FH Bielefeld
University of
Applied Sciences

Vorlesung

Betriebssysteme

Teil 3

Prozesse

FH Bielefeld
University of
Applied Sciences

Inhalt

- Prozesse, Lebenszyklus
- 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

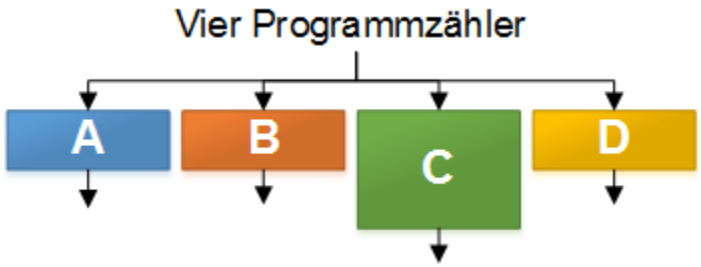
FH Bielefeld
University of
Applied Sciences

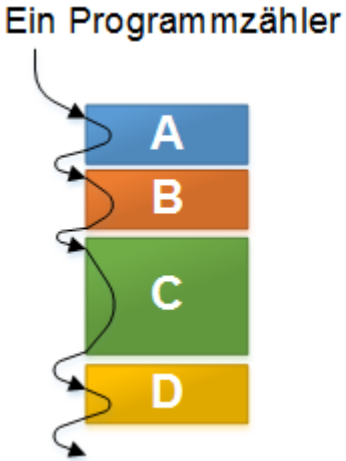
Prozesse

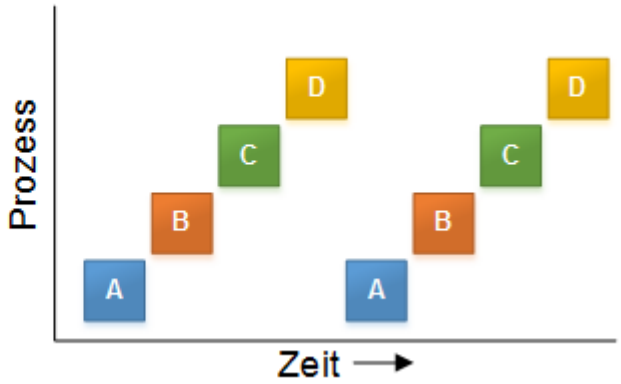
- Programm vs. Prozess
- In den meisten Betriebssystemen
  - laufen **mehrere Programme** auf einem Rechner (Mehrprogrammbetrieb, ***multi-tasking***) simultan und
  - **mehrere Nutzer** teilen sich den Rechner (Mehrbenutzerbetrieb, ***multi-user***).
- Die einzelnen Programme werden vom Betriebssystem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

FH Bielefeld
University of
Applied Sciences

Programmzähler







FH Bielefeld
University of
Applied Sciences

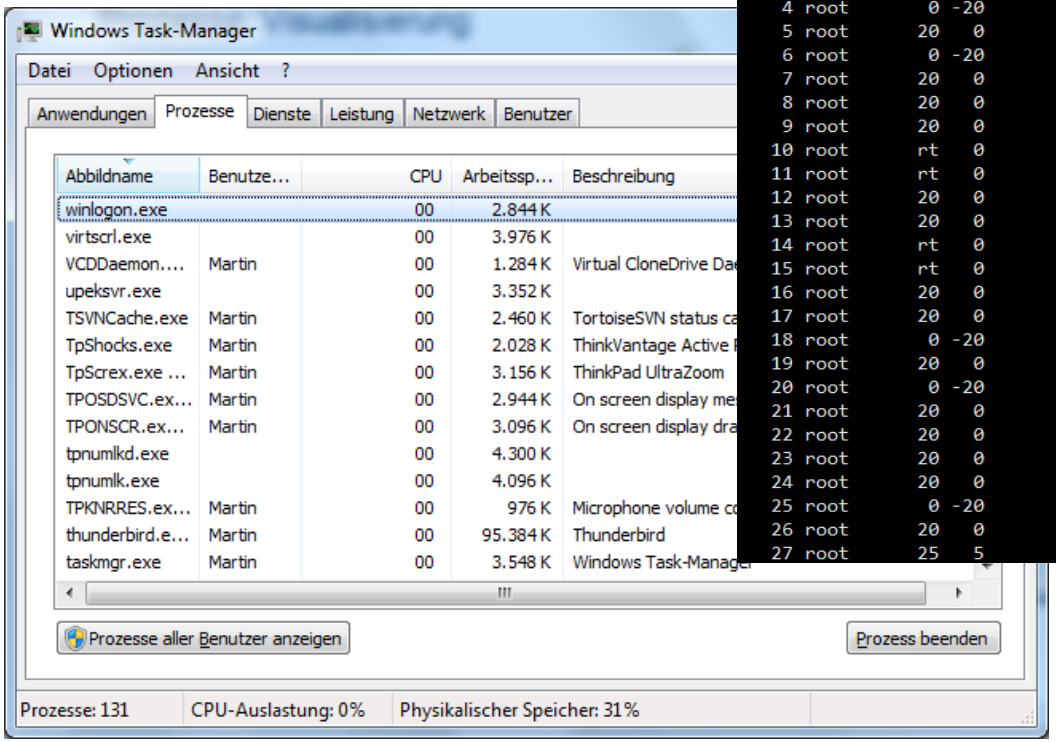
Prozesse: Definition *Prozess*

- 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

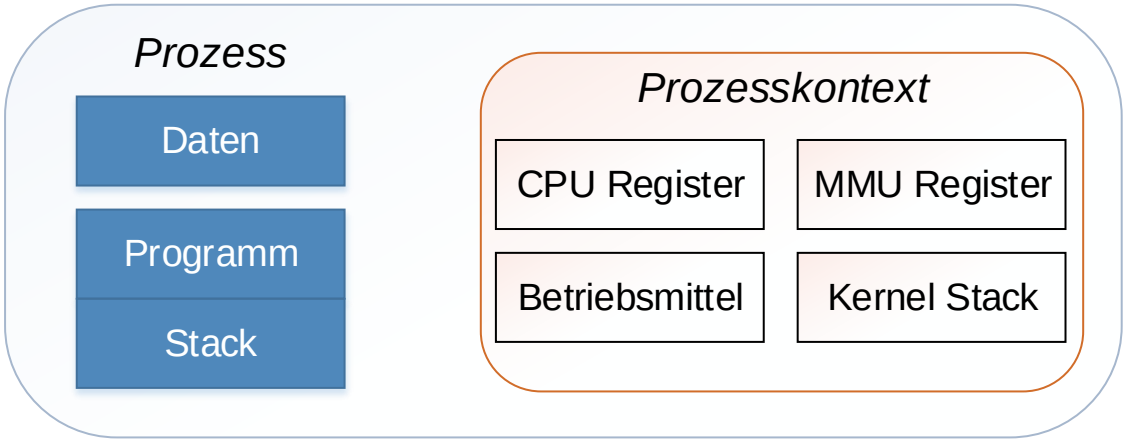
```
top - 08:31:25 up 0 min,  1 user,  load average: 0,32, 0,10, 0,04
Tasks: 173 total,   1 running, 172 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0,0 us,  0,2 sy,  0,0 ni, 99,7 id,  0,2 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem :  3710260 total,  3105888 free,   337392 used,   266980 buff/cache
KiB Swap:  3858428 total,  3858428 free,        0 used.  3123432 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1053 user      20   0 9880992 170692  59752 S   2,0  4,6   0:03.51 ethdcrminer64
    1 root      20   0  119948   6136   4060 S   0,0  0,2   0:01.02 systemd
    2 root      20   0       0      0      0 S   0,0  0,0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S   0,0  0,0   0:00.00 kworker/0:0
    4 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 kworker/0:0H
    5 root      20   0       0      0      0 S   0,0  0,0   0:00.00 kworker/u4:0
    6 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 mm_percpu_wq
    7 root      20   0       0      0      0 S   0,0  0,0   0:00.01 ksoftirqd/0
    8 root      20   0       0      0      0 S   0,0  0,0   0:00.01 rcu_sched
    9 root      20   0       0      0      0 S   0,0  0,0   0:00.00 rcu_bh
   10 root      rt   0       0      0      0 S   0,0  0,0   0:00.00 migration/0
   11 root      rt   0       0      0      0 S   0,0  0,0   0:00.00 watchdog/0
   12 root      20   0       0      0      0 S   0,0  0,0   0:00.00 cpuhp/0
   13 root      20   0       0      0      0 S   0,0  0,0   0:00.00 cpuhp/1
   14 root      rt   0       0      0      0 S   0,0  0,0   0:00.00 watchdog/1
   15 root      rt   0       0      0      0 S   0,0  0,0   0:00.00 migration/1
   16 root      20   0       0      0      0 S   0,0  0,0   0:00.01 ksoftirqd/1
   17 root      20   0       0      0      0 S   0,0  0,0   0:00.00 kworker/1:0
   18 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 kworker/1:0H
   19 root      20   0       0      0      0 S   0,0  0,0   0:00.00 kdevtmpfs
   20 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 netns
   21 root      20   0       0      0      0 S   0,0  0,0   0:00.02 kworker/0:1
   22 root      20   0       0      0      0 S   0,0  0,0   0:00.01 kworker/1:1
   23 root      20   0       0      0      0 S   0,0  0,0   0:00.00 khungtaskd
   24 root      20   0       0      0      0 S   0,0  0,0   0:00.00 oom_reaper
   25 root       0 -20       0      0      0 S   0,0  0,0   0:00.00 writeback
   26 root      20   0       0      0      0 S   0,0  0,0   0:00.00 kcompactd0
   27 root      25   5       0      0      0 S   0,0  0,0   0:00.00 ksmd
```



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

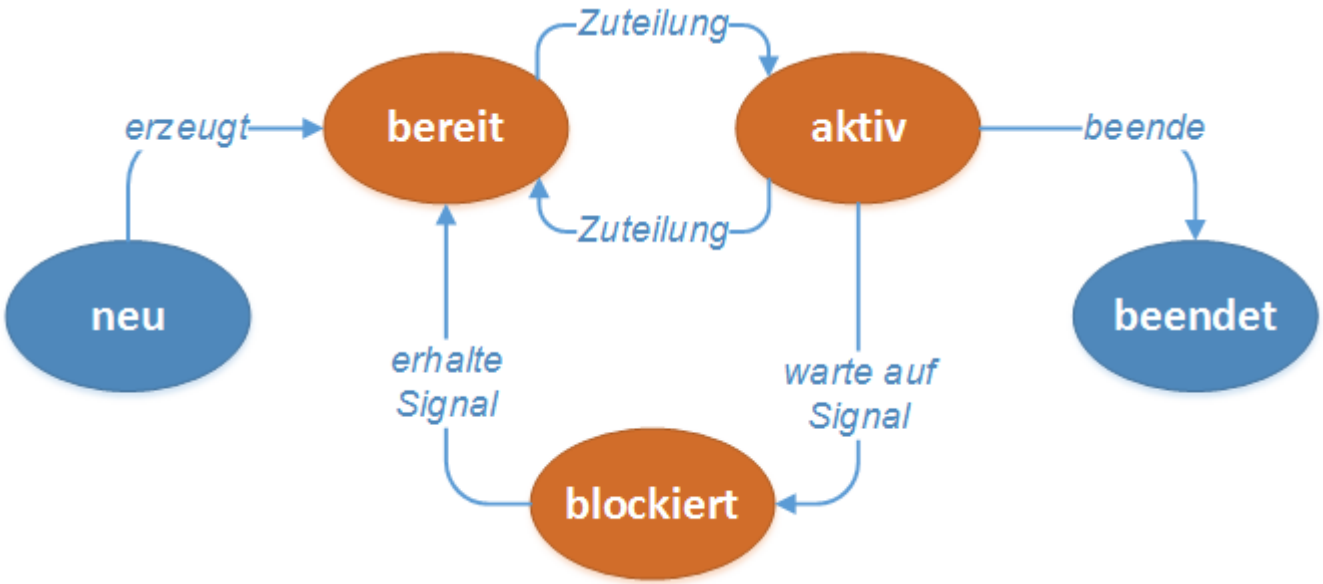
Prozesse: Zustände

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 in diesem Zustand 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.

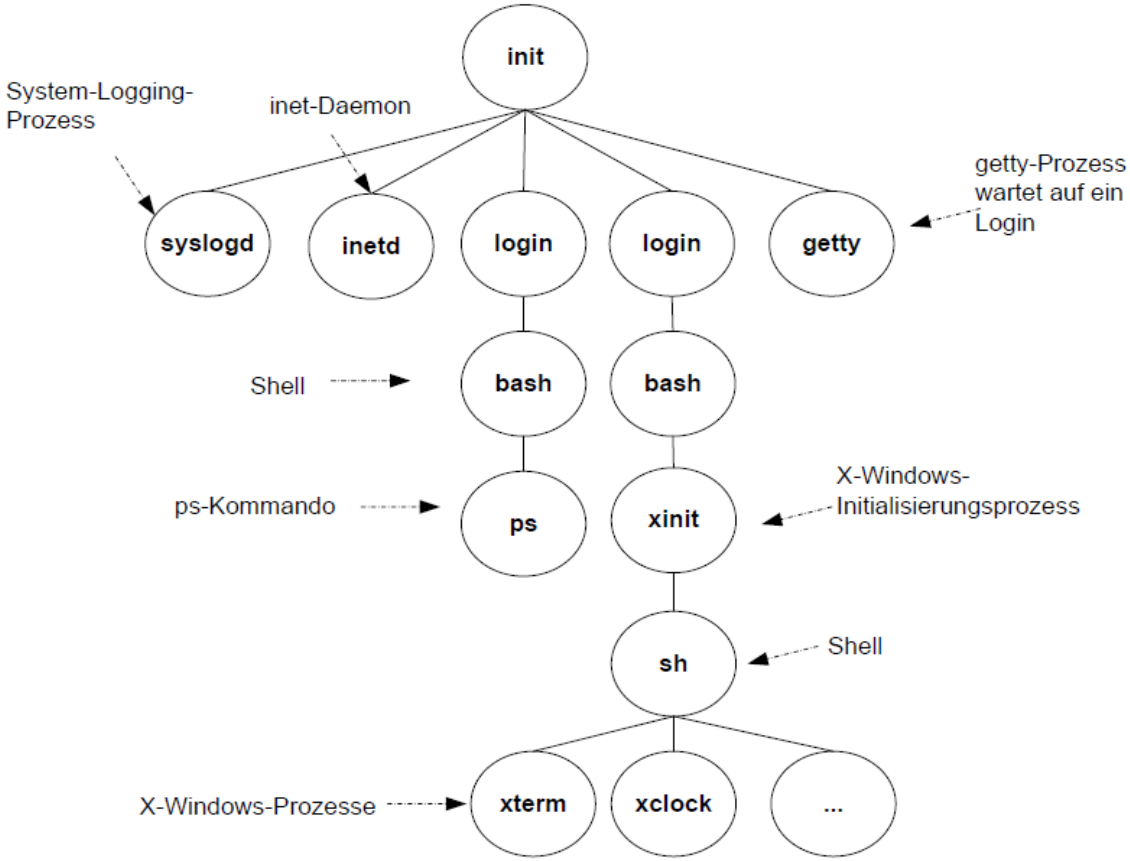
Prozesse: Erzeugung (Forts.)

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

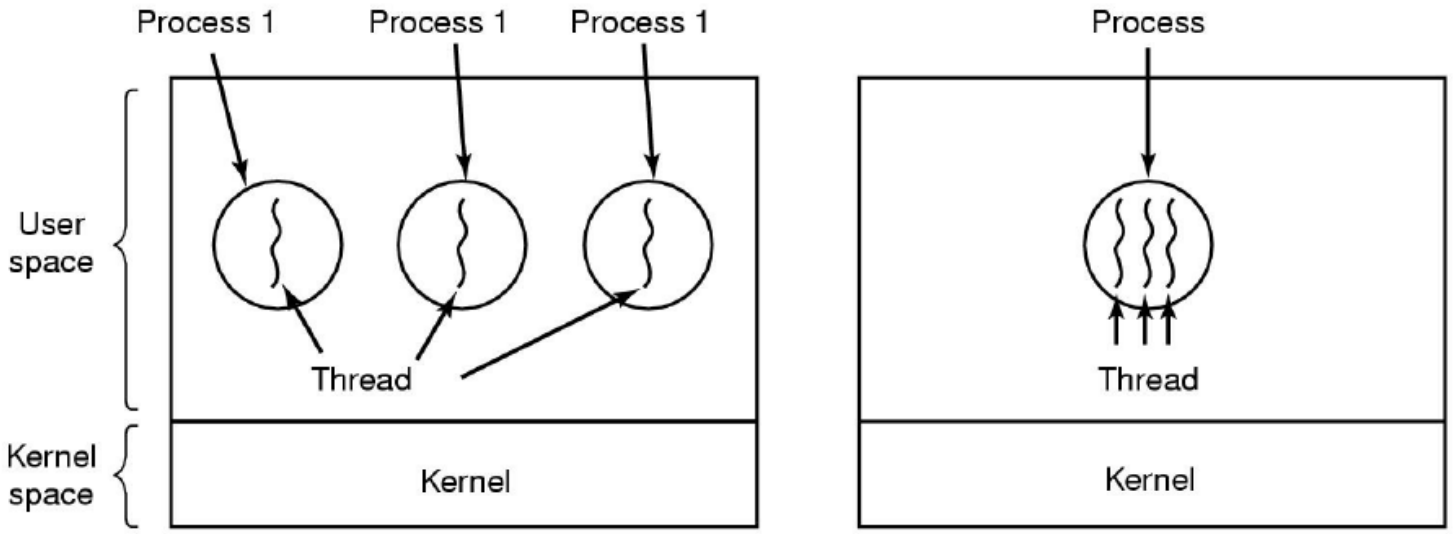
Inhalt

- Prozesse und Lebenszyklus von Prozessen
- Threads
- Scheduling

FH Bielefeld
University of
Applied Sciences

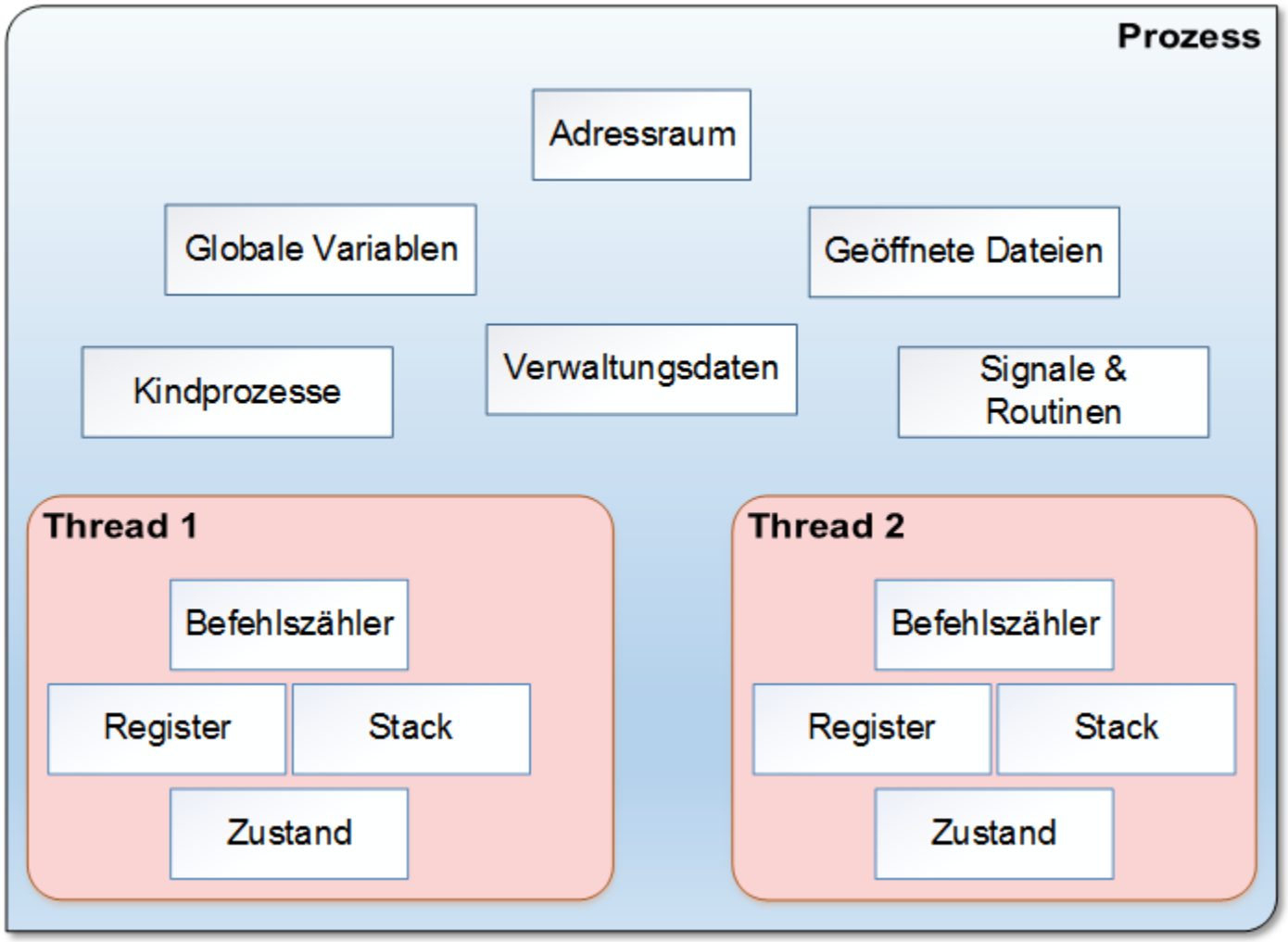
Prozesse: Leichtgewichtige Prozesse (Threads)

- **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

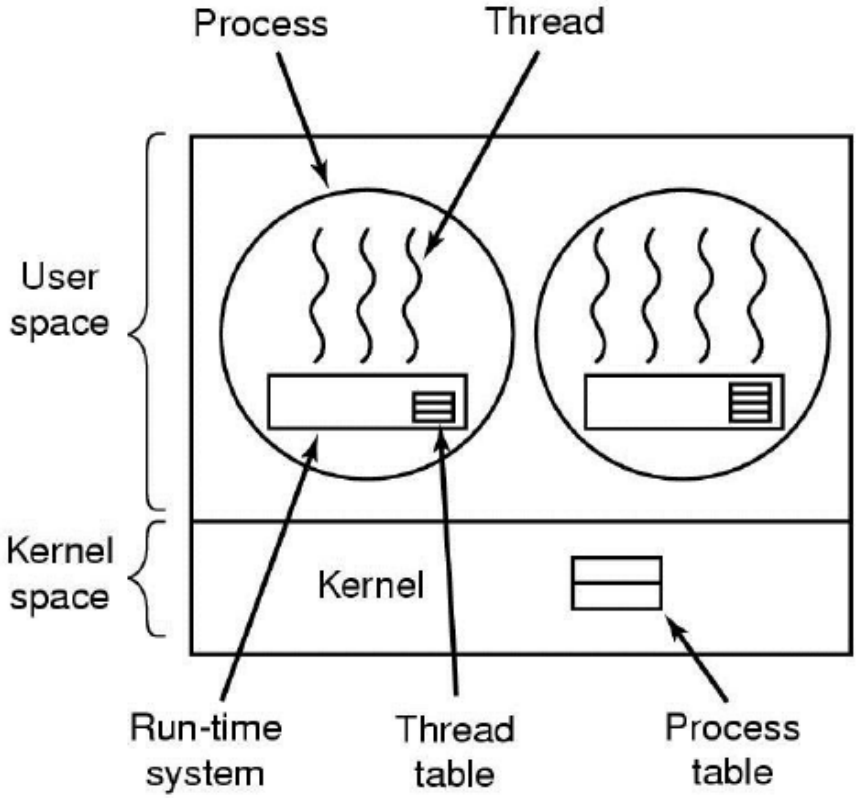
- **Kontextwechsel** zwischen Threads ist **effizienter** als zwischen Prozessen:
  - Kein Umschalten zwischen den Adressräumen.
  - Kein Retten und Restaurieren des Kontextes (nur Programmzähler, Register und Stackpointer)

  → 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**, ….

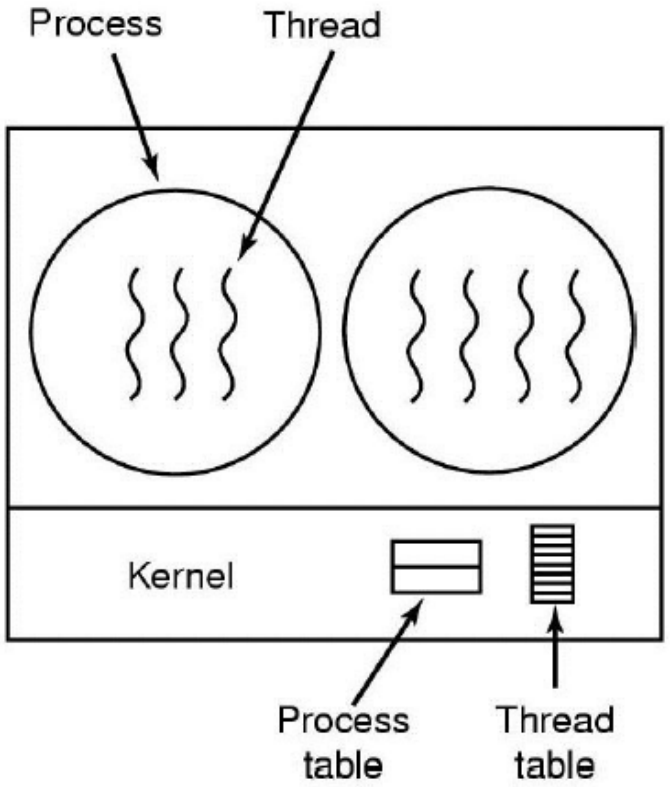
FH Bielefeld
University of
Applied Sciences

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

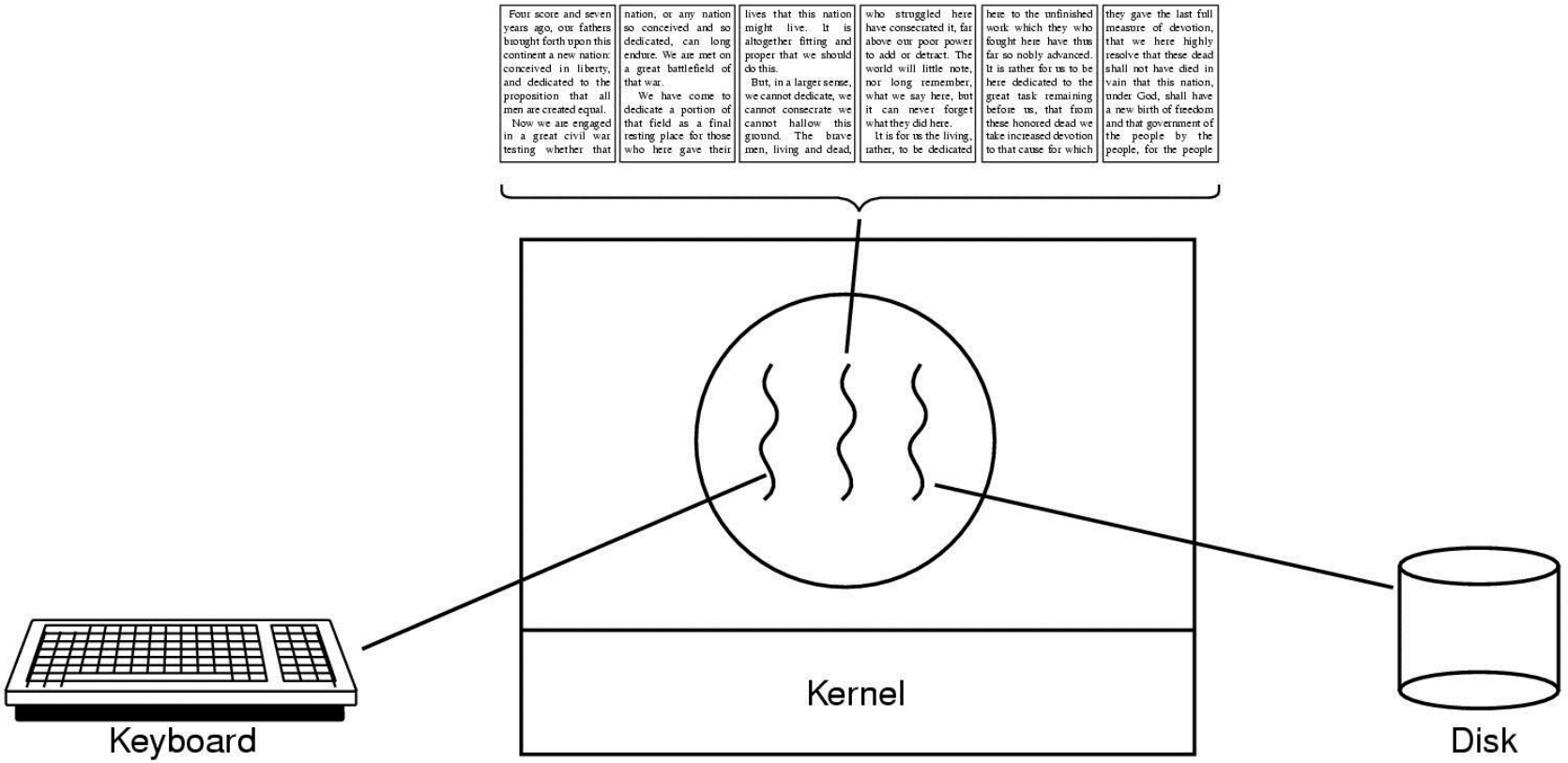
Threads werden genutzt, um:

- 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

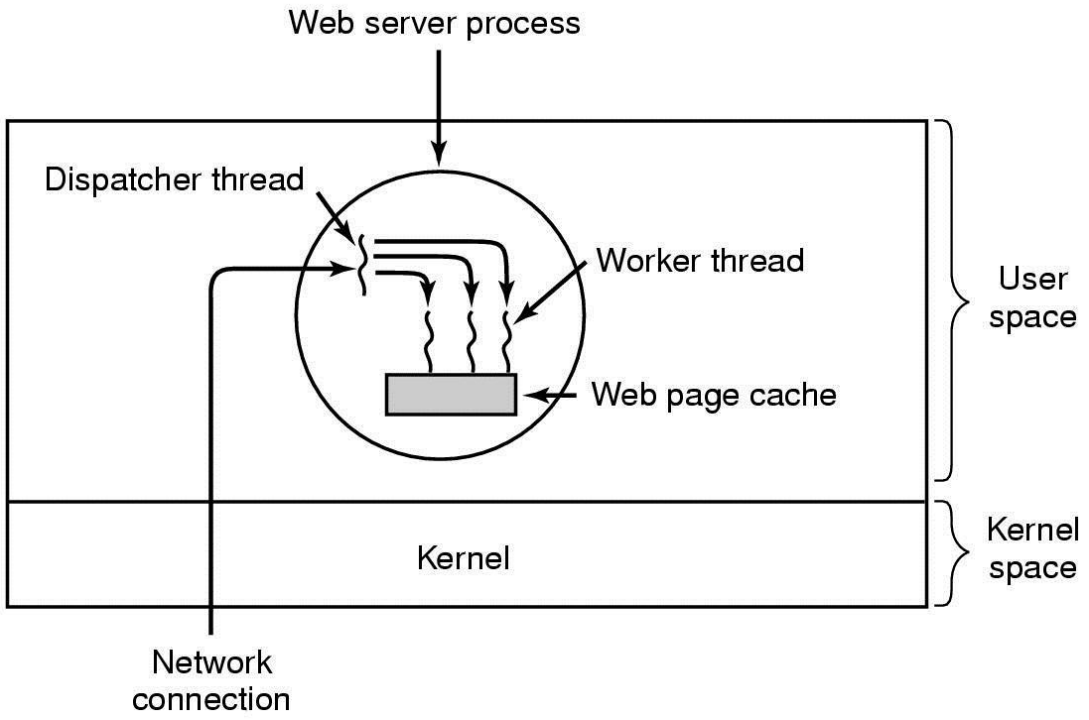
Threads: Anwendung

Zum Beispiel Textverarbeitungsprogramm mit 3 Threads:



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
- 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**)

Scheduling: Begriffe

Begriffe im Zusammenhang mit dem Scheduling:

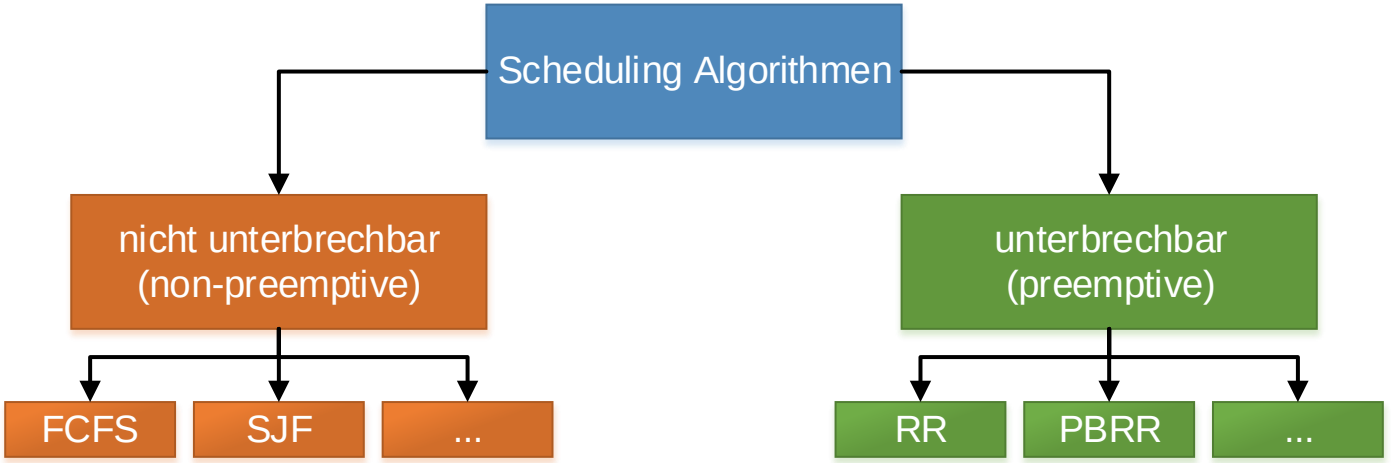
- **Ankunftszeit** eines Prozesses P_i: $T_{a,i}$
- **Startzeit** eines Prozesses P_i: $T_{s,i}$
- **Unterbrechungszeit** eines Prozessen P_i: $T_{u,i}$
- **Rechenzeit** eines Prozesses P_i: T_i

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 Prozessen: $T_{\tilde{W}} = \frac{1}{n} \sum_{i=1}^{n} T_{w,i}$
- **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

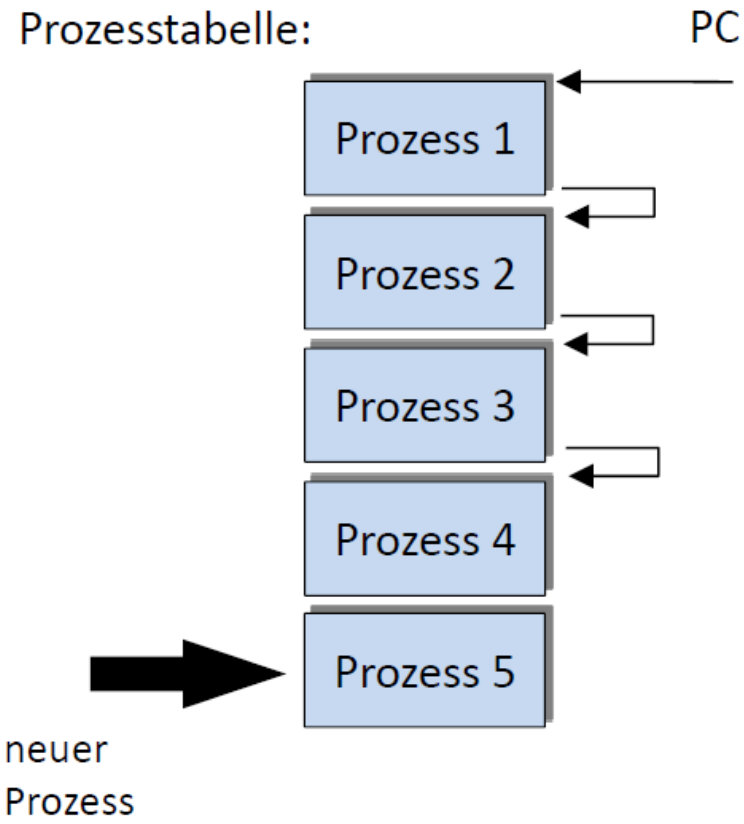
Scheduling: Algorithmen



FH Bielefeld
University of
Applied Sciences

Scheduling Algorithmen: FCFS

First-Come First-Serve (FCFS)

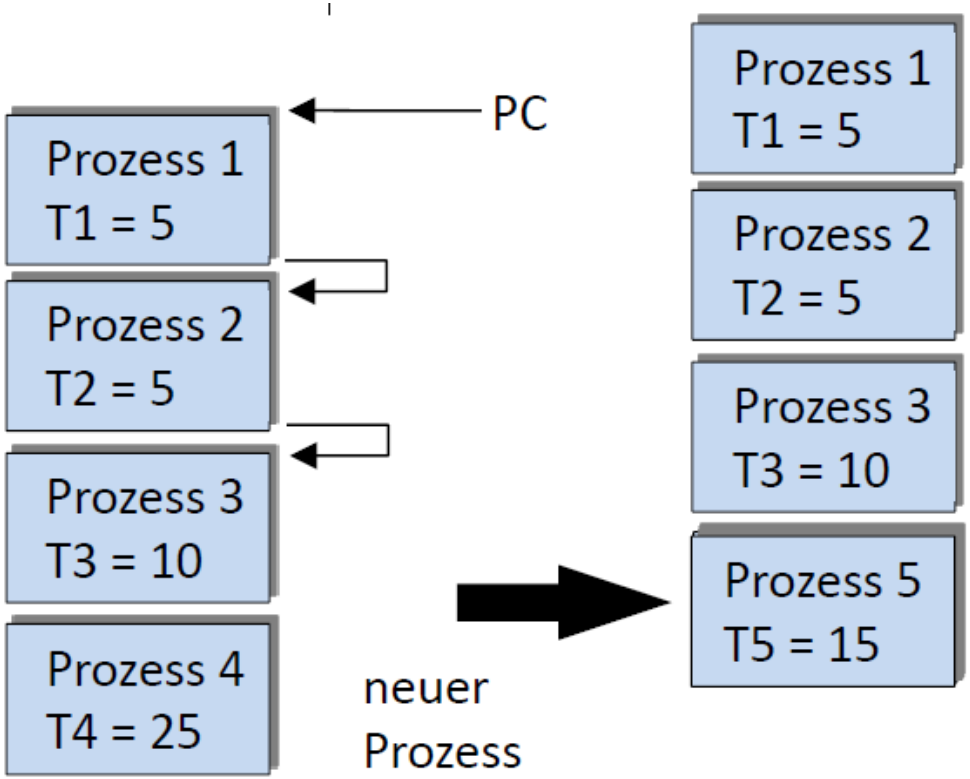


- Prozesse sind nicht unterbrechbar (non-preemptive)
  $$T_{u,i} = 0$$
- Wartezeit für Prozess P_i:
  $$T_{w,i} = \sum_{j=1}^{i-1} T_j$$
- Verweilzeit für Prozess P_i:
  $$T_{v,i} = T_{w,i} + T_i = \sum_{j=1}^{i} T_j$$
- Durchschnittliche Verweilzeit:
  $$T_{\tilde{V}} = \frac{1}{n} \sum_{i=1}^{n} (n+1-i)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

FH Bielefeld
University of
Applied Sciences

Scheduling Algorithmen: SJF

- Shortest Job First (SJF)
- Prozesstabelle:



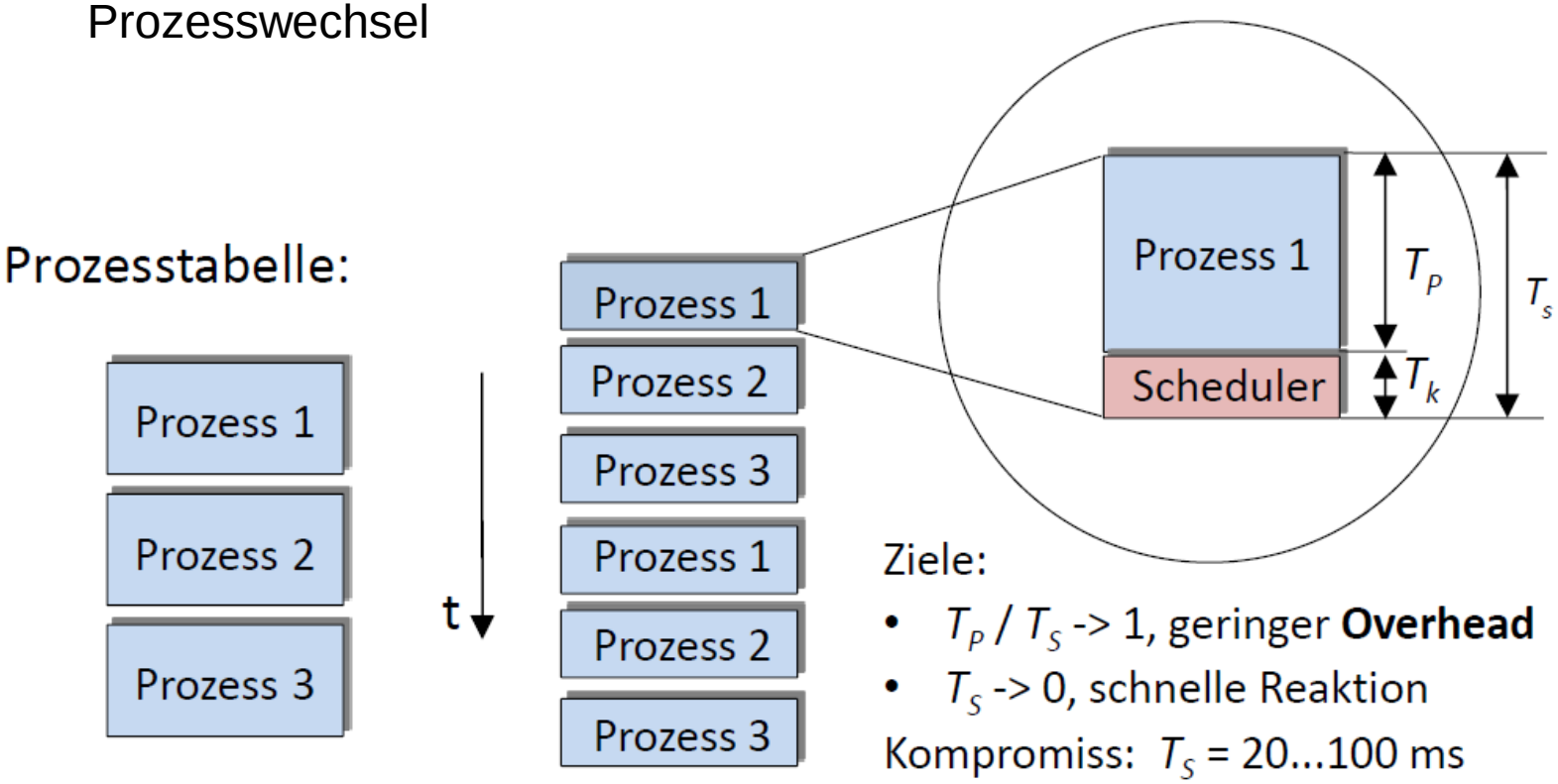
- FCFS und SJF sind nicht unterbrechbar.

  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

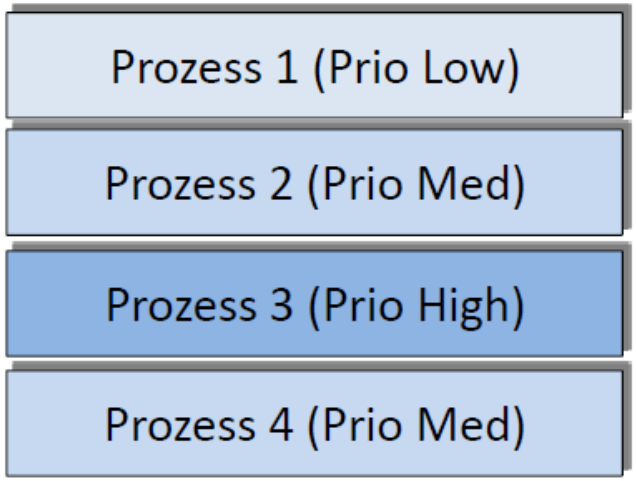


FH Bielefeld
University of
Applied Sciences

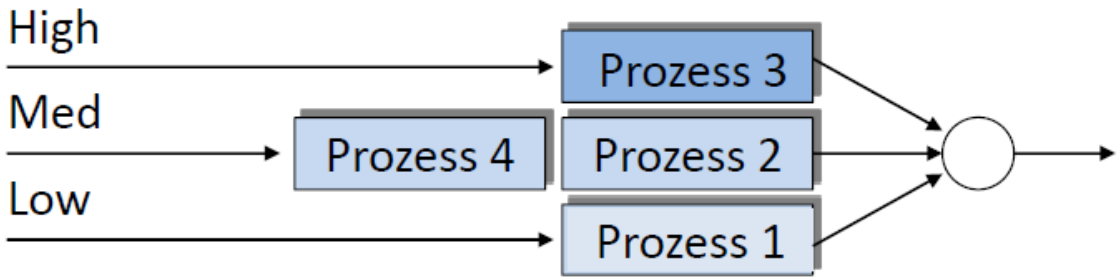
Scheduling Algorithmen: PBRR

- Prioritätenbasiertes Round Robin (PBRR)

Prozesstabelle:



Warteschlangen:



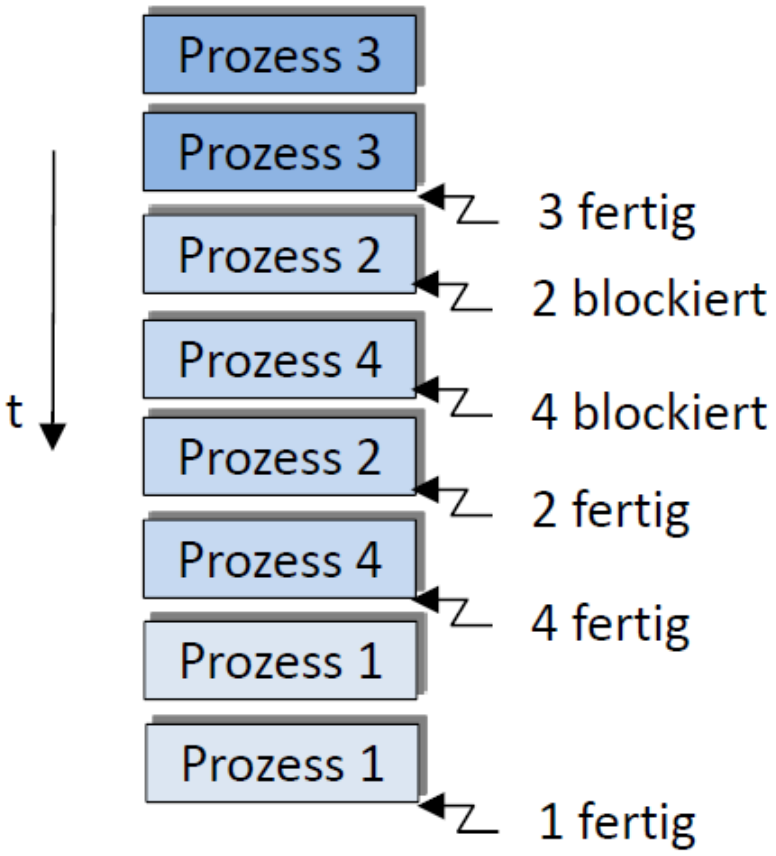
FH Bielefeld
University of
Applied Sciences

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:



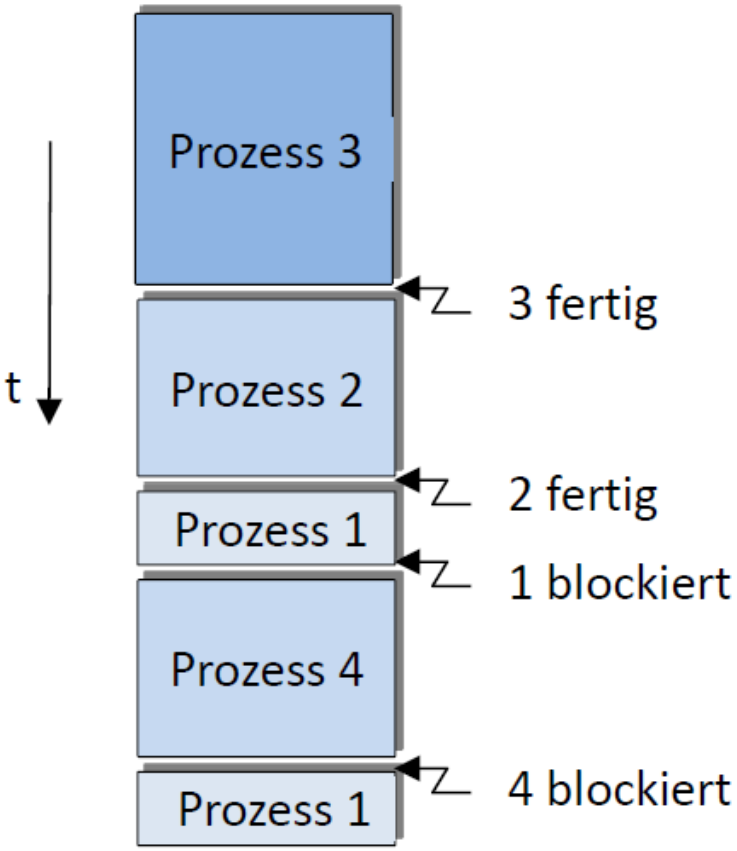
FH Bielefeld
University of
Applied Sciences

Scheduling Algorithmen: PBRR

Variante II:

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

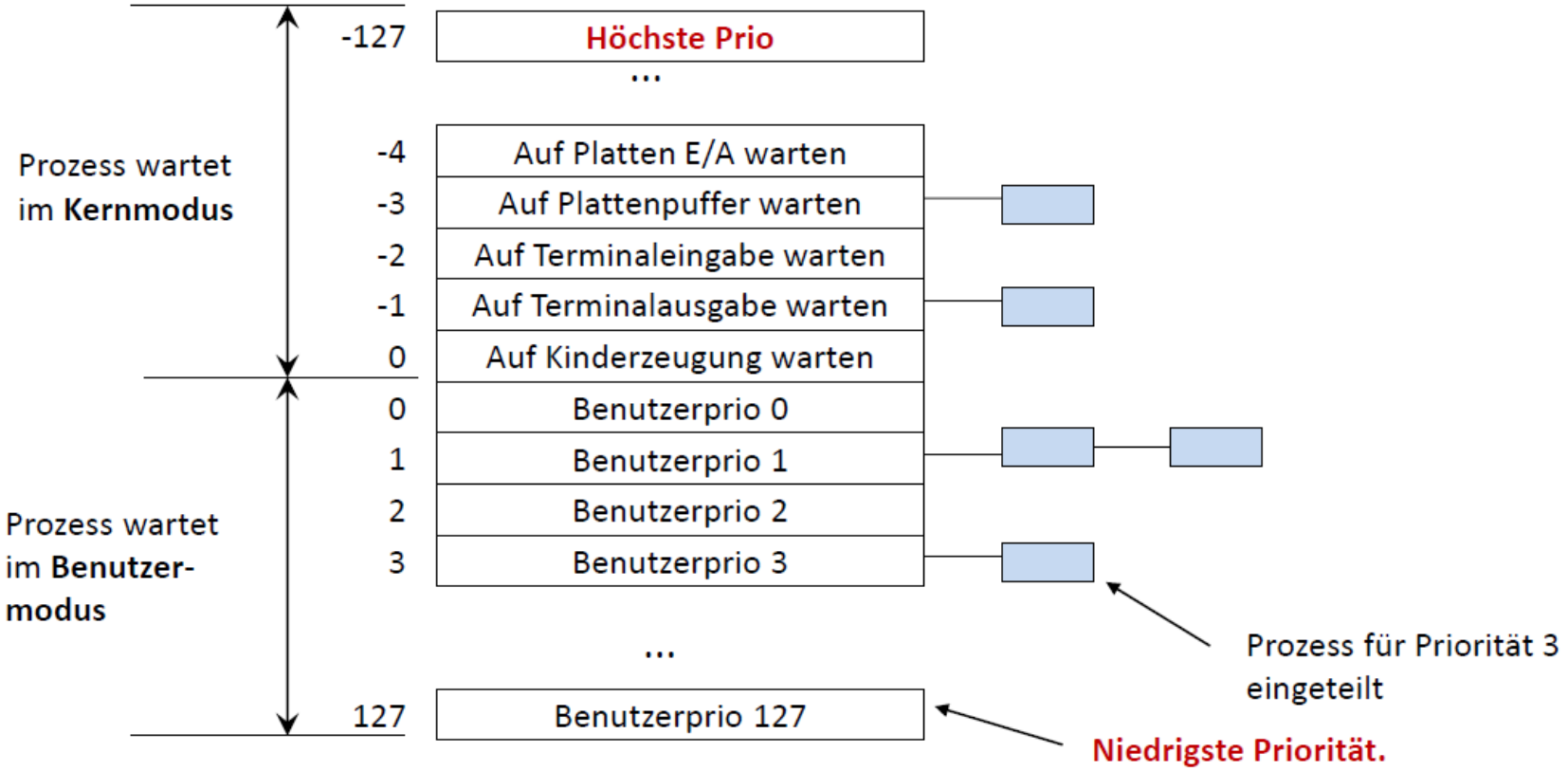
Beispiel für Zeitablauf:



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

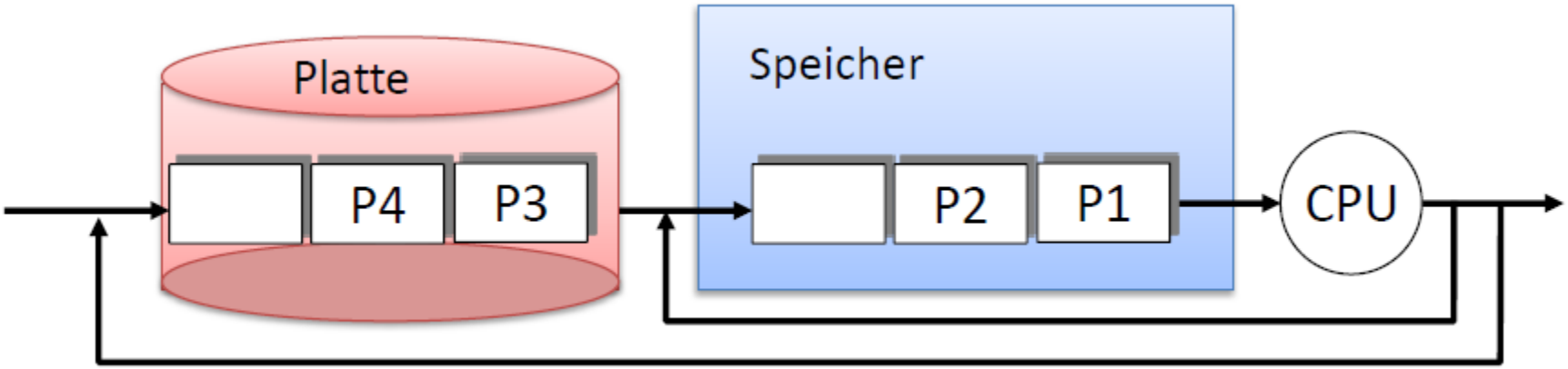
- Manchmal wird auch mit **variablen Zeitscheiben** gearbeitet:
  - innerhalb einer Warteschlange: *Round Robin*
  - bei *niedrigerer* Priorität: *längeres* Quantum
  - Falls Prozess Quantum aufgebraucht: erniedrigen der Priorität
  - CPU-lastiger Prozess erhält längeres Quantum, wird seltener unterbrochen
- **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

Multi-Level-Scheduling

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

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

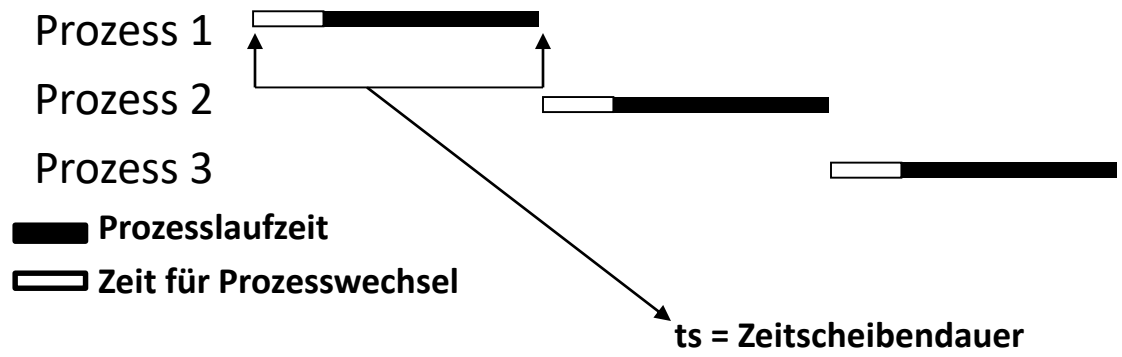
- **CPU Scheduling** (*Kurzzeitscheduling*): Es wird zwischen den Prozessen im Hauptspeicher ausgewählt.
- **Speicher Scheduling** (*Langzeitscheduling*): Verschiebt Prozesse zwischen Hauptspeicher und Platte, so dass alle Prozesse eine Chance haben, irgendwann im Speicher zu sein und ausgeführt zu werden.



FH Bielefeld
University of
Applied Sciences

Testfrage

Erläuterung:



- 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

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