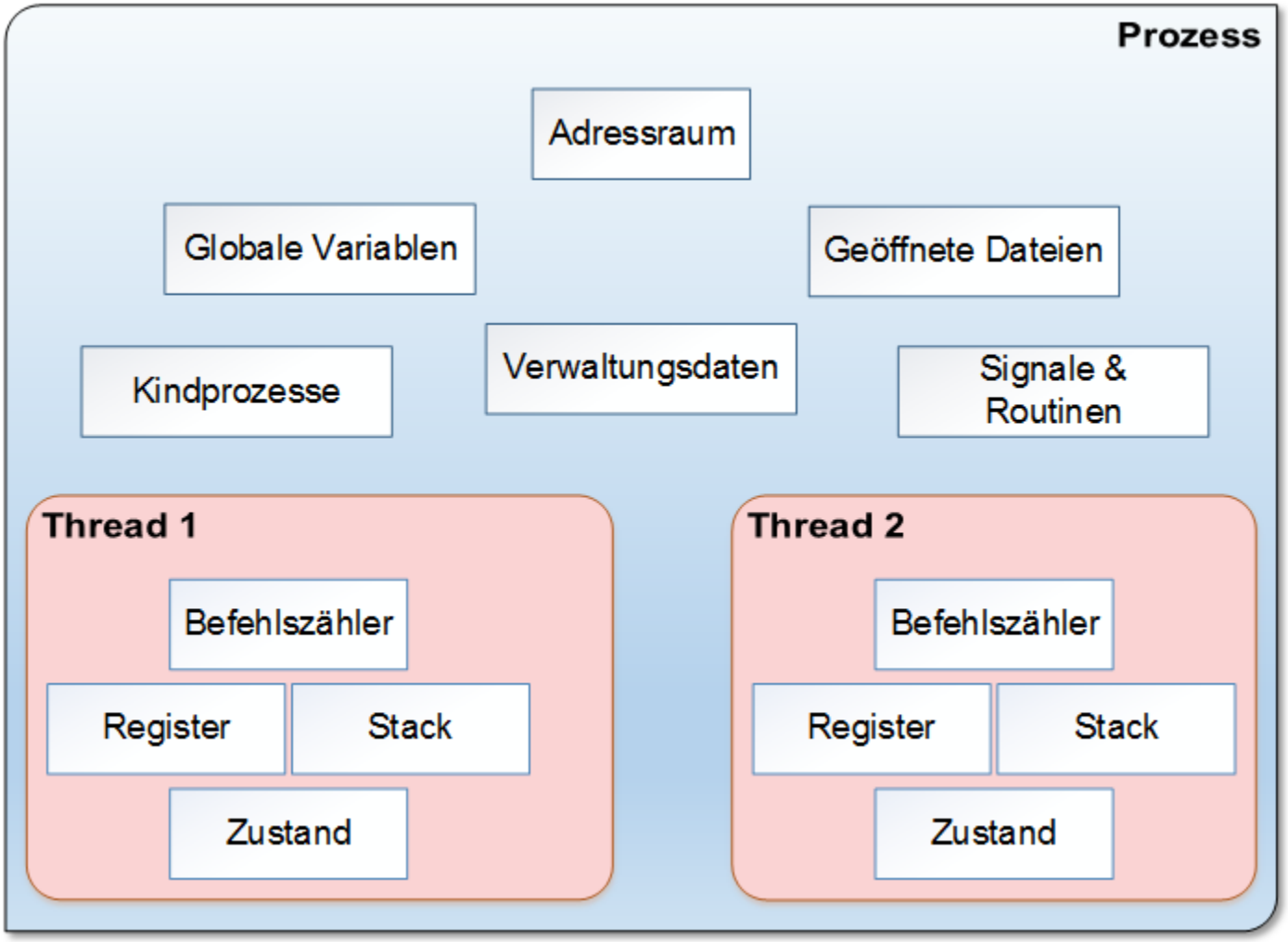
Prozesse: Leichtgewichtige Prozesse (Threads)

- **Threads** (Fäden) sind parallele Verarbeitungsflüsse, die nicht in einem eigenen Adressraum ablaufen.
- Sie teilen sich **gemeinsame Ressourcen** innerhalb eines Prozesses



(Quelle Bild: A. Tanenbaum)

Prozesse & Threads



FH Bielefeld
University of
Applied Sciences

Prozesse vs. Threads

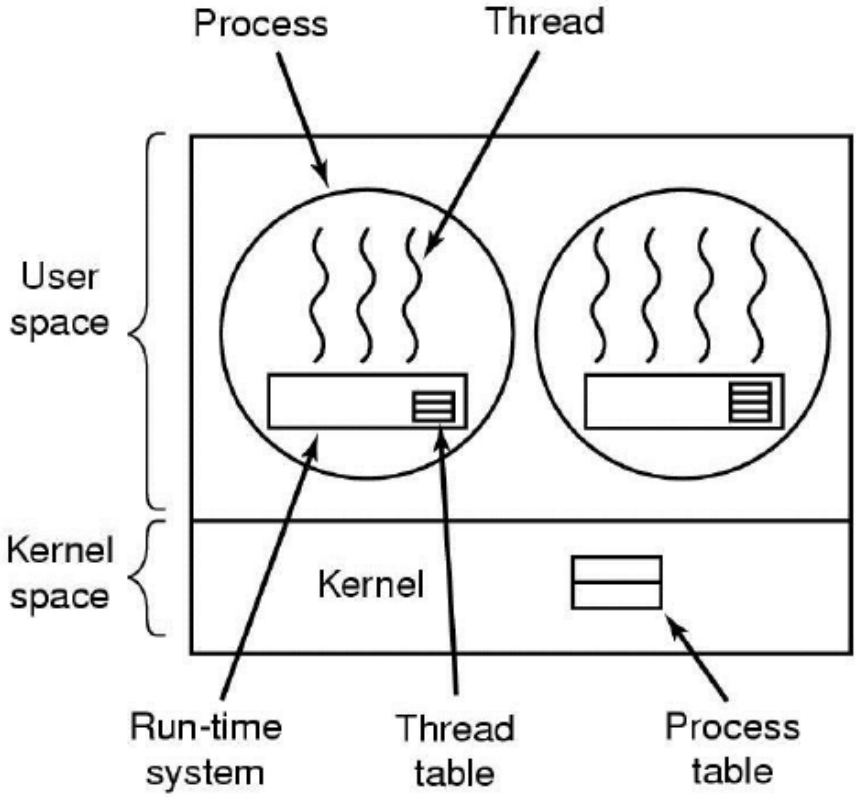
- **Kontextwechsel** zwischen Threads ist **effizienter** als zwischen Prozessen:
  - Kein Umschalten zwischen den Adressräumen.
  - Kein Retten und Restaurieren des Kontextes (nur Programmzähler, Register und Stackpointer)

  → Pro Zeiteinheit sind **mehr Threadwechsel** als Prozesswechsel möglich
- **Threads** innerhalb eines Prozesses sind **nicht gegeneinander geschützt**.
- **Portierbarkeit** wird z.B. durch POSIX API gewährleistet:
  - 60 Funktionen zum Erzeugen, Beenden, … von Threads
  - **pthread_create**, **pthread_exit**, ….

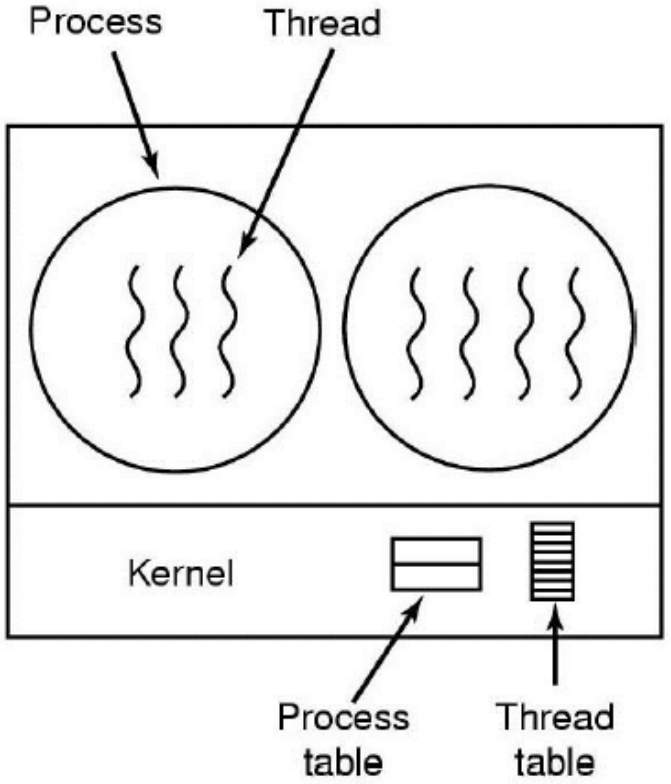
FH Bielefeld
University of
Applied Sciences

Leichtgewichtige Prozesse (Threads)

- Threads können im **Benutzermodus** oder im **Kernmodus** verwaltet werden:



(Quelle Bild: A. Tanenbaum)



FH Bielefeld
University of
Applied Sciences

Threads: Modi

	Threads im Benutzermodus	Threads im Kernmodus
Vorteile	**Schnelle** Thread**umschaltung** (kein Einsprung in den Kern)	Verwaltung **einheitlich** für alle laufenden Prozesse
	Erweiterung auf nicht Multi-threading-fähige Systeme möglich	Threadwechsel **schneller** als reine Prozessumschaltung
	Einsatz **sprachbezogener** Multi-threading-Modelle möglich (z.B. Java Threads)	Vorteile von **Multiprozessor-umgebungen** können genutzt werden
Nachteile	Bei blockierenden Aufrufen **blockieren** alle Threads	Klar **langsamer** als Threads im Benutzermodus
	Vorteile von **Multiprozessor-systemen** können **nicht genutzt** werden (alle Threads auf der selben CPU)	

FH Bielefeld
University of
Applied Sciences

Threads: Nutzung

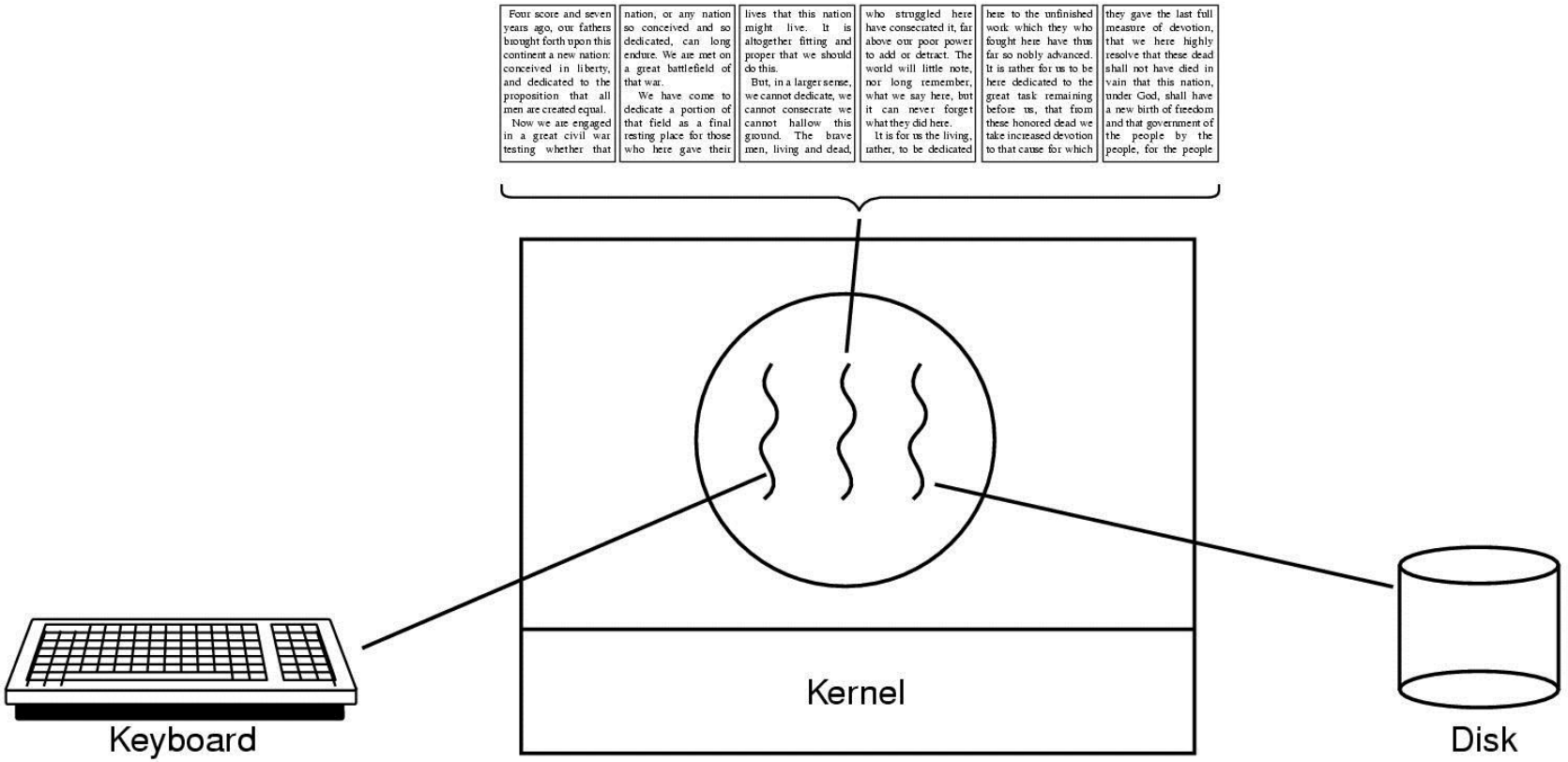
Threads werden genutzt, um:

- Programme mit mehreren **gleichzeitigen Aktivitäten** zu modellieren.
- Höhere Performance als bei Aufteilung auf mehrere Prozesse zu erzielen, da
  - Erzeugung und Terminierung einfacher (keine Ressourcen) und
  - Umschaltung einfacher.
- Parallelen Einheiten das Arbeiten auf **gemeinsamen Daten** zu ermöglichen (identischer Adressraum).
- Vorteile bei *Hyperthreading* Prozessoren auszunutzen.

FH Bielefeld
University of
Applied Sciences

Threads: Anwendung

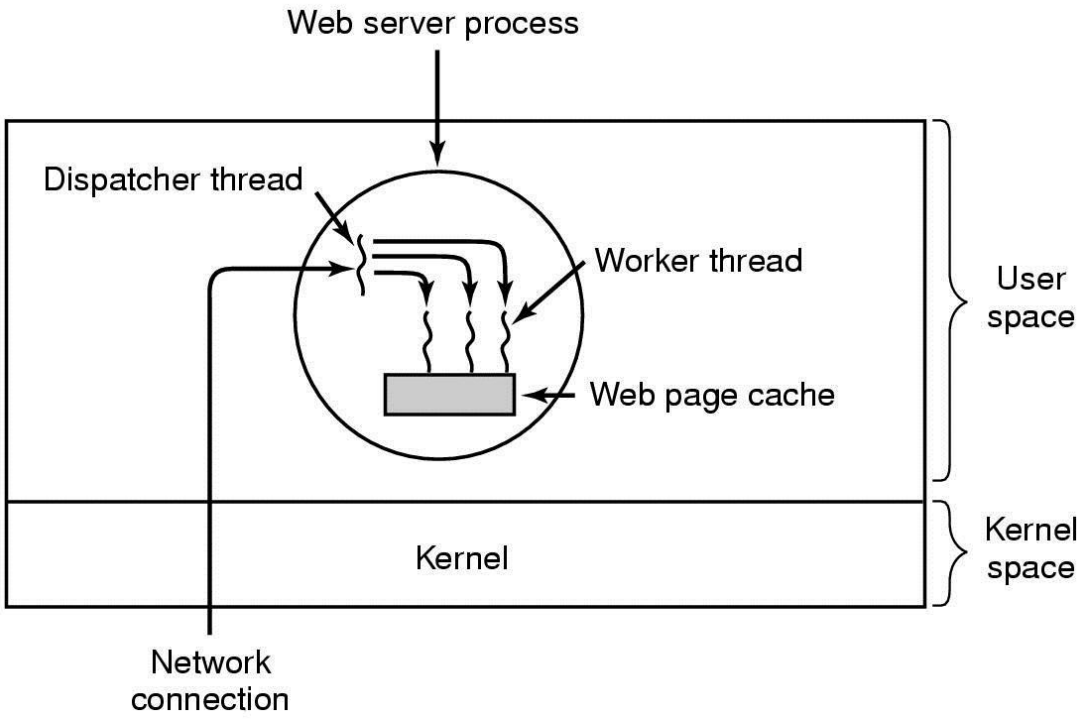
Zum Beispiel Textverarbeitungsprogramm mit 3 Threads:



FH Bielefeld
University of
Applied Sciences

Threads: Anwendung

- Zum Beispiel **Web-Server mit mehreren Threads**:
  (a) *Dispatcher* Thread
  (b) *Worker* Thread



FH Bielefeld
University of
Applied Sciences

Inhalt

- Prozesse und Lebenszyklus von Prozessen
- Threads
- Scheduling

FH Bielefeld
University of
Applied Sciences

Scheduling

- **Aufgabe** des Scheduling:
  - Mehrere Prozesse konkurrieren um die Ressource CPU.
  - Der Scheduler entscheidet, welcher Prozess die Ressource erhält.
- ***Allgemeine* Ziele** des Scheduling:
  - Jeder Prozess bekommt Rechenzeit (*Fairness*)
  - Gewichtung von Prozessen
    - Kurze **Antwortzeit** bei interaktiven Prozessen
    - Wartezeiten minimieren (möglichst wenig Zeit im “Bereit” Zustand)
    - Erfüllung von **Echtzeitanforderungen** (definierte Reaktionszeit)
    - **Ressourcenbelegung** optimieren → **CPU** Auslastung maximieren
    - **Durchsatz** maximieren (Prozesse pro Zeiteinheit)

FH Bielefeld
University of
Applied Sciences

Scheduling

- **Ziele des Scheduling:**
  - Echtzeit-Systeme:
    - **Vorhersehbares** Verhalten: ein Prozess hat eine "***Deadline***".
    - Ein Überschreiten der definierten Reaktionszeit ist in keinem Fall akzeptabel. (Beispiel: Sicherheitsabschaltung eines Antriebs)
  - Interaktive Systeme:
    - Benutzer erwartet **schnelle Reaktion** auf seine Anforderung.
    - Keine harte Echtzeitanforderung, da verspätete Reaktion zwar ärgerlich aber nicht kritisch ist
  - Batch System (z.B. Rechenzentrum):
    - Maximaler **Durchsatz** von Prozessen
    - CPU gleichmäßig belegen
    - Minimieren der Zeit vom Start bis zum Ende eines Prozesses (**Turnaround Time**)

FH Bielefeld
University of
Applied Sciences

Scheduling

Wann erzeugt der Scheduler einen Schedule?

- Nach dem **Erzeugen** eines neuen Prozesses:
  - Wird der neue oder der erzeugende Prozess weiter ausgeführt?
- Nach dem **Beenden** eines Prozesses
- Nach **Ablauf** einer Zeitscheibe
- Wenn ein Prozess **blockiert**, z.B. bei einer E/A Anforderung oder bei einer Interprozess-Kommunikation
- Wenn ein E/A-Ereignis eintritt (**Interrupt**)

FH Bielefeld
University of
Applied Sciences

Scheduling: Begriffe

Begriffe im Zusammenhang mit dem Scheduling:

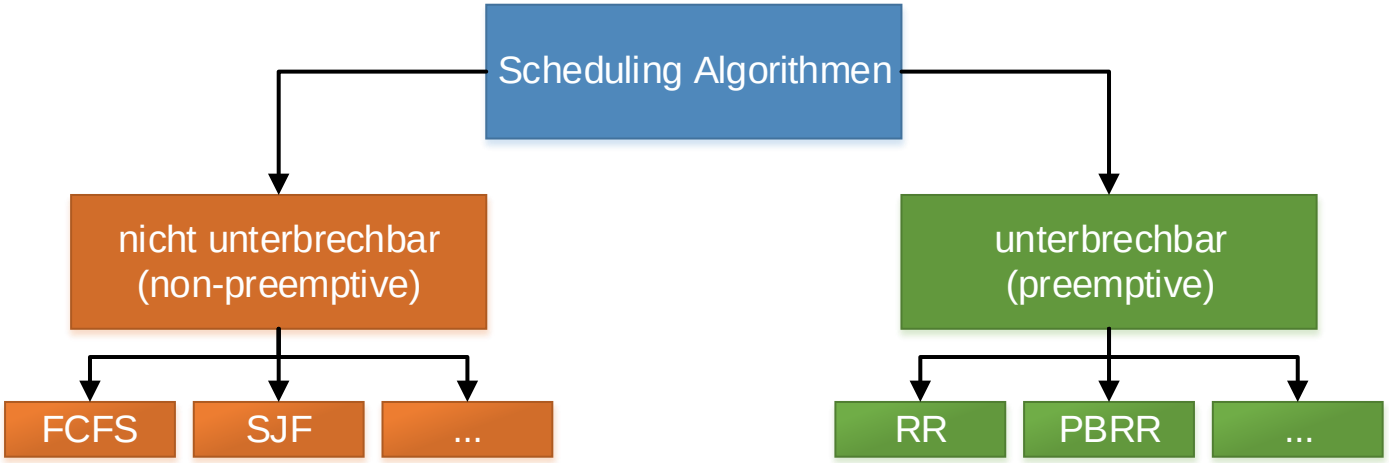
- **Ankunftszeit** eines Prozesses P_i: $T_{a,i}$
- **Startzeit** eines Prozesses P_i: $T_{s,i}$
- **Unterbrechungszeit** eines Prozesses P_i: $T_{u,i}$
- **Rechenzeit** eines Prozesses P_i: T_i

Abgeleitete Größen:

- **Wartezeit** eines Prozesses P_i: $T_{w,i} = T_{s,i} - T_{a,i} + T_{u,i}$
- **Verweilzeit** eines Prozesses P_i: $T_{v,i} = T_{w,i} + T_i$
- **Mittlere Wartezeit** für Menge von Prozesses: $T_{\tilde{W}} = \frac{1}{n} \sum_{i=1}^{n} T_{w,i}$
- **Mittlere Verweilzeit** für Menge von Prozessen: $T_{\tilde{V}} = \frac{1}{n} \sum_{i=1}^{n} T_{v,i}$

FH Bielefeld
University of
Applied Sciences

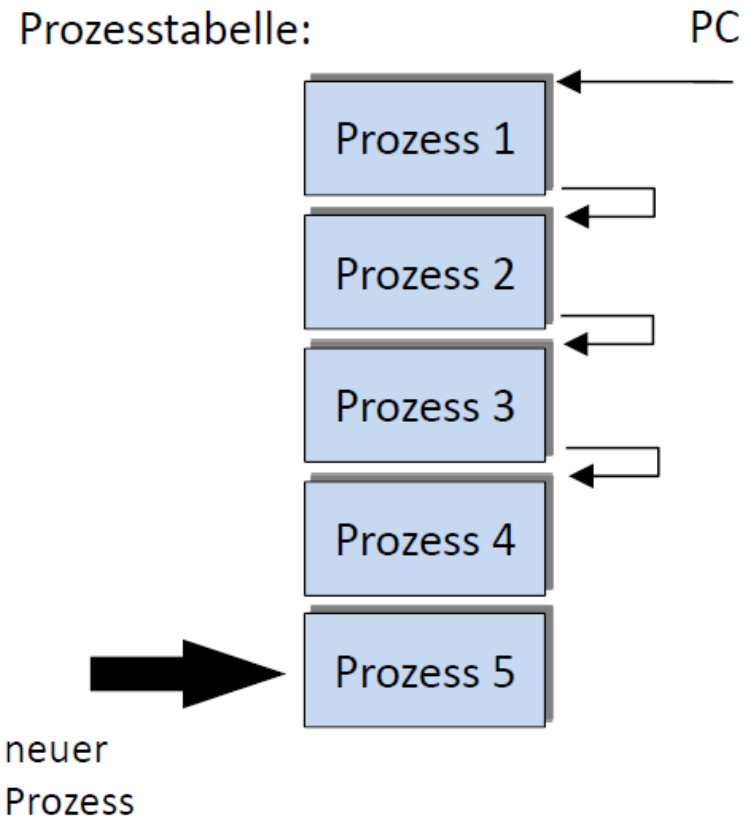
Scheduling: Algorithmen



FH Bielefeld
University of
Applied Sciences

Scheduling Algorithmen: FCFS

First-Come First-Serve (FCFS)

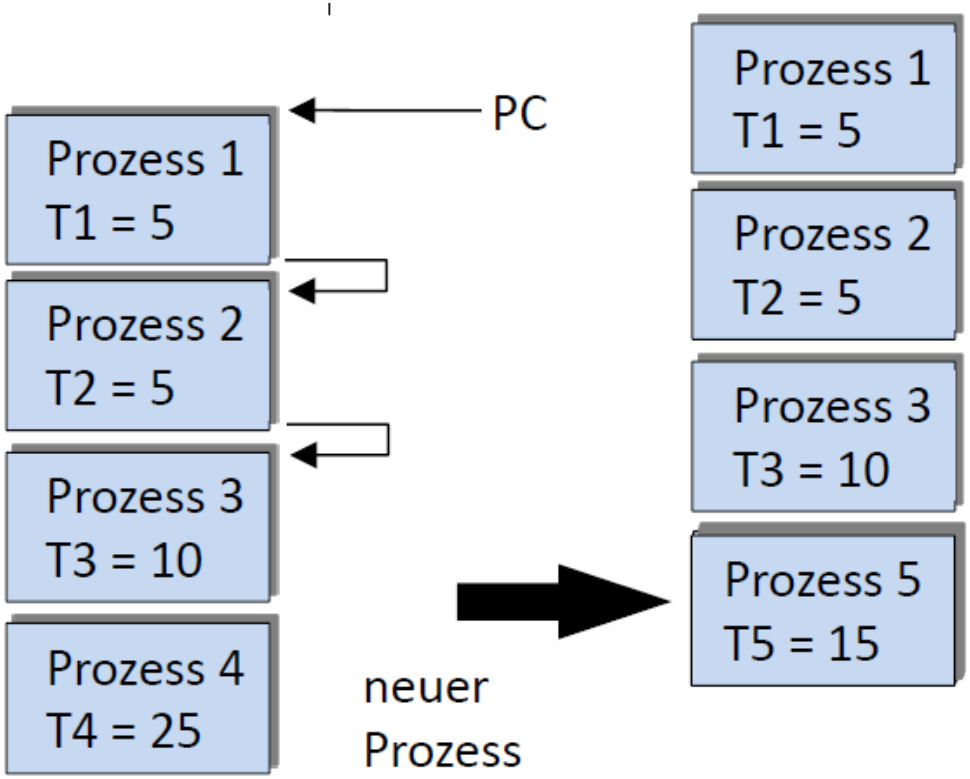


- Prozesse sind nicht unterbrechbar (non-preemptive)
  $$T_{u,i} = 0$$
- Wartezeit für Prozess P_i:
  $$T_{w,i} = \sum_{j=1}^{i-1} T_j$$
- Verweilzeit für Prozess P_i:
  $$T_{v,i} = T_{w,i} + T_i = \sum_{j=1}^{i} T_j$$
- Durchschnittliche Verweilzeit:
  $$T_{\tilde{V}} = \frac{1}{n} \sum_{i=1}^{n} (n+1-i) T_i$$
- Gesamte Bearbeitungszeit:
  $$T_G = \sum_{i=1}^{n} T_i$$

FH Bielefeld
University of
Applied Sciences

Scheduling Algorithmen: SJF

- Shortest Job First (SJF)
- Prozesstabelle:



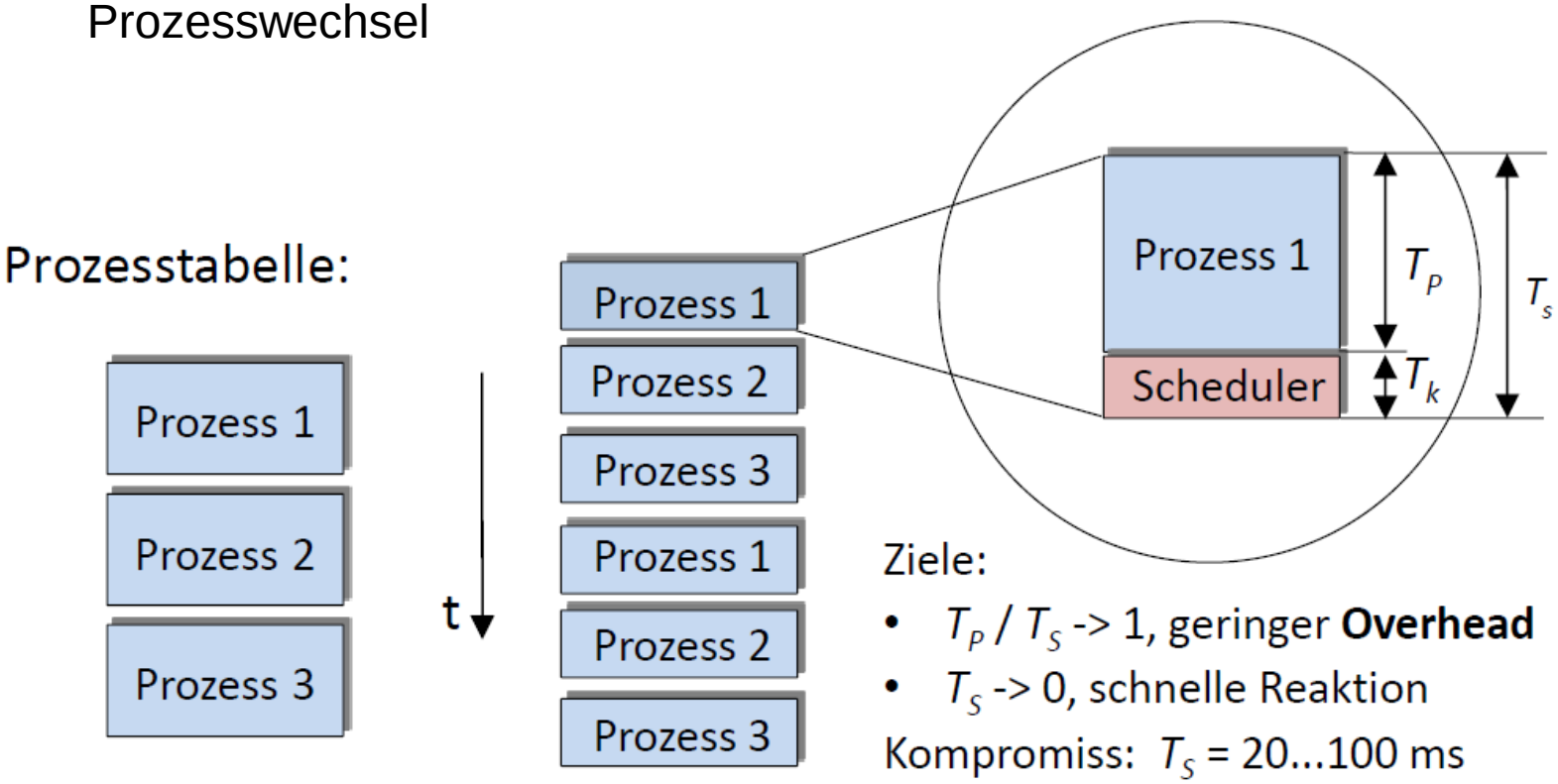
- FCFS und SJF sind nicht unterbrechbar.

  Damit sind sie ungeeignet für interaktive / Mehrbenutzer-Systeme

FH Bielefeld
University of
Applied Sciences

Scheduling Algorithmen: RR

- Round Robin (RR)
- Preemptives Scheduling, nach Ablauf einer Zeitscheibe *TS* erfolgt ein Prozesswechsel

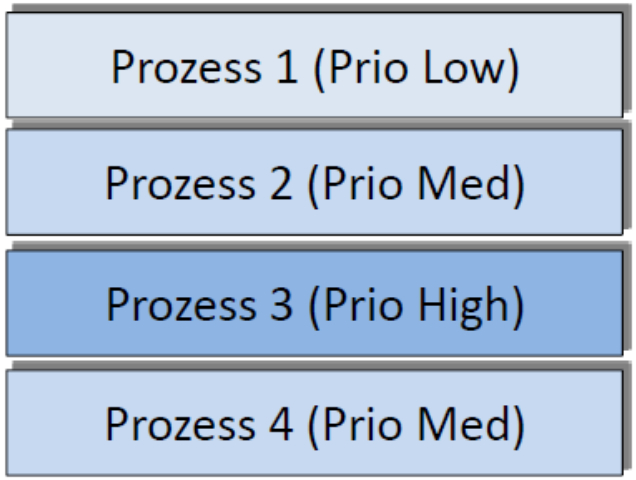


FH Bielefeld
University of
Applied Sciences

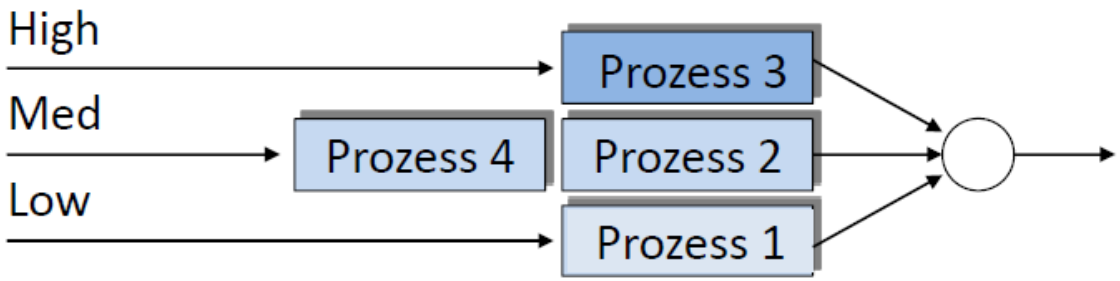
Scheduling Algorithmen: PBRR

- Prioritätenbasiertes Round Robin (PBRR)

Prozesstabelle:



Warteschlangen:



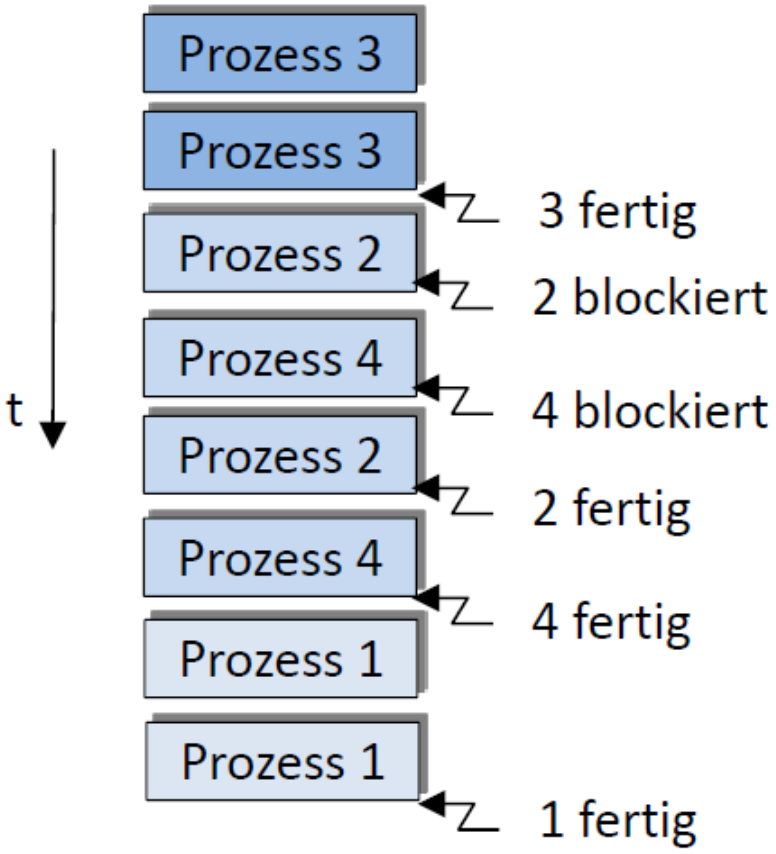
FH Bielefeld
University of
Applied Sciences

Scheduling Algorithmen: PBRR

Variante I:

- Die Warteschlangen werden **gemäß ihrer Priorität** (höchste zuerst) per RR abgearbeitet
  - Bis alle Prozesse der Priorität fertig sind
  - Dann wird nächsthöchste Warteschlange betrachtet
- **Problem**: treffen ständig hochpriorisierte Prozesse ein, verhungern die niedrigpriorisierten

Beispiel für Zeitablauf:



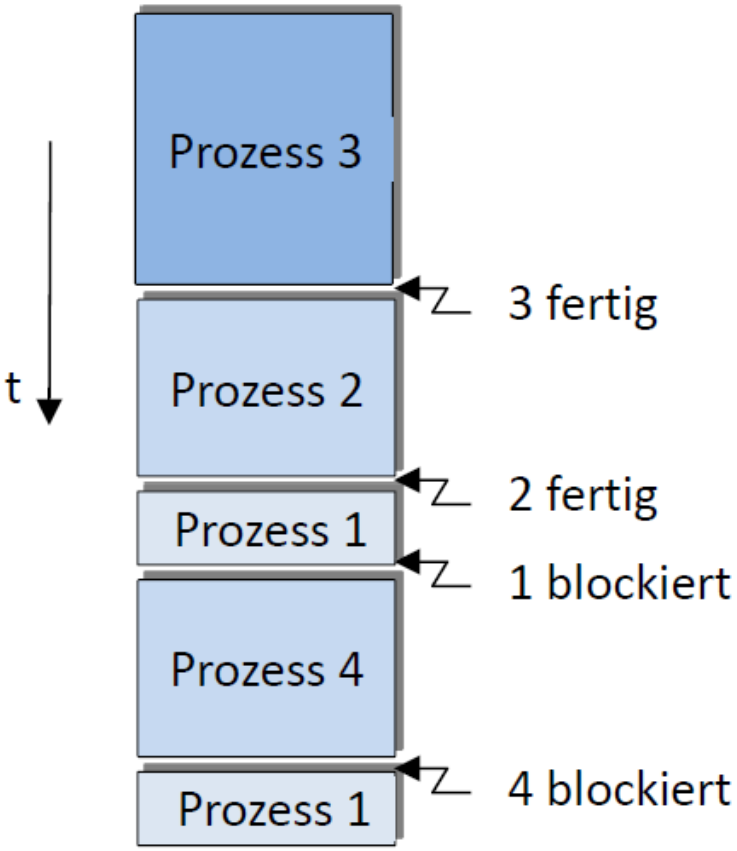
FH Bielefeld
University of
Applied Sciences

Scheduling Algorithmen: PBRR

Variante II:

- Längere Zeitscheiben für Warteschlangen mit höherer **Priorität**
- Wechsel zwischen Warteschlangen per RR
- Mischformen mit Variante I möglich

Beispiel für Zeitablauf:

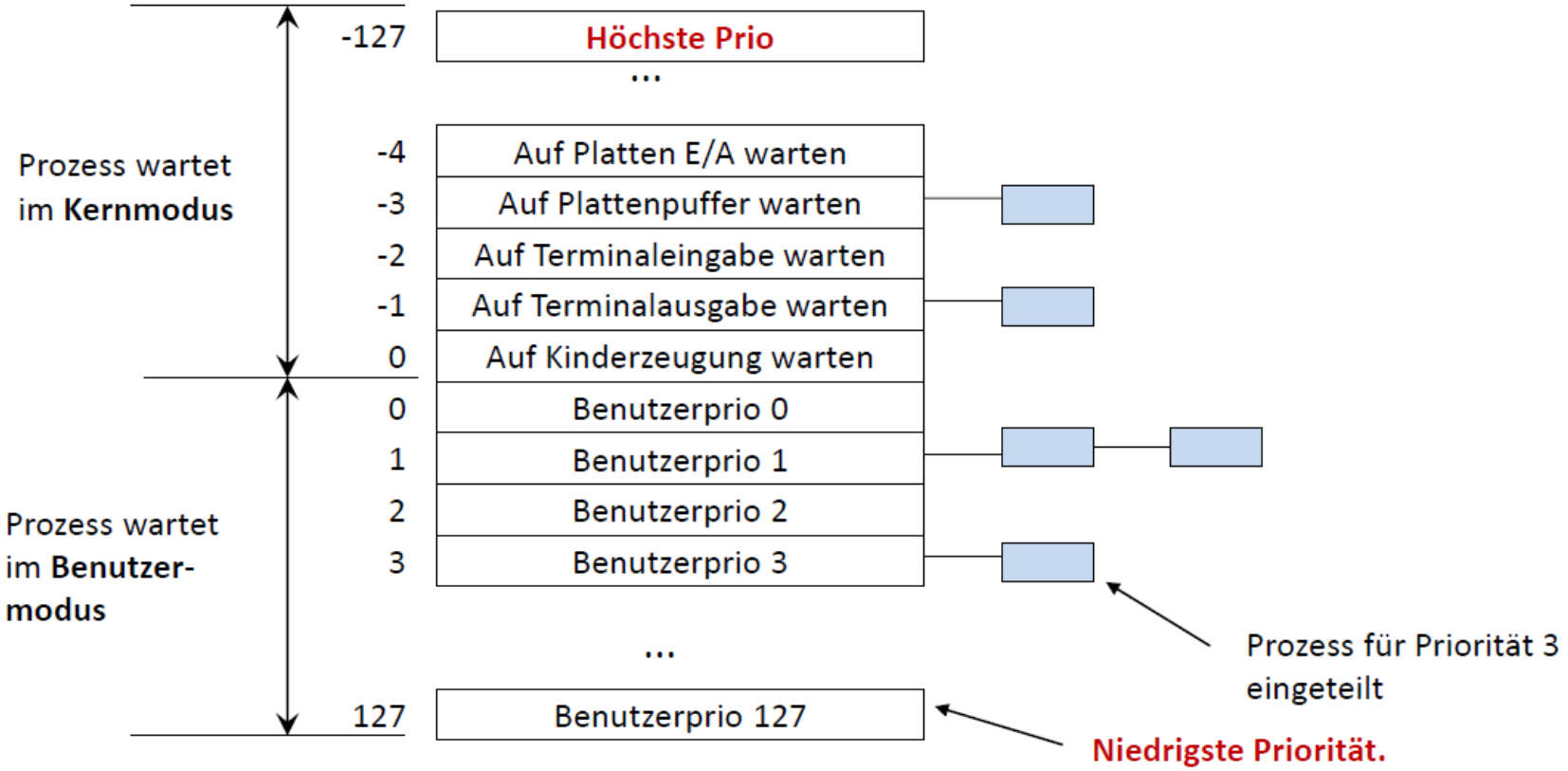


FH Bielefeld
University of
Applied Sciences

Beispiel: Scheduling in Unix

Hinweis: Jede Unix-Variante hat einen eigenen Algorithmus

Hier: *round robin with multilevel feedback algorithm*



FH Bielefeld
University of
Applied Sciences

Beispiel: Scheduling in Unix

- Einmal pro Sekunde wird die Priorität jedes Prozesse im Benutzermodus neu berechnet (*Aging*):
  - *priority = CPU_usage / 2 + nice + base*
- Dabei bedeuten:
  - *CPU_usage*:
    - Die **Anzahl der Systemzeitscheiben**, die der Prozess bereits hatte.
    - Um Prozesse nicht zu bestrafen, wird der Wert **jede Sekunde halbiert**, d.h. der Einfluss der letzten Sekunde ist ½, der der vorletzten Sekunde ist ¼ usw.
  - *nice*:
    - Wert zwischen -20 und 20, **Standardwert** ist 0.
    - **Benutzerprozesse** können den *nice*-Wert auf Werte zwischen 1...20 erhöhen (d.h. Priorität heruntersetzen).
    - **Systemadministratoren** können auch negative Werte verlangen (d.h. Priorität hochsetzen).
  - *base*: Basispriorität, ist fest im System.

FH Bielefeld
University of
Applied Sciences

Multi-level Feedback Algorithm

- Manchmal wird auch mit **variablen Zeitscheiben** gearbeitet:
  - innerhalb einer Warteschlange: *Round Robin*
  - bei *niedrigerer* Priorität: *längeres* Quantum
  - Falls Prozess Quantum aufgebraucht: erniedrigen der Priorität
  - CPU-lastiger Prozess erhält längeres Quantum, wird seltener unterbrochen
- **Windows** (NT, XP):
  - Multi-level Feedback Scheduling mit *32 Prioritätsklassen*
  - Priorität 16-31 (höchste):
    - Echtzeitklasse, statische Priorität
  - Priorität 1-15:
    - normale Prozesse, dynamische Priorität, starke Prioritätserhöhung bei Benutzereingabe,
    - moderatere Erhöhung bei Ende einer E/A, danach schrittweise Reduktion zum Ausgangswert
  - Priorität 0 (niedrigste):
    - Idle-Prozess
- Peter Mandl: Grundkurs Betriebssysteme, Online verfügbar über SpringerLink

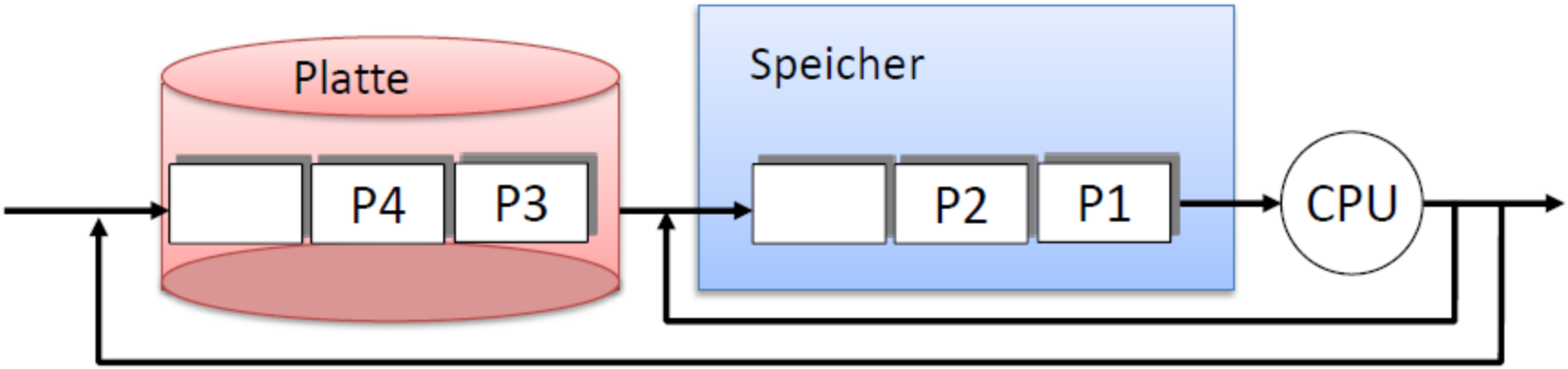
FH Bielefeld
University of
Applied Sciences

Multi-Level-Scheduling

Bei sehr vielen Prozessen können nicht alle Prozesse im Hauptspeicher gehalten werden.

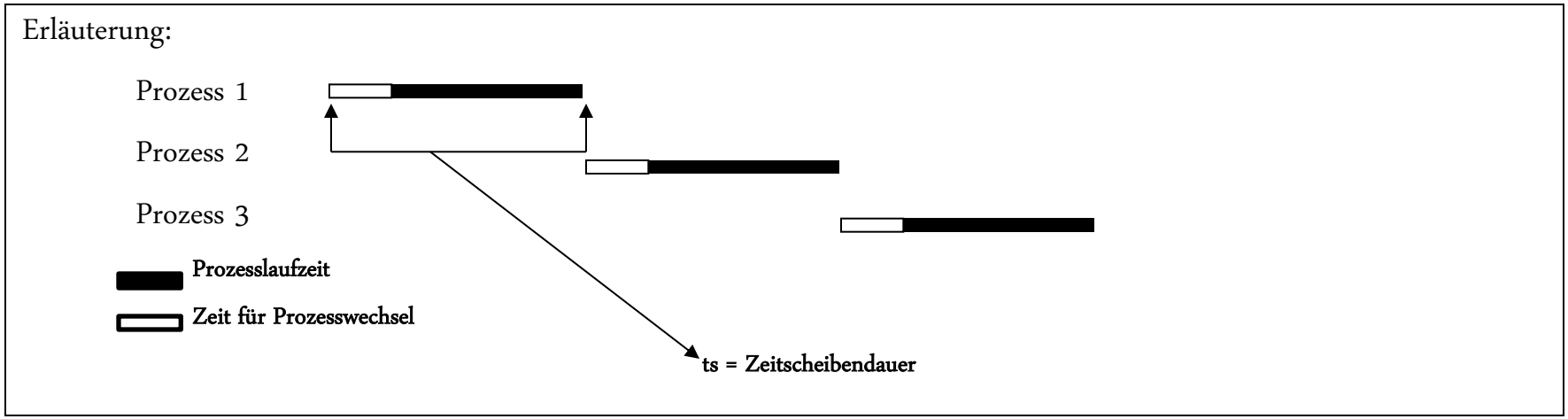
Es wird deshalb ein *zweistufiger* Scheduling-Algorithmus benutzt.

- **CPU Scheduling** (*Kurzzeitscheduling*): Es wird zwischen den Prozessen im Hauptspeicher ausgewählt.
- **Speicher Scheduling** (*Langzeitscheduling*): Verschiebt Prozesse zwischen Hauptspeicher und Platte, so dass alle Prozesse eine Chance haben, irgendwann im Speicher zu sein und ausgeführt zu werden.



FH Bielefeld
University of
Applied Sciences

Testfrage



- Nennen Sie jeweils die Begründung, die Zeitscheibendauer ***ts*** möglichst groß, bzw. möglichst klein zu wählen.

FH Bielefeld
University of
Applied Sciences

Testfrage

Fünf Stapelaufträge A bis E treffen nahezu gleichzeitig in der Reihenfolge A, B, C, D und E in einem Rechenzentrum ein. Ihre geschätzten Laufzeiten sind 9, 5, 2, 4 und 12 Minuten. Ihre extern festgelegten Prioritäten sind 3 (Wissenschaftlicher Mitarbeiter), 5 (Dekan), 2 (Pförtner), 1 (Student) und 4 (Professor). Für jeden der nachstehenden Scheduling Algorithmen bestimme man die **mittlere Verweilzeit (nach welcher Zeit, ab Ankunft, die Prozesse abgearbeitet waren)**. Der Verwaltungsaufwand kann vernachlässigt werden. Die Zeitscheibendauer sei sehr viel kleiner als 1 Minute.

a) First-Come-First-Served

b) Shortest Job First

c) Round Robin

d) Round Robin mit Berücksichtigung der Prioritäten

FH Bielefeld
University of
Applied Sciences

Lösung Aufgabe 1a) und 1b)

a) *First-Come-First-Served*
Die Reihenfolge ist die Eingangsreihenfolge A, B, C, D, E:

$$\bar{T} = \frac{5 \cdot 9 + 4 \cdot 5 + 3 \cdot 2 + 2 \cdot 4 + 1 \cdot 12}{5} \text{min} = 18,2 \text{min}$$

A=**9**
B=**5**+9
...

b) *Shortest Job First*
Die Reihenfolge ist C, D, B, A, E:

$$\bar{T} = \frac{5 \cdot 2 + 4 \cdot 4 + 3 \cdot 5 + 2 \cdot 9 + 1 \cdot 12}{5} \text{min} = 14,2 \text{min}$$

C=**2**
D=**4**+2
...

FH Bielefeld
University of
Applied Sciences

Lösung Aufgabe 1c)
Round Robin mit konstanter Zeitscheibe

c) In den ersten zwei Minuten laufen alle Prozesse gleichberechtigt, nach 10 Minuten ist demnach der erste Prozess(Prozess C) fertig, da er ein Fünftel der CPU bekommen hat und zwei Minuten rechnete. Anschließend teilen sich vier Prozesse die CPU, jeder der vier Prozesse hat bereits zwei Minuten „verbraucht“. Prozess D wird also nach weiteren acht Minuten beendet sein. Nach dem gleichen Schema wird der letzte Prozess nach 32 Minuten beendet sein.

A	B	C	D	E	Zeit
9	5	2	4	12	0
7	3		2	10	T_{VC}=5*2 = 10
5	1			8	T_{VD}=10 + 4*2 = 18
4				7	T_{VB}=18 + 3*1 = 21
				3	T_{VA}=21 + 2*4 = 29
					T_{VE}=29 + 1*3 = 32

Tabelle der Restlaufzeiten

Prozess C ist nach 10 Minuten, Prozess D nach 18, B nach 21, A nach 29 und E nach 32 Minuten fertig. Die mittlere Antwortzeit $\bar{T}$ ist demnach:

$$\bar{T} = \frac{10 + 18 + 21 + 29 + 32}{5}\text{min} = 22\text{min}$$

FH Bielefeld
University of
Applied Sciences

Lösung Aufgabe 1 d)

d) Jeder Prozess bekommt entsprechend seiner Priorität Anteile *n* von der CPU. Zu Beginn sind es 1+2+3+4+5=15 Anteile, von denen z.B. Prozess B 5 erhält.
Prozess B und C haben das beste Verhältnis V von Priorität zu Laufzeit und sind demnach als erste nach einer Zeit von

$$5\text{min} \cdot \frac{n}{\text{Priorität=5}} = 15\text{min}$$ bzw.

$$2\text{min} \cdot \frac{n}{\text{Priorität=2}} = 15\text{min}$$ fertig

A rechnet noch $$9\text{min} - 15\text{min} \cdot \frac{\text{Priorität=3}}{15} = 6\text{min}$$

	A	B	C	D	E
Pri.	3	5	2	1	4
Zeit	9	5	2	4	12

D entsprechend noch 3min und E noch 8min.

Übrig bleiben die Prozesse A (6 min), D (3 min) und E (8 min), die sich jetzt n=3+1+4=8 Teile der CPU teilen müssen. Nach der Terminierung von B und C sind nach weiteren 16 Minuten die Prozesse A und E fertig:

$$6\text{min} \cdot \frac{n}{\text{Priorität=3}} = 16\text{min}$$

$$8\text{min} \cdot \frac{n}{\text{Priorität=4}} = 16\text{min}$$

Der letzte Prozess hat in den 16 Minuten ein Achtel der CPU bekommen, konnte also 2 Minuten Rechenzeit gutmachen. Insgesamt bleibt ihm noch eine Minute zu rechnen. Da er die CPU jetzt allein nutzen kann, terminiert er nach 15+16+1=32 Minuten

FH Bielefeld
University of
Applied Sciences

Lösung Aufgabe 1 d)
Round Robin mit Zeitscheibendauer proportional zur Priorität

Prozesse					CPU-Anteile	Durchlauf-zeit	Verweilzeit der im Durchlauf terminierten Prozesse
A(3)	B(5)	C(2)	D(1)	E(4)			
9	5	2	4	12			
6	-	-	3	8	15	15 min.	$T_{VB}=T_{VC}=15$
3	-	-	2	4	8	8 min.	
-			1	-	8	8 min.	$T_{VA}=T_{VE}=15+8+8=31$
			-		1	1 min.	$T_{VD}=31+1=32$

Prozesse B und C sind nach 15 Minuten, Prozesse A und E nach 31 und D nach 32 Minuten fertig. Die mittlere Antwortzeit

ist demnach: $\bar{T} = \frac{15+15+31+31+32}{5}\text{min} = 24,8\text{min}$

FH Bielefeld
University of
Applied Sciences

Inhalt

- Prozesse und Lebenszyklus von Prozessen
- Threads
- Scheduling

FH Bielefeld
University of
Applied Sciences

Vorlesung

Vielen Dank für Ihre Aufmerksamkeit

Dozent

Prof. Dr.-Ing.
Martin Hoffmann
martin.hoffmann@fh-bielefeld.de