FH Bielefeld
University of
Applied Sciences

Vorlesung

Betriebssysteme

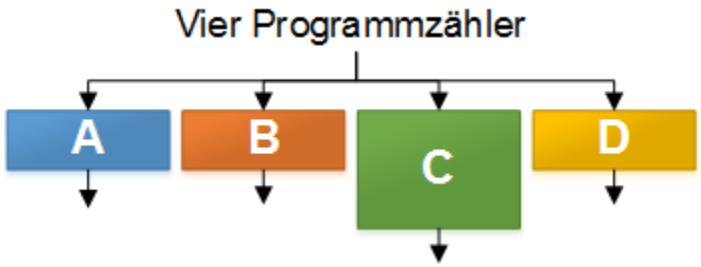
Teil 4

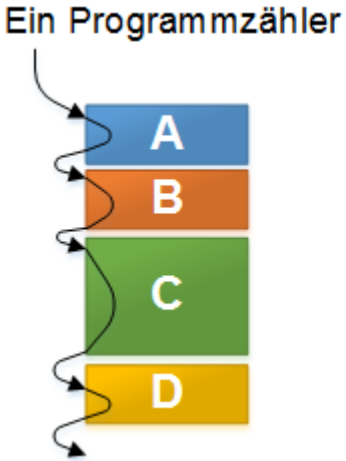
Scheduling und Interprozesskommunikation

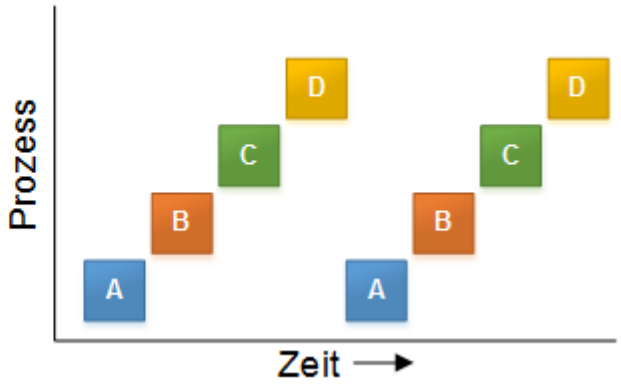
Synchronisation

FH Bielefeld
University of
Applied Sciences

Programmzähler







FH Bielefeld
University of
Applied Sciences

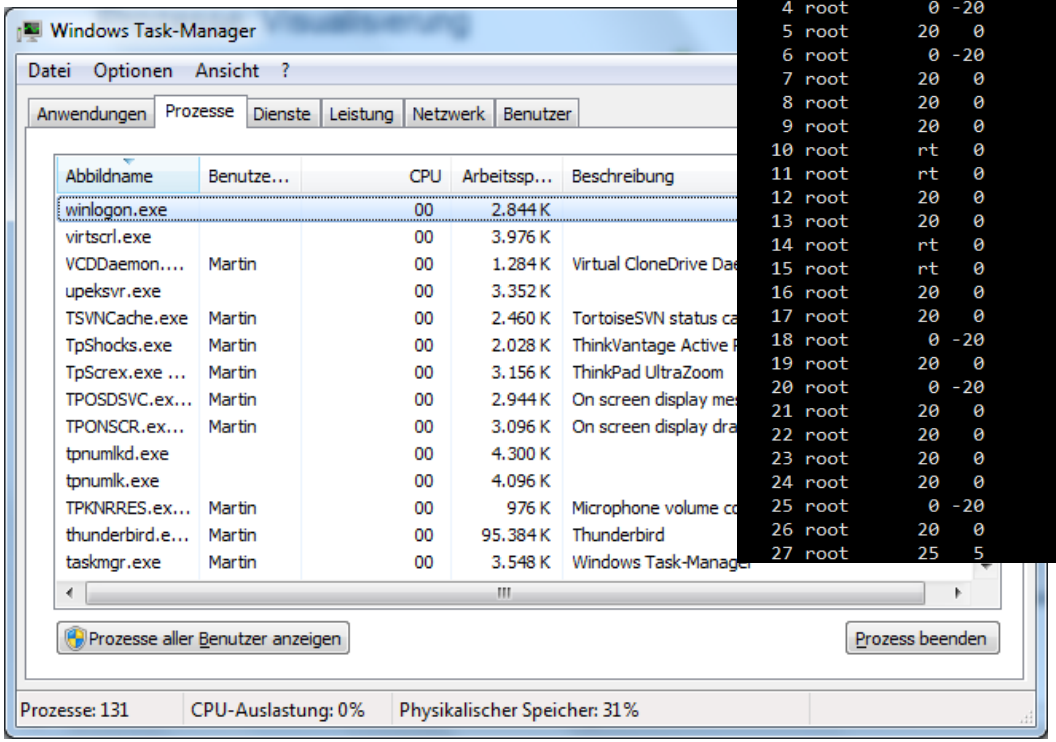
Prozesse: Definition *Prozess*

- 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
```



Wiederholung 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

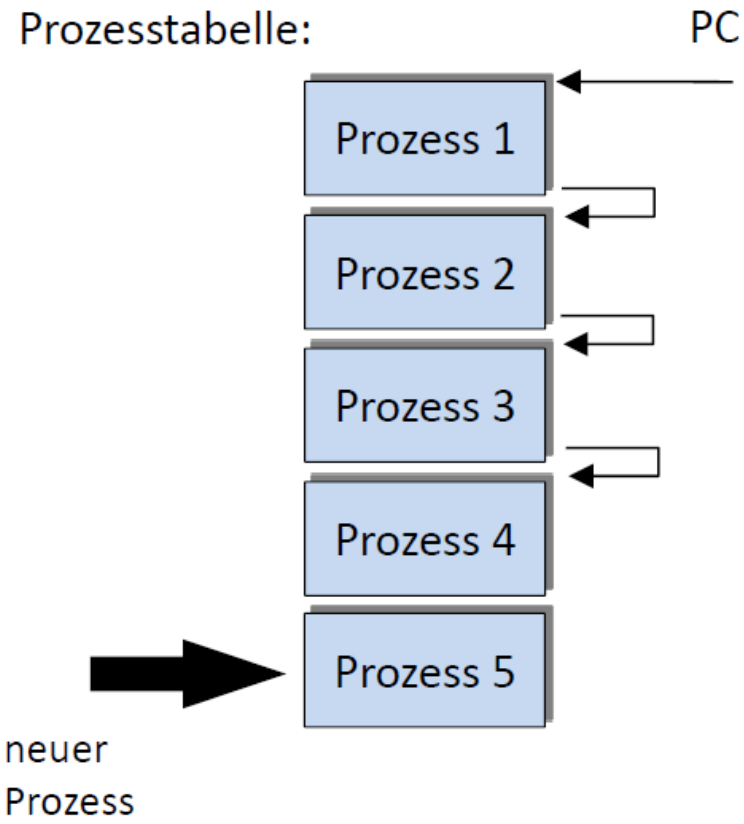
Abgeleitete Größen:

- **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: FCFS

First-Come First-Serve (FCFS)

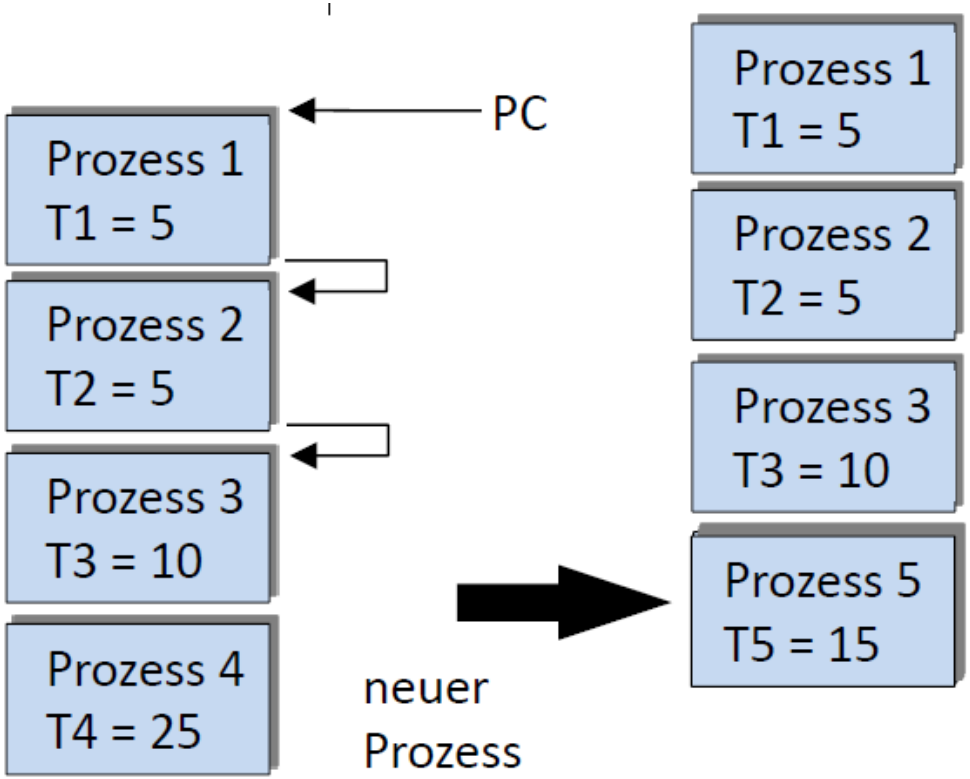


- Prozesse sind nicht unterbrechbar (non-preemptive)
  $$T_{u,i} = 0$$
- Wartezeit für Prozess P_i:
  $$T_{w,i} = \sum_{j=1}^{i-1} T_j$$
- Verweilzeit für Prozess P_i:
  $$T_{v,i} = T_{w,i} + T_i = \sum_{j=1}^{i} T_j$$
- Durchschnittliche Verweilzeit:
  $$T_{\tilde{V}} = \frac{1}{n} \sum_{i=1}^{n} (n+1-i)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

FH Bielefeld
University of
Applied Sciences

Scheduling Algorithmen: SJF

- Shortest Job First (SJF)
- Prozesstabelle:



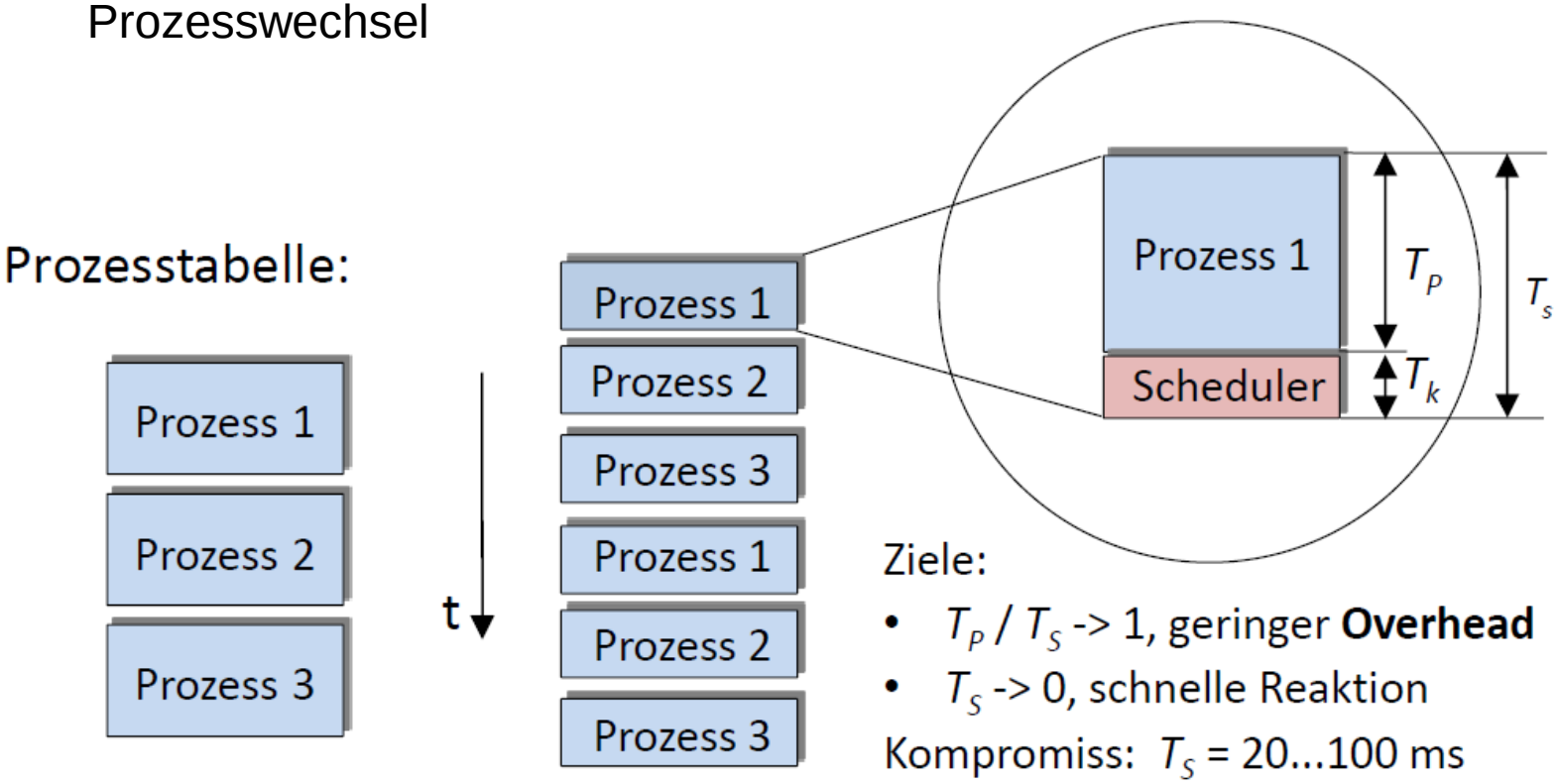
- FCFS und SJF sind nicht unterbrechbar.

  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

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:

$$6\text{min} \cdot \frac{n}{\text{Priorität}=3} = 16\text{min} \qquad 8\text{min} \cdot \frac{n}{\text{Priorität}=4} = 16\text{min}$$

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

Übungsaufgabe

FH Bielefeld
University of
Applied Sciences

Inhalt

- Interprozesskommunikation
  - Pipes, IPC, Shared Memory
- Mutexe und Semaphore
  - Schutz von kritischen Abschnitten
- Verklemmungen
  - Modellieren
  - Erkennen und Beheben
  - Verhindern (Banker Algorithm)

IPC - Ziele

- Interprozesskommunikation kennenlernen
  - Beispiel Pipes
  - Shared memory
  - Sockets

- Race Condition erklären können
  - Mutex und Semaphore erklären und anwenden können

Interprozesskommunikation

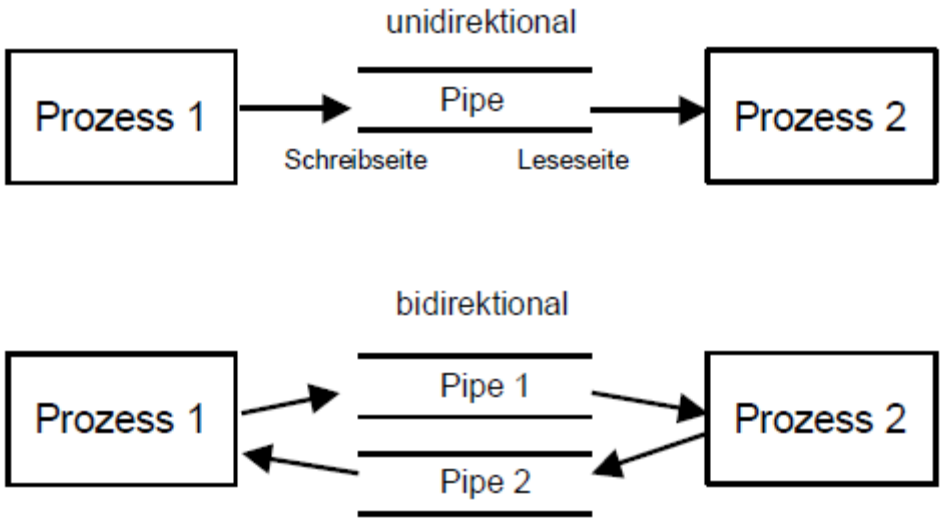
- Unter Unix (je nach Derivat) und Windows gibt es verschiedene IPC-Mechanismen:
  - **Pipes und FIFO‘s** (Named Pipes) als Nachrichtenkanal
  - **Nachrichtenwarteschlangen** (Message Queues)
  - **Gemeinsam genutzter Speicher** (Shared Memory)
  - **Sockets** mit IP-Loopback-Mechanismus

- Ggf. Synchronisationsmechanismen erforderlich:
  - Semaphore und Signale

FH Bielefeld
University of
Applied Sciences

Pipes

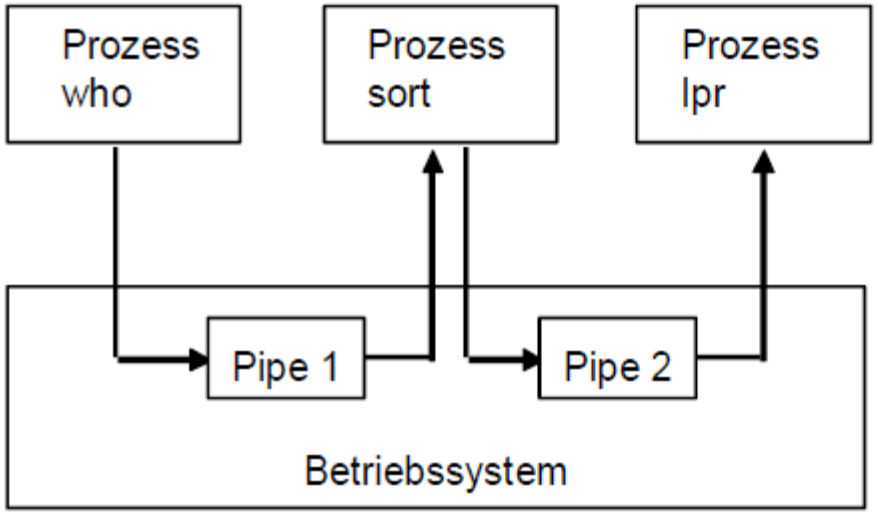
- Pipes: Spezieller unidirektionaler Mechanismus
- Unidirektionale und bidirektionale Kommunikation durch Nutzung mehrerer Pipes
- Bidirektionale Kommunikation über zwei Pipes kann sowohl halb- als auch vollduplex betrieben werden



FH Bielefeld
University of
Applied Sciences

Pipes unter Linux

- Pipes werden u.a. genutzt, um die Standardausgabe eines Prozesses mit der Standardeingabe eines weiteren Prozesses zu verbinden
- Beispiel in Unix: who | sort | lpr



Unix-Kommandos:
who | sort | lpr

FH Bielefeld
University of
Applied Sciences

Wiederholung Dateien

- Einführung Dateiverwaltung
  - **int open(char *name, int how)**
    - Öffnet die Datei zum Lesen oder Schreiben, anhängend oder nicht etc. Die Art des Öffnens ist durch **how** angegeben
  - **int close(int fd)**
  - **int dup(int fd)**
    - Stellt einen neuen Filedeskriptor bereit und weist diesem den Wert des alten zu
  - **int pipe(int *fd)**
    - Zwei Filedeskriptoren werden kreiert, **fd[0]** zum Lesen aus der Pipe und **fd[1]** zum Schreiben in die Pipe.

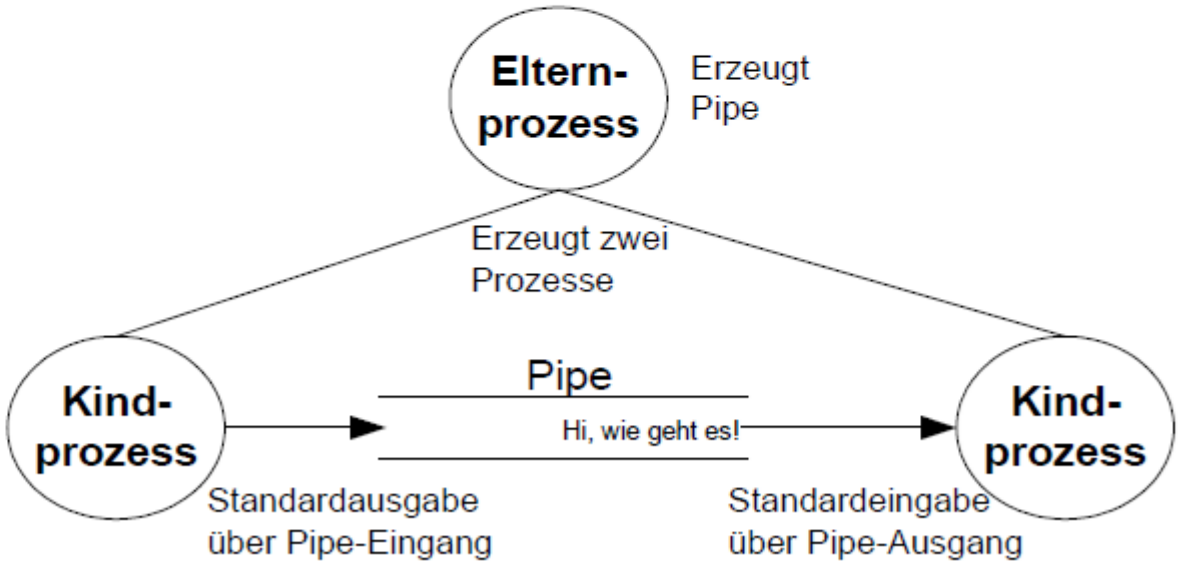
Pipes: Programmierung

- Erzeugen einer Pipe:
  - Unter Unix mit dem Systemaufruf pipe() oder popen()
  - Unter Windows NT mit CreatePipe()
- Schließen einer Pipe:
  - Unter Unix mit dem Systemaufruf close() oder pclose()
  - Unter Windows NT mit closeHandle()
- Elternprozess erzeugt Pipe und vererbt sie an den Kindprozess
- Man kann Pipes blockierend (Normalmodus) und nicht blockierend einsetzen. Blockierend bedeutet:
  - Wenn die Pipe voll ist blockiert der Sendeprozess
  - Wenn die Pipe leer ist blockiert der Leseprozess
  - Sinnvoll für Erzeuger-Verbraucher-Problem

FH Bielefeld
University of
Applied Sciences

Pipes: Beispiel (Teil 1)

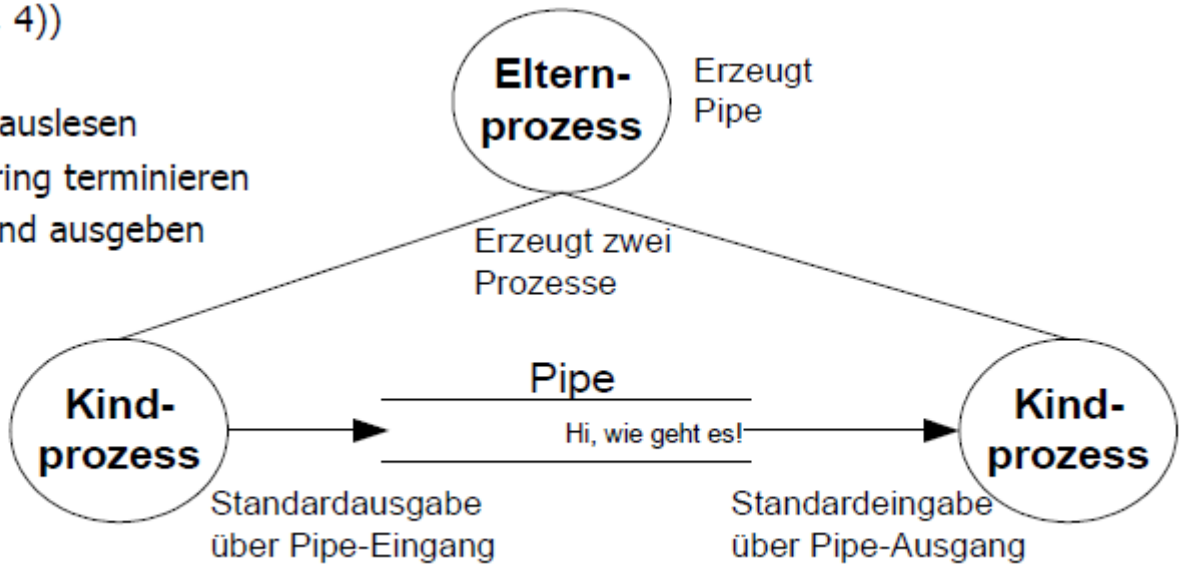
```
int fds[2] //Filedescriptoren für Pipe
...
pipe(fds);
if (fork() == 0) {
      // 1. Kindprozess, Standardausgabe auf Pipe-Schreibseite (Pipe-Eingang) legen
      // und Pipe-Leseseite (Pipe-Ausgang) schließen (wird nicht benötigt)
      dup2(fds[1], 1);  // 1 = Standardausgabe
      close(fds[0]);
      write (1, text, strlen(text)+1);
}
else{ ...
```



FH Bielefeld
University of
Applied Sciences

Pipes: Beispiel (Teil 2)

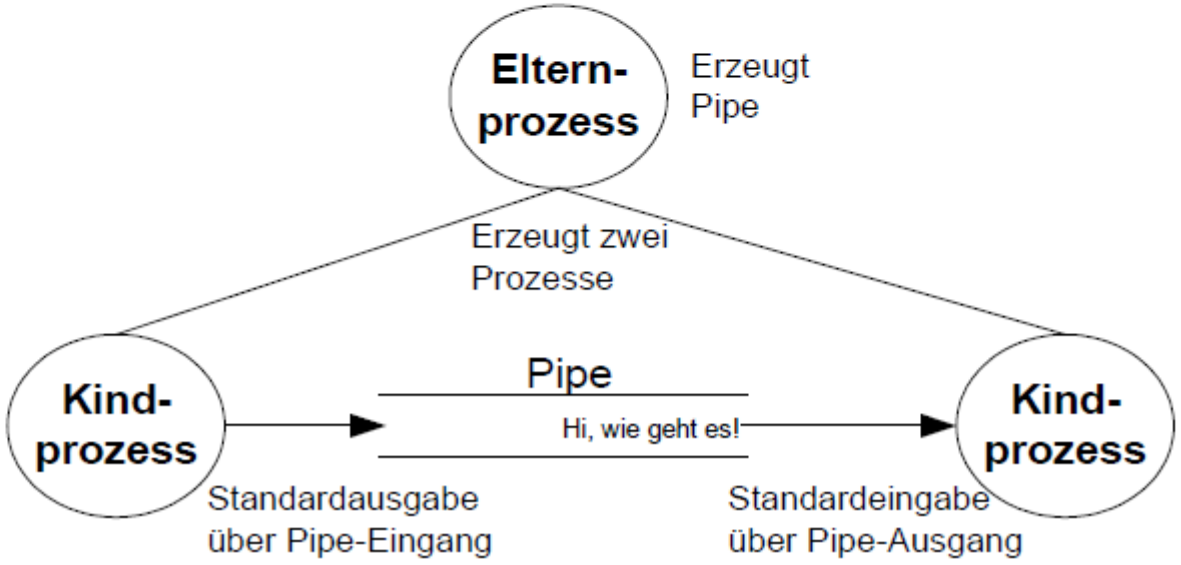
```
else{

   if (fork() == 0) {
        // 2. Kindprozess, Pipe-Leseseite (Pipe-Ausgang) auf
        // Standardeingabe umlenken und Pipe-Schreibseite
        // (Pipe-Eingang) schließen
        dup2(fds[0], 0); // 0 = standardeingabe
        close(fds[1]);
        while (count = read(0, buffer, 4))
        {
              // Pipe in einer Schleife auslesen
              buffer[count] = 0; // String terminieren
              printf("%s", buffer) // und ausgeben
        }
   }
```



FH Bielefeld
University of
Applied Sciences

Pipes: Beispiel (Teil 3)

```
...
      else {
         // Im Vaterprozess: Pipe an beiden Seiten schließen und
         // auf das Beenden der Kindprozesse warten
         close(fds[0]);
         close[fds[1]);
         wait(&status);
         wait(&status);
      }
      exit(0);
   }
}
```



FH Bielefeld
University of
Applied Sciences

Interprozesskommunikation

- *Parallele* Prozesse:
  - Rechner mit **mehreren CPUs** oder Netzwerke aus unabhängigen Rechnern (Multiprozessorsystemen) können mehrere Prozesse **zeitgleich** ausführen
  - Prozesse laufen **unabhängig** von einander (*parallel*)
- *Nebenläufige* Prozesse:
  - Rechner mit **einer CPU** können immer nur einen Prozess bearbeiten
  - Prozesse laufen hintereinander in **beliebiger Reihenfolge** (*nebenläufig*)
  - Parallelität (*Pseudoparallelität*) wird durch Multitasking realisiert

FH Bielefeld
University of
Applied Sciences

Nebenläufige Prozesse

Kommunikation zwischen Prozessen:

- Prozesse arbeiten oft zusammen, um ihre Aufgabe zu erfüllen.
- Dabei stellen sich folgende Fragen:
  - Wie findet der **Austausch** der Daten zwischen den Prozessen statt?
    - Über **gemeinsame Variablen** (gemeinsame Speicherbereiche)?
    - Über **Nachrichtenaustausch** (*Message Passing*)?
  - Wie wird die **Konsistenz** gemeinsam genutzter Daten sichergestellt?
  - Wie wird die richtige **Reihenfolge** beim Zugriff auf gemeinsame Daten sichergestellt?
- Die beiden letzten Fragen führen zum Problem der *Prozess**synchronisation***
  - ***Scheduling beeinflusst Abarbeitungsreihenfolge von Maschinenbefehlen***
  - ***Außerhalb der Kontrolle des Anwendungsentwicklers***

FH Bielefeld
University of
Applied Sciences

Vorlesung

Vielen Dank für Ihre Aufmerksamkeit

Dozent

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