

The slide is a title slide for a presentation. It has a blue header bar at the top with the text 'Betriebssysteme 2023' on the left, 'Prof. Dr. Jörg Mielebacher' in the center, and '1' on the right. The main content area is white and contains the title 'BETRIEBSSYSTEME PROZESSE' in a large, bold, black sans-serif font. Below the title is a horizontal line, followed by the author's name 'Prof. Dr. Jörg Mielebacher' and his email address 'joerg.mielebacher@mosbach.dhbw.de' in a smaller, black sans-serif font.

Betriebssysteme 2023 Prof. Dr. Jörg Mielebacher 1

BETRIEBSSYSTEME PROZESSE

Prof. Dr. Jörg Mielebacher
joerg.mielebacher@mosbach.dhbw.de

Die folgenden Folien führen in Prozesse und ihre Verwaltung durch das Betriebssystem ein. Zu jeder Folie sind Notizenseiten erfasst.

Verbesserungsvorschläge und Fehlerhinweise können Sie gerne an die Adresse joerg.mielebacher@mosbach.dhbw.de senden.

Rechtliche Hinweise: Die Rechte an geschützten Marken liegen bei den jeweiligen Markeninhabern. Alle Rechte an diesen Folien, Notizen und sonstigen Materialien liegen bei ihrem Autor, Jörg Mielebacher. Jede Form der teilweisen oder vollständigen Weitergabe, Speicherung auf Servern oder Nutzung in Lehrveranstaltungen, die nicht von dem Autor selbst durchgeführt werden, erfordert seine schriftliche Zustimmung. Eine schriftliche Zustimmung ist darüber hinaus für jede kommerzielle Nutzung erforderlich. Für inhaltliche Fehler kann keine Haftung übernommen werden.

Wiederholung

- Das Betriebssystem verwaltet die Ressourcen des Computers und stellt sie Anwendungsprogrammen bereit.
- Das Betriebssystem abstrahiert von der tatsächlichen Hardware.
- Das Betriebssystem verwaltet Hardware, Prozesse, Benutzer und Daten.
- Der Übergang von Stapelverarbeitung zum Dialogbetrieb hat die Betriebssysteme stark beeinflusst, außerdem die stark gestiegene Leistungsfähigkeit der Hardware.
- Betriebssysteme lassen sich nach unterschiedlichen Merkmalen einordnen, z.B. Architektur, Anwendungsgebiet, Echtzeitfähigkeit, Betriebsarten usw.
- Es gibt zahlreiche Betriebssysteme für unterschiedlichste Anwendungen.

Folie 3

Betriebssysteme 2023	Prof. Dr. Jörg Mielebacher	3
<p data-bbox="475 589 639 633">Grundlagen</p>		



Wie bereits besprochen, versteht man unter einem Prozess (engl. Task) ein Programm in Ausführung, jeder Prozess ist eine Instanz eines Programms, wobei es zu einem Programm mehrere Prozess geben kann, z.B. wenn dieses mehrfach gestartet wird. Das Betriebssystem erfasst zu jedem Prozess weitere Informationen, z.B. eine eindeutige Prozess-ID – dazu später mehr. Jedem Prozess werden – wie auf den folgenden Folien beschrieben – Ressourcen durch das Betriebssystem zugeteilt, insbesondere einen eigenen virtuellen Speicherbereich und sind hierdurch von anderen Prozessen isoliert.

Ein Programm wiederum ist eine Folge von Anweisungen. Das Programm ist üblicherweise auf einem persistenten Speichermedium des Rechners abgelegt (z.B. auf der Festplatte). Soll das Programm ausgeführt werden, muss es in den Hauptspeicher geladen und als Prozess ausgeführt werden. Programm und Prozess sind also nicht gleichbedeutend.

Ein Thread wiederum ist ein Strang von Anweisungen, der innerhalb eines Prozesses ausgeführt wird. Mehrere Threads können in einem Prozess nebenläufig ausgeführt werden, wodurch Aufgaben parallel und ggf. auf unterschiedlichen Prozessoren oder Prozessorkernen ablaufen können. Jeder Prozess hat mindestens einen Thread. Threads werden oft als leichtgewichtige Prozesse bezeichnet. Alle Threads eines Prozesses teilen sich denselben virtuellen Speicher – mehr dazu später.

Betriebssysteme 2023	Prof. Dr. Jörg Mielebacher	5
----------------------	----------------------------	---

Prozesse

- Untersuchen Sie auf Ihrem Rechner die aktuell ablaufenden Prozesse. Wie gehen Sie vor?

Betriebssysteme 2023 Prof. Dr. Jörg Mielebacher 6

Prozesse unter Windows

Task-Manager
Datei Optionen Ansicht

Prozesse Leistung App-Verlauf Autostart Benutzer Details Dienste

Name	PID	Status	Benutzername	CPU	Arbeitsspei...	UAC-Virtualis...
firefox.exe	1844	Wind ausgeführt	jm	00	33.028 K	Deaktiviert
firefox.exe	3560	Wind ausgeführt	jm	00	35.384 K	Deaktiviert
firefox.exe	9040	Wind ausgeführt	jm	00	25.368 K	Deaktiviert
firefox.exe	9692	Wind ausgeführt	jm	00	9.332 K	Deaktiviert
FMService64.exe	5624	Wind ausgeführt	SYSTEM	00	860 K	Nicht zugelas...
Fontdrshost.exe	1176	Wind ausgeführt	UMFD-0	00	9.672 K	Deaktiviert
Greenshot.exe	17782	Wind ausgeführt	jm	00	20.628 K	Deaktiviert
grm-mini.exe	14996	Wind ausgeführt	SYSTEM	00	5.528 K	Nicht zugelas...
gryncit.exe	16496	Wind ausgeführt	jm	00	29.736 K	Deaktiviert
ibmpmavc.exe	3182	Wind ausgeführt	SYSTEM	00	1.020 K	Nicht zugelas...
igfxCISService.exe	3756	Wind ausgeführt	SYSTEM	00	968 K	Nicht zugelas...
igfxEM.exe	8164	Wind ausgeführt	jm	00	3.372 K	Deaktiviert
igfxext.exe	15264	Wind ausgeführt	jm	00	1.624 K	Deaktiviert
ired.exe	6048	Wind ausgeführt	SYSTEM	00	968 K	Nicht zugelas...
IntelAudioService.exe	6128	Wind ausgeführt	SYSTEM	00	7.264 K	Nicht zugelas...
IntelCpHdCPSvc.exe	2268	Wind ausgeführt	SYSTEM	00	708 K	Nicht zugelas...
IntelCpHdCPSvc.exe	2512	Wind ausgeführt	SYSTEM	00	788 K	Nicht zugelas...
ipsec.exe	6112	Wind ausgeführt	SYSTEM	00	812 K	Nicht zugelas...
ipsec.exe	6120	Wind ausgeführt	SYSTEM	00	562 K	Nicht zugelas...
lsass.exe	0	Wind ausgeführt	SYSTEM	91	8 K	Nicht zugelas...
LITSvc.exe	3184	Wind ausgeführt	SYSTEM	00	744 K	Nicht zugelas...
LMS.exe	5180	Wind ausgeführt	SYSTEM	00	2.116 K	Nicht zugelas...
lsass.exe	14996	Ansehen	jm	91	8 K	Deaktiviert

Weniger Details Task beenden

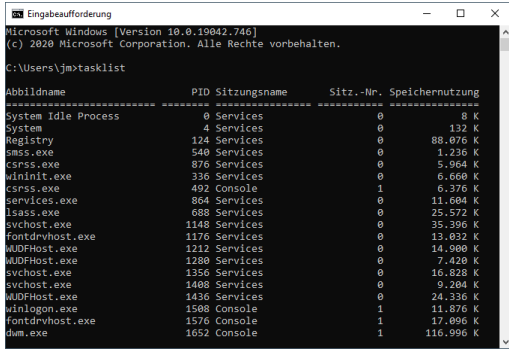
Task-Manager
Ctrl+Shift+Esc

Unter Windows kann man jederzeit den Task-Manager öffnen, z.B. durch Ctrl+Shift+Esc. Standardmäßig wird nur eine verkürzte Darstellung verwendet, die man mit „Mehr Details“ hilfreicher gestalten kann. Insbesondere unter „Details“ hat man einen sehr guten Überblick über die laufenden Prozesse, der man z.B. die eindeutige Prozess-ID, die Arbeitsspeichernutzung oder den ausführenden Benutzer entnehmen kann. Als Dienste werden hierbei alle Prozesse bezeichnet, die im Hintergrund ausgeführt werden – aber auch bei Ihnen handelt es sich um Prozesse im Sinne der Definition.

In der SysInternals-Suite gibt es mit dem Process Explorer einen wesentlich leistungsfähigeren Task-Manager.

Betriebssysteme 2023 Prof. Dr. Jörg Mielebacher 7

Prozesse unter Windows



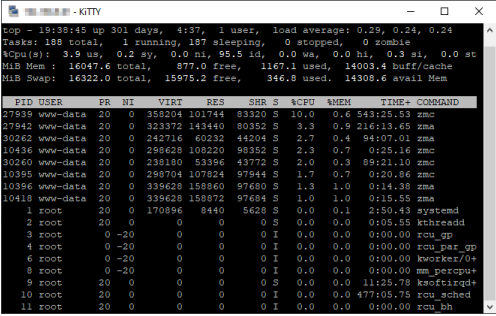
tasklist

Abbildname	PID	Sitzungsname	Sitz.-Nr.	Speichernutzung
System	0	Services	0	0 K
System	4	Services	0	132 K
Registry	124	Services	0	88,076 K
smss.exe	540	Services	0	1,236 K
csrss.exe	876	Services	0	5,864 K
wininit.exe	336	Services	0	6,660 K
csrss.exe	492	Console	1	6,376 K
services.exe	864	Services	0	11,604 K
lsass.exe	680	Services	0	25,572 K
svchost.exe	1148	Services	0	35,396 K
fontdrvhost.exe	1176	Services	0	13,032 K
WUDFHost.exe	1212	Services	0	14,900 K
WUDFHost.exe	1280	Services	0	7,420 K
svchost.exe	1356	Services	0	16,828 K
svchost.exe	1408	Services	0	9,204 K
WUDFHost.exe	1436	Services	0	24,336 K
winlogon.exe	1508	Console	1	11,876 K
fontdrvhost.exe	1576	Console	1	17,096 K
dwm.exe	1652	Console	1	116,996 K

Möchte man z.B. in Skripten Informationen über laufende Prozesse abrufen, bietet sich der Befehl `tasklist` an. Hiermit könnte man z.B. prüfen, ob ein bestimmter Prozess noch läuft.

Betriebssysteme 2023 Prof. Dr. Jörg Mielebacher 8

Prozesse unter Linux



```

top - 19:38:45 up 301 days, 4:37, 1 user, load average: 0.29, 0.24, 0.24
Tasks: 188 total, 1 running, 187 sleeping, 0 stopped, 0 zombie
%Cpu(s): 3.9 us, 0.2 sy, 0.0 ni, 95.5 id, 0.0 wa, 0.0 hi, 0.3 si, 0.0 st
MiB Mem : 16047.6 total, 877.0 free, 1167.1 used, 14003.4 buff/cache
MiB Swap: 16322.0 total, 15975.2 free, 346.8 used, 14308.6 avail Mem

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
27939 www-data  20   0 358204 101744 83320 S   10.0   0.6 543:25.53 zmc
27942 www-data  20   0 323372 143440 80352 S    3.3   0.9 216:13.65 zma
30262 www-data  20   0 242716 60232 4204  S    2.7   0.4  94:07.01 zma
10436 www-data  20   0 298628 108220 98352 S    2.3   0.7   0:25.16 zmc
30260 www-data  20   0 238180 53396 43772 S    2.0   0.3 89:21.10 zmc
10395 www-data  20   0 298704 107824 97944 S    1.7   0.7   0:20.86 zmc
10396 www-data  20   0 339628 158860 97680 S    1.3   1.0   0:14.36 zma
10418 www-data  20   0 339628 158872 97684 S    1.0   1.0   0:15.55 zma
  1 root      20   0 170896 8440 5628  S    0.0   0.1 2:50.43 systemd
  2 root      20   0      0      0      0  S    0.0   0.0 0:05.55 kthreadd
  3 root      0 -20      0      0      0  I    0.0   0.0 0:00.00 rcu_gp
  4 root      0 -20      0      0      0  I    0.0   0.0 0:00.00 rcu_per_gp
  6 root      0 -20      0      0      0  I    0.0   0.0 0:00.00 kworker/0+
  8 root      0 -20      0      0      0  I    0.0   0.0 0:00.00 mm_percpu+
  9 root      20   0      0      0      0  S    0.0   0.0 11:25.78 ksoftirq+
 10 root      20   0      0      0      0  I    0.0   0.0 477:05.75 rcu_sched
 11 root      20   0      0      0      0  I    0.0   0.0 0:00.00 rcu_bh

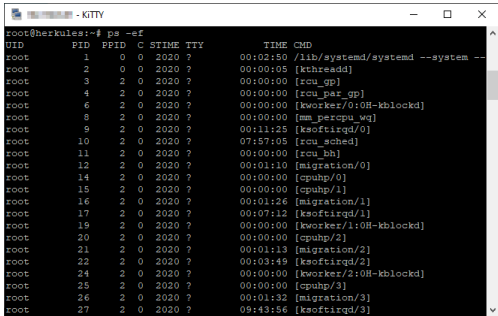
```

top

Unter Linux entspricht dem Task-Manager der Befehl `top`, der ständig aktuell Informationen zur Auslastung des Systems anzeigt sowie eine Liste aller Prozesse. Man kann `top` durch Drücken von `q` wieder verlassen.

Betriebssysteme 2023 Prof. Dr. Jörg Mielebacher 9

Prozesse unter Linux



```
root@herkules:~# ps -ef
UID          PID  PPID  C  STIME TTY          TIME CMD
root         1      0   0   2020 ?        00:02:50 /lib/systemd/systemd --system --
root         2      0   0   2020 ?        00:00:00 [kthreadd]
root         3      2   0   2020 ?        00:00:00 [rcu_gp]
root         4      2   0   2020 ?        00:00:00 [rcu_gpd]
root         6      2   0   2020 ?        00:00:00 [worker/0:0H-kblockd]
root         8      2   0   2020 ?        00:00:00 [mm_percpu_wq]
root         9      2   0   2020 ?        00:11:25 [ksoftirqd/0]
root        10      2   0   2020 ?        07:57:05 [rcu_sched]
root        11      2   0   2020 ?        00:00:00 [rcu_bh]
root        12      2   0   2020 ?        00:01:10 [migration/0]
root        14      2   0   2020 ?        00:00:00 [cpuhp/0]
root        15      2   0   2020 ?        00:00:00 [cpuhp/1]
root        16      2   0   2020 ?        00:01:26 [migration/1]
root        17      2   0   2020 ?        00:07:12 [ksoftirqd/1]
root        19      2   0   2020 ?        00:00:00 [worker/1:0H-kblockd]
root        20      2   0   2020 ?        00:00:00 [cpuhp/2]
root        21      2   0   2020 ?        00:01:13 [migration/2]
root        22      2   0   2020 ?        00:03:49 [ksoftirqd/2]
root        24      2   0   2020 ?        00:00:00 [worker/2:0H-kblockd]
root        25      2   0   2020 ?        00:00:00 [cpuhp/3]
root        26      2   0   2020 ?        00:01:32 [migration/3]
root        27      2   0   2020 ?        09:43:56 [ksoftirqd/3]
```

ps -ef

Ebenfalls möglich (und das Gegenstück zu tasklist unter Windows) ist der Befehl ps. So zeigt ps -ef alle Prozesse an. Das Ergebnis könnte man dann ggf. mit grep nach bestimmten Begriffen durchsuchen, z.B. ps -ef | grep java.



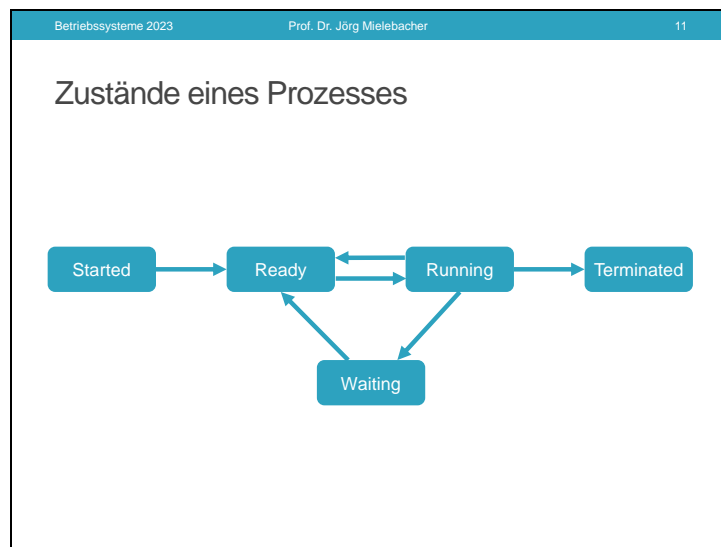
Wird ein Prozess gestartet – wird also das Programm in den Speicher geladen – umfasst der Speicherinhalt i.W. vier Teilsegmente:

Das Text-Segment enthält den Maschinencode des Programms.

Das Daten-Segment enthält die Daten, die bereits beim Start des Programms einen festgelegten Wert haben, also Konstanten sowie globale und statische Variablen. Lokale Variablen, die in einer Funktion angelegt werden, sind hier nicht abgelegt. Teilweise wird neben dem Data- auch noch das BSS-Segment (Block Started by Symbol) ausgewiesen, das alle globalen oder statischen, nicht initialisierten Variablen enthält.

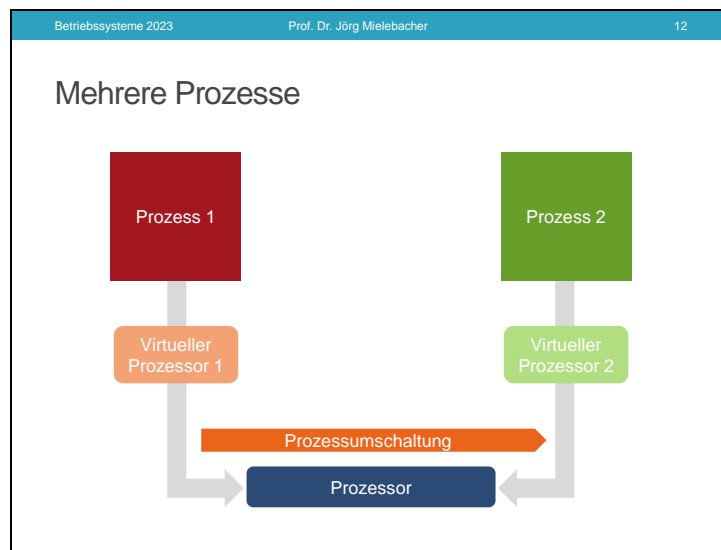
Das Heap-Segment hat im Gegensatz zu Text und Data keine feste Größe, sondern kann während des Laufzeit des Prozesses wachsen. Das Heap-Segment enthält den dynamisch zur Laufzeit reservierten Speicher. In C erfolgt dies z.B. mit malloc, in C++ mit new – sie nutzen die Systemaufrufe brk und sbrk.

Das Stack-Segment wiederum wächst in Richtung des Heap-Segments. Es enthält den sog. Call Stack, d.h. immer wenn eine Funktion aufgerufen wird, werden die Informationen der Funktion, in der der Aufruf erfolgt, auf dem Stack abgelegt, vor allem Parameter, Rücksprungadresse und die lokalen Variablen. Beim Verlassen der Funktion wird der Stack-Eintrag wieder entfernt.



Mit dem Erzeugen eines Prozesses befindet er sich zunächst im Zustand Started. Im Zustand Ready wartet er auf Zuteilung zum Prozessor, kann er ihn nutzen, wechselt er in den Zustand Running. Wartet der Prozess z.B. auf Benutzereingaben oder auf andere Ressourcen, befindet er sich im Zustand Waiting. Am Ende wechselt der Prozess in den Zustand Terminated, gleiches gilt, wenn er aktiv beendet wurde.

Das hier gezeigte Modell mit 5 Zuständen ist nicht das einzige Modell. Im einfachsten Fall wird ein 2-Zustands-Modell Idle – Running verwendet, aber auch feinere Unterteilungen sind anzutreffen.



Das Betriebssystem ordnet jedem Prozess einen virtuellen Prozessor zu. Diese virtuellen Prozessoren müssen nun nach einem bestimmten Verfahren von Zeit zu Zeit den oder die tatsächlichen Prozessoren nutzen können, um ihren zugeordneten Prozess auszuführen. Den Wechsel von Prozess zu Prozess bezeichnet man dabei als Prozessumschaltung. Die einzelnen Verfahren, in welcher Reihenfolge welcher Prozesse für wie lange Ressourcen zugeteilt bekommt, werden wir später näher betrachten.

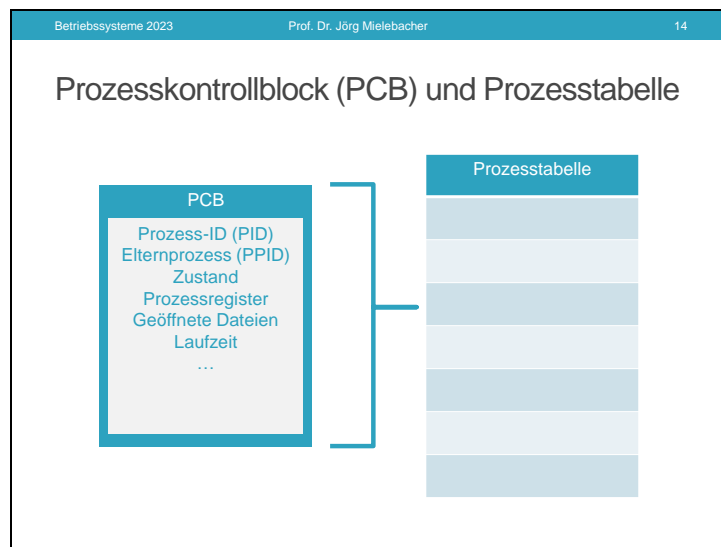


Eine Prozessumschaltung bedeutet natürlich auch, dass z.B. die Registerinhalte des echten Prozessors zwischengespeichert werden müssen. Das Betriebssystem verwaltet alle Informationen des Prozesses im sog. Prozesskontext. Der Prozesskontext ist i.W. dreigeteilt:

Der System-Kontext enthält die Daten, die das Betriebssysteme für die Verwaltung des Prozesses vorhält, z.B. die Prozess-ID, den Prozesszustand, seine Priorität, den Elternprozess (d.h. der Prozess, von dem der Prozess erzeugt wurde), etwaige Kindprozesse, ggf. geöffnete Dateien oder auch die Laufzeit.

Der Hardware-Kontext enthält alle notwendigen Prozessorregister, vor allem den Befehlszähler, der auf die aktuelle Anweisung verweist. Außerdem Verweise auf die Seitentabelle, mehr dazu im Kapitel Speicher.

Der Benutzer-Kontext wiederum enthält die Daten des Prozesses im virtuellen Adressraum des Prozesses.



Das Betriebssystem fasst System- und Hardware-Kontext in einer speziellen Datenstruktur zusammen, dem sog. Prozesskontrollblock (Process Control Block, PCB). Die Prozesstabelle enthält die PCBs aller Prozesse.

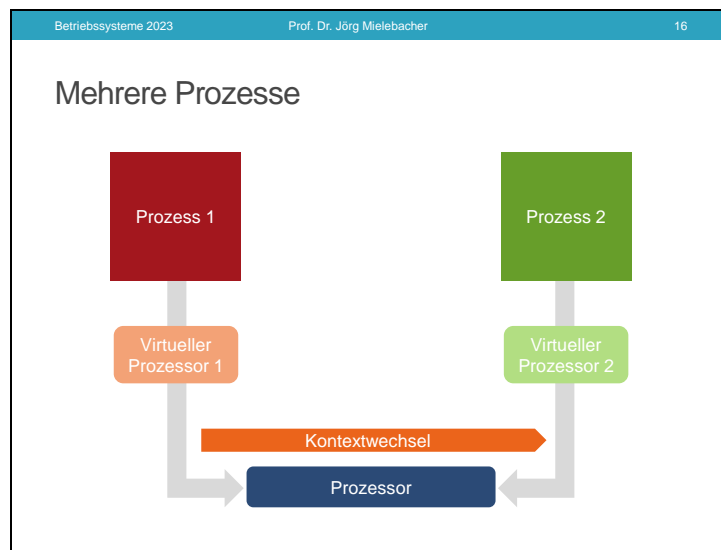
Betriebssysteme 2023 Prof. Dr. Jörg Mielebacher 15

Beispiel Linux-ps

```

root@serverkali:~# ps -fu www-data
  UID          PID   PPID    C   SZ   RSS   FSR  STIME  TTY          TIME CMD
www-data    2775 20426   0 49967 12020   3 00:00 ?        00:00:00 /usr/sbin/apache2 -k start
www-data    2776 20426   0 49967 12022   4 00:00 ?        00:00:00 /usr/sbin/apache2 -k start
www-data    3171 19588   0 71464 49316   5 01:55 ?        00:02:34 apache2 -DFOREGROUND
www-data    4532 15588   0 78488 62304   3 04:17 ?        00:02:04 apache2 -DFOREGROUND
www-data    5439 27759   0 14297 39916   4 06:18 ?        00:00:02 /usr/bin/perl -wT /usr/bin/rmfilter.pl --filter_id=2 --daemon
www-data    5440 27759   0 14176 39676   3 06:18 ?        00:00:03 /usr/bin/perl -wT /usr/bin/rmfilter.pl --filter_id=1 --daemon
www-data    5445 27759   0 14245 47824   0 06:18 ?        00:00:03 /usr/bin/perl -wT /usr/bin/rmfilter.pl --filter_id=3 --daemon
www-data    7529 19588   0 71476 48284   3 11:40 ?        00:01:02 apache2 -DFOREGROUND
www-data    7775 19588   0 71469 48216   2 12:25 ?        00:00:59 apache2 -DFOREGROUND
www-data    8120 19588   0 71483 48889   4 13:13 ?        00:00:48 apache2 -DFOREGROUND
www-data    9342 19588   0 71472 50264   3 13:43 ?        00:00:43 apache2 -DFOREGROUND
www-data    9209 19588   0 71410 49149   1 14:08 ?        00:00:26 apache2 -DFOREGROUND
www-data   11411 19588   0 71420 49152   2 21:08 ?        00:00:08 apache2 -DFOREGROUND
www-data   11412 19588   0 71409 49216   0 21:09 ?        00:00:04 apache2 -DFOREGROUND
www-data   11808 27759   2 75410 110764   3 21:21 ?        00:01:37 /usr/bin/rm -R 4
www-data   11811 27759   1 83107 111642   1 21:21 ?        00:00:37 /usr/bin/rm -R 4
www-data   12012 19588   0 87798 114592   4 22:04 ?        00:00:05 apache2 -DFOREGROUND
  
```

Der ps-Befehl erlaubt unter Linux tiefe Einblicke in die Prozesstabelle. Im erweiterten Format `-f` werden angezeigt: Eigentümer (Username), Prozess-ID (PID), Elternprozess (PPID), prozentuale CPU-Nutzung über die Lebenszeit, Nutzung des virtuellen Speichers, Nutzung des physikalischen Speichers (d.h. nicht ausgelagert), zugeordneter Prozessor, Startzeitpunkt, verwendetes Terminal, gesamte Nutzungszeit des Prozessors, Kommando mit dem der Prozess gestartet wurde inkl. Parametern.



Anhand der Informationen in der Prozesstabelle wird nun beim Wechsel von Prozessen der Zustand des nächsten Prozesses mit den Inhalten seines PCB wiederhergestellt, während der Zustand des bisherigen Prozesses in seinem PCB gesichert wird. Kontextwechsel erfordern daher auch Zeit.

Betriebssysteme 2023	Prof. Dr. Jörg Mielebacher	17
<p data-bbox="469 584 633 636">Scheduling</p>		

Betriebssysteme 2023

Prof. Dr. Jörg Mielebacher

18

Definitionen

Prozess-Scheduling

Non-preemptive Scheduling

Preemptive Scheduling

Prozess-Scheduling beschreibt die dynamische Zuteilung der auszuführenden Prozessen zu dem Prozessor (bzw. den Prozessoren). Hierfür gibt es verschiedene Ansätze, diese kann man zunächst unterteilen in non-preemptive Scheduling, bei dem ein Prozess nicht unterbrochen werden darf; der Prozess gibt den Prozessor lediglich „freiwillig“ ab – daher auch der Name kooperatives Scheduling. Beim preemptive Scheduling kann das Betriebssystem jederzeit einen Prozess unterbrechen, um einem anderen Prozess Zugriff auf den Prozessor zu geben.

Betriebssysteme 2023 Prof. Dr. Jörg Mielebacher 19

Ziele

- Möglichst **gute Auslastung der CPU**
- Möglichst **hoher Durchsatz**
- Möglichst **kurze Laufzeit** der Prozesse
- Möglichst **kurze Wartezeit**
- **Einhaltung von Vorgaben**

- ➔ **Scheduling erfordert selbst Ressourcen**
- ➔ **Prozessumschaltung erfordert Ressourcen**

Mit dem Scheduling verfolgt man eine Reihe von Zielen:

- 1.) Man strebt an, dass die CPU möglichst gut ausgelastet ist, d.h. solange Prozesse die CPU benötigen, sollte die CPU auch möglichst voll genutzt werden.
- 2.) Man strebt an, dass möglichst viele Prozesse gleichzeitig ausgeführt werden können, also einen hohen Durchsatz.
- 3.) Die Zeit, die ein Prozess zur Ausführung benötigt, soll möglichst kurz sein. Sobald mehrere andere Prozesse bedient werden müssen, kann nämlich i.d.R. nicht die normale Laufzeit erreicht werden, da die Ausführung unterbrochen wird.
- 4.) Die Zeit, die ein Prozess darauf wartet, die CPU zu erhalten, soll insgesamt möglichst kurz sein.

Im Echtzeitumfeld spielt außerdem die Einhaltung von Vorgaben (sog. Deadlines) eine zusätzliche Rolle. Um die Echtzeitfähigkeit wahren zu können, dürfen Prozesse nicht zu lange auf die CPU warten müssen.

Allerdings muss man bei der Umsetzung des Scheduling bedenken, dass ein Scheduler selbst wiederum Ressourcen erfordert und die CPU nutzen muss (meist umso mehr, je komplexer das Verfahren). Außerdem können sich häufige Prozessumschaltungen negativ auswirken, da auch diese Ressourcen erfordern.

Betriebssysteme 2023Prof. Dr. Jörg Mielebacher20

Scheduler vs. Dispatcher

- **Scheduler**
 - Auswahl des nächsten Prozesses, der die CPU nutzen darf
 - Kriterien wie gesehen
- **Dispatcher**
 - Durchführen des Kontextwechsels
 - Stoppen des bisherigen Prozesses
 - Starten des nächsten Prozesses
 - Kriterium: **Dispatch Latency**

Das Betriebssystem umfasst zwei Komponenten – den Scheduler, der auswählt, welcher Prozess als nächstes die CPU nutzen darf und der Dispatcher, der den Kontextwechsel zum nächsten Prozess durchführt, also den bisherigen Prozess anhält und den nächsten startet. Als Dispatch Latency bezeichnet man dabei die Zeit, die für die Umschaltung benötigt wird, also vom Stoppen des einen Prozesses bis zum Start des nächsten Prozesses. Man strebt eine möglichst geringe Dispatch Latency an.



Das Scheduling wird insofern erschwert, als i.d.R. alle Prozesse unterschiedlich ablaufen. Z.B. könnte ein Prozess längere Phasen des Rechnens (also der CPU-Nutzung) haben, gefolgt von kurzen Ein-/Ausgabephasen. Die roten Phasen bezeichnet man dabei als CPU-Burst, die grünen als IO-Burst.

Ein anderer Prozess könnte demgegenüber längere IO-Phasen haben, die von kürzeren Phasen des Rechnens unterbrochen sind. Während der erste Prozess somit als rechenintensiv bezeichnet werden kann, würde man den zweiten Prozess IO-intensiv nennen.

Offenkundig hat also die Charakteristik des Prozesses Einfluss darauf, ob sich ein Scheduling-Verfahren eignet oder nicht.

Mit dem Scheduling-Simulator der TU Dortmund können Sie ein wenig mit den Scheduling-Verfahren, die wir uns gleich anschauen, beschäftigen:

<https://ess.cs.tu-dortmund.de/Software/AnimOS/CPU-Scheduling/>



Das einfachste Schedulingverfahren ist First-Come First-Served (FCFS), bei dem die Prozesse in der Reihenfolge ihres Eintreffens abgearbeitet werden. Es wird üblicherweise durch eine FIFO-Warteschlange realisiert und erfordert daher wenig Ressourcen für das Scheduling. Es ist non-preemptive und ist das typische Scheduling-Verfahren für die Stapelverarbeitung. Da jeder Prozess erst vollständig abgearbeitet werden muss, ist es für den Dialogbetrieb nicht geeignet und führt zu vergleichsweise langen Wartezeiten. Auch ist die CPU-Auslastung i.d.R. eher gering.

Betriebssysteme 2023	Prof. Dr. Jörg Mielebacher	23
<h3>Shortest Job First (SJF)</h3> <ul style="list-style-type: none">▪ Prozess mit der kürzesten Dauer wird zuerst ausgeführt▪ Non-preemptive▪ Geeignet für Stapelverarbeitung▪ Meist Reduktion der Wartezeit gegenüber FCFS▪ Aber: Erwartete Laufzeit wird benötigt		

Shortest Job First (SJF) wählt als nächsten Prozess denjenigen mit der kürzesten Laufzeit. Es ist non-preemptive und geeignet für die Stapelverarbeitung. Gegenüber FCFS beobachtet man meist eine Reduktion der mittleren Wartezeit. SJF benötigt zumindest Schätzungen der erwarteten Laufzeit.

Betriebssysteme 2023 Prof. Dr. Jörg Mielebacher 24

Shortest Remaining Time (SRT)

- Prozess mit der kürzesten Restdauer wird ausgeführt
- Preemptive
- Geeignet für Stapelverarbeitung
- Ungeeignet für interaktive System, da Restdauer unbekannt

Shortest Remaining Time (SRT) ist eine preemptive Variante von SJF. Hier wird als nächstes derjenige Prozess mit der kürzesten Restdauer ausgewählt. SRT eignet sich daher für die Stapelverarbeitung, aber nicht für die interaktive Systeme, weil i.d.R. die Restdauer unbekannt ist.

Betriebssysteme 2023	Prof. Dr. Jörg Mielebacher	25
<h2>Prioritätsbasiertes Scheduling</h2> <ul style="list-style-type: none">▪ Prozess mit der höchsten Priorität wird als nächstes ausgeführt.▪ Bei gleicher Priorität: FCFS▪ Non-preemptive▪ Verbreitet für Stapelverarbeitung		

Das Prioritätsbasierte Scheduling ist eine Erweiterung von FCFS, wobei jedem Prozess eine Priorität zugewiesen wird und stets als nächstes der Prozess mit der höchsten Priorität ausgewählt wird. Bei gleicher Priorität der wartenden Prozesse wird FCFS verwendet. Umsetzen lässt sich das Scheduling mit sog. Prioritätswarteschlangen, also ebenfalls sehr ressourcenarm. Es ist ebenfalls non-preemptive und in der Stapelverarbeitung verbreitet.

Betriebssysteme 2023

Prof. Dr. Jörg Mielebacher

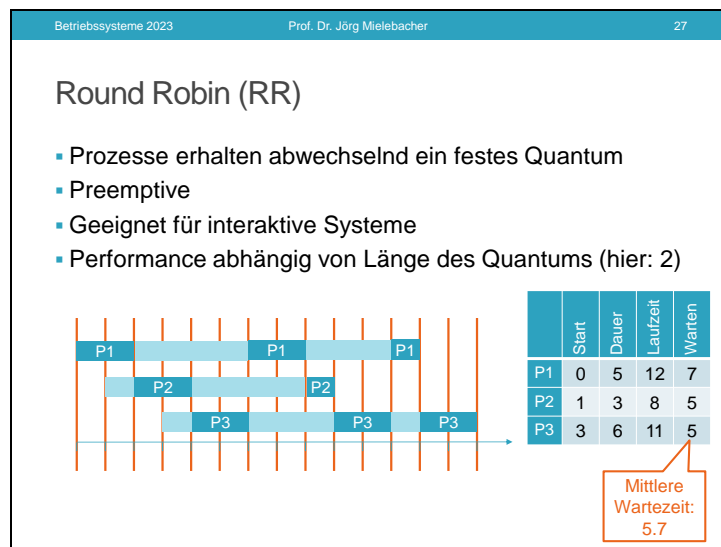
26

Aufgabe

- Welche Probleme erwarten Sie beim prioritätsbasierten Scheduling?

Starvation: Prozesse mit niedriger Priorität müssen so lange warten, bis alle Prozesse höherer Priorität abgearbeitet sind. Eine Lösung kann darin bestehen, die Priorität mit im Lauf des Wartens schrittweise zu erhöhen.

Prioritätsumkehr: Prozesse mit niedriger Priorität blockieren u.U. Prozesse höherer Priorität, wenn sie gemeinsame Ressourcen nutzen. Dies lässt sich lösen, indem man dem Prozess, der die Ressourcen aktuell verwendet, für diese Zeit eine höhere Priorität zuweist, sodass die Ressourcen schnell wieder freigegeben werden.



Bei Round Robin erhalten die Prozesse ein festes Quantum. Die Ausführung wird also immer wieder kurz unterbrochen und der Prozess muss gewechselt werden. Dadurch gehört Round Robin (RR) zu den preemptive Verfahren. Die wiederholte Zuteilung des Prozessors verhindert unplanbare, lange Wartezeiten und ist daher auch für interaktive Systeme geeignet. Die Performance ist allerdings abhängig von der Länge des Quantums.

Im gezeigten Beispiel beträgt das Quantum 2. Die Prozesse treffen nacheinander ein, werden also nacheinander in die Warteschlange aufgenommen, um anschließend wiederkehrend den Prozessor zu erhalten, bis der Prozess beendet wird.

Betriebssysteme 2023

Prof. Dr. Jörg Mielebacher

28

Aufgabe

- Welche Probleme erwarten Sie bei einem zu kleinen oder zu großen Quantum?

Zu kleines Quantum: Häufige Kontextwechsel werden nötig, die zeitintensiv sind und dadurch unnötig Ressourcen binden.

Zu großes Quantum: Die Antwortzeit des Prozesses kann für die interaktive Nutzung zu lang werden.

Auf der Grundlage der zu erwartenden Prozesse muss also das optimale Quantum geschätzt werden.

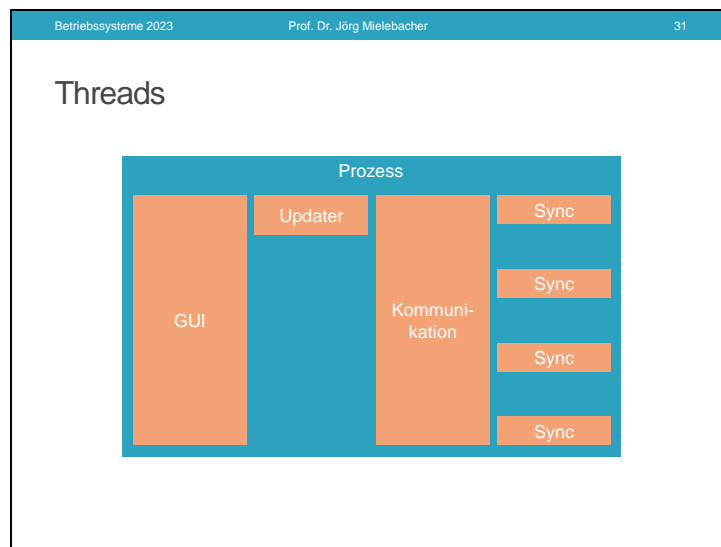
Betriebssysteme 2023 Prof. Dr. Jörg Mielebacher 29

Multilevel Queue Scheduling

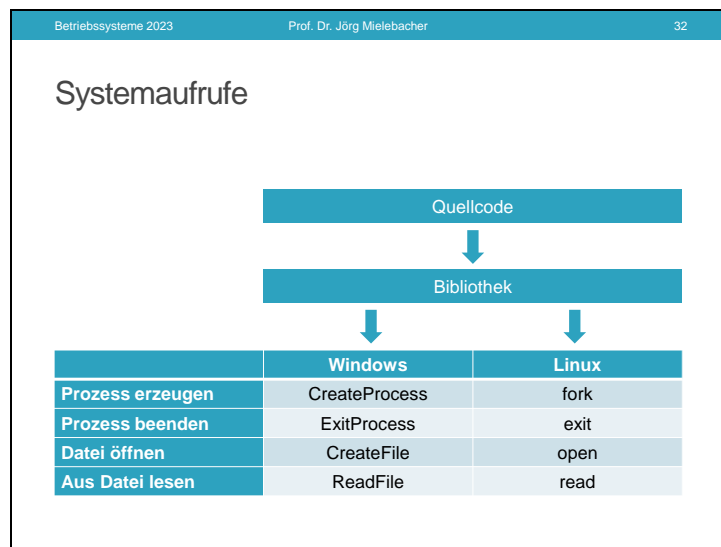
- Unterschiedliche Scheduling-Verfahren je nach Gruppe der Prozesse
- Z.B. RR für interaktive Prozesse und FCFS für Stapelprozesse

Das Multilevel Queue Scheduling ist kein eigenständiges Scheduling-Verfahren, sondern kombiniert unterschiedliche Verfahren, die es für Gruppen von Prozessen anwendet. Das löst insbesondere den Umgang mit den sehr verschiedenen Anforderungen von Stapelverarbeitungs- und interaktiven Prozessen.

Betriebssysteme 2023	Prof. Dr. Jörg Mielebacher	30
<p data-bbox="467 586 924 631">Aus dem Leben eines Prozesses</p>		



Threads sind Anweisungsstränge innerhalb eines Prozesses, die nebenläufig ausgeführt werden können. In dem hier gezeigten Prozess gibt es beispielsweise eine GUI-Thread, der für die Aktualisierung der Benutzeroberfläche zuständig ist. Außerdem wird beim Start ein Updater-Thread gestartet, der auf Aktualisierungen der Software prüft. Parallel gibt es über die gesamte Lebenszeit einen Kommunikations-Thread, der Anfragen per Netzwerk entgegennimmt. Zusätzlich könnte es einen wiederholt gestarteten Thread geben, der Daten im Hintergrund synchronisiert. Der Vorteil von Threads besteht darin, dass Prozesswechsel überflüssig sind, da die Threads Teil des Prozesses sind. Dennoch können sie parallel auf unterschiedlichen Prozessoren oder Prozessorkernen ausgeführt werden. Ohne solche nebenläufigen Threads müsste der Prozess immer auf einem Prozessor ausgeführt werden und könnte die heute üblichen Mehrkernprozessoren nicht sinnvoll nutzen. Im Gegensatz zu Prozessen sind Threads jedoch nicht voneinander isoliert. Sie haben Zugriff auf alle Ressourcen des Prozesses, was die Koordination des Zugriffs darauf erfordert.



Prozesse im User Mode können nicht auf Kernel-Code und –Daten zugreifen. Um ihnen den Zugriff auf Betriebssystemfunktionalität zu erlauben, stellt das Betriebssystem eine Schnittstelle für Systemaufrufe zur Verfügung. Welche dies sind, legt das jeweilige Betriebssystem fest, wobei es wiederkehrende Funktionalität gibt, die man auf jedem Betriebssystem findet.

In der jeweiligen Programmiersprache z.B. stehen wiederum eigene Befehle zur Verfügung, z.B. `fopen` in C oder `ifstream` in C++, um eine Datei zu öffnen; diese wiederum nutzen – je nach Plattform – die dort zur Verfügung stehenden Systemaufrufe.

Betriebssysteme 2023 Prof. Dr. Jörg Mielebacher 33

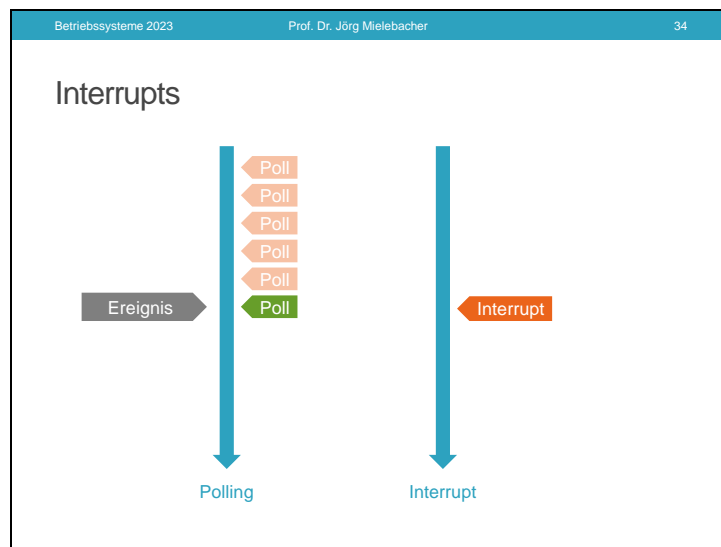
Ausführen eines Systemaufrufs

- Prozesskontext sichern (inkl. Befehlszähler)
- Befehlszähler wird auf Systemroutine gesetzt
- Umschalten von User Mode in Kernel Mode
- Ausführen der Systemroutine
- Wiederherstellen des Prozesskontexts
- Umschalten von Kernel Mode in User Mode
- Ausführung des Prozesses nach Systemaufruf fortsetzen

➔ Trap / Supervisor-Call (SVC)

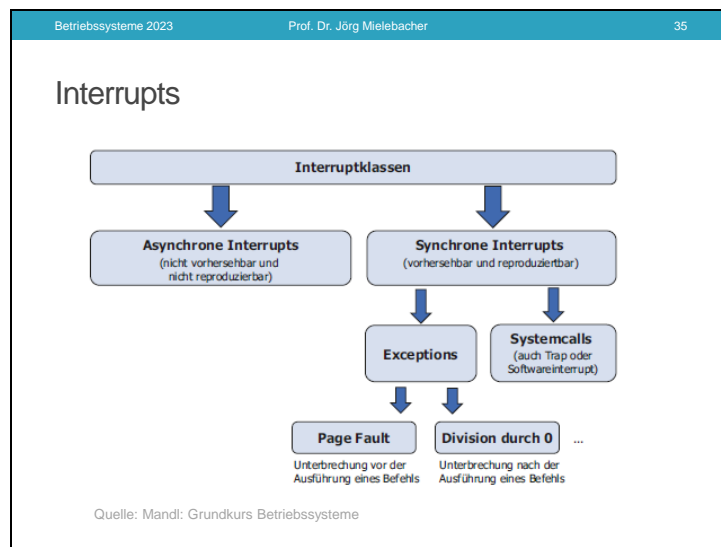
Wenn ein Systemaufruf erfolgen soll, muss zunächst des Prozesskontext gesichert werden, insbesondere auch der Befehlszähler, da man später mit der nächsten Anweisung fortfahren möchte. Da nun die Systemroutine ausgeführt werden soll, muss der Befehlszähler des Prozessors jetzt auf die Systemroutine gesetzt werden. Außerdem schaltet man, da nun Kernel-Code ausgeführt wird, vom User Mode in den Kernel Mode um. Die Systemroutine kann nun ablaufen. Ist sie abgeschlossen, kehrt man zum ursprünglichen Prozess zurück, dessen Kontext wiederhergestellt wird. Außerdem wird vom Kernel Mode zurück in den User Mode gewechselt. Der Prozess kann nun die nächste Anweisung nach dem Systemaufruf ausführen.

Dennoch hat dieses scheinbar so einleuchtende Vorgehen einen Haken: Möchte man ein Unterprogramm aufrufen, benötigt man dessen Adresse, um den Befehlszähler darauf zu setzen. Prozesse im User Mode sollen jedoch nicht die Adressen von Systemroutinen kennen. Daher nutzen Systemaufrufe sog. Traps, die mit einem Maschinenbefehl, dem Supervisor-Call (SVC), dafür sorgen, dass vom User Mode in den Kernel Mode gewechselt wird, um dort die Systemroutine auszuführen.



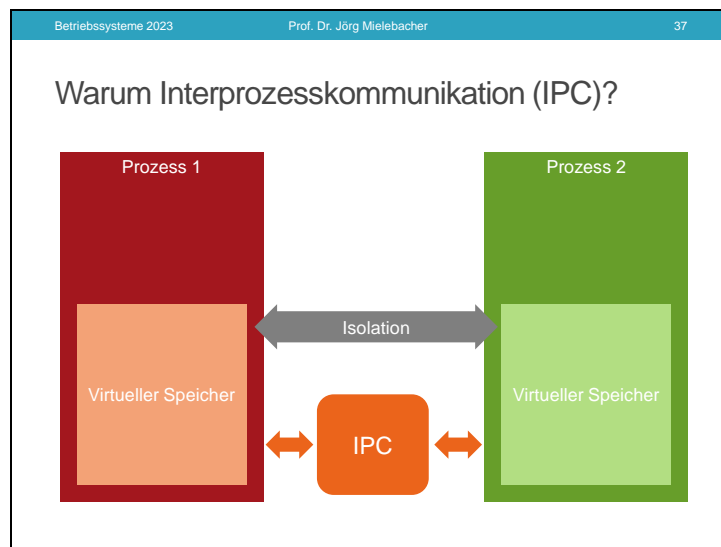
Häufig stellt sich das Problem, dass ein bestimmtes Ereignis erkannt werden muss, um darauf mit bestimmten Routinen zu reagieren. Hierfür gibt es zwei Ansätze:

- 1.) Beim sog. Polling wird wiederholt geprüft, ob das bestimmte Ereignis auftritt. Das ist ggf. sehr ressourcenintensiv.
- 2.) Durch Interrupts ist es möglich, bei Auftreten bestimmter Ereignisse die Ausführung des aktuellen Prozesses zu unterbrechen und eine dem Interrupt zugeordnete Routine auszuführen, die sog. Interrupt-Service-Routine (ISR).



Es gibt unterschiedliche Arten von Interrupts: Asynchrone Interrupts sind nicht Teil eines Prozesses; sie entstehen z.B. wenn eine Nachricht über das Netzwerk eintrifft oder wenn Block von einem Datenträger gelesen wurde. Zu den Synchronen Interrupts zählen Ausnahmen (Exceptions), die der Prozessor auslöst, wenn ein Befehl dies notwendig macht (z.B. Division durch 0). Eine besondere Art synchroner Interrupts sind die bereits kennengelernten Traps, die bei Systemaufrufen erfolgen.

Betriebssysteme 2023	Prof. Dr. Jörg Mielebacher	36
<p data-bbox="469 586 852 636">Interprozesskommunikation</p>		



Prozesse nutzen ihren jeweils eigenen virtuellen Adressraum, d.h. zwei Prozesse verfügen über keinen gemeinsamen Speicher – sie sind (aus gutem Grund) voneinander isoliert. Müssen Prozesse jedoch Informationen miteinander austauschen, wird diese Isolation zu einem Problem. Betriebssysteme bieten daher Unterstützung für die sog. Interprozesskommunikation an (Inter-process communication, IPC).

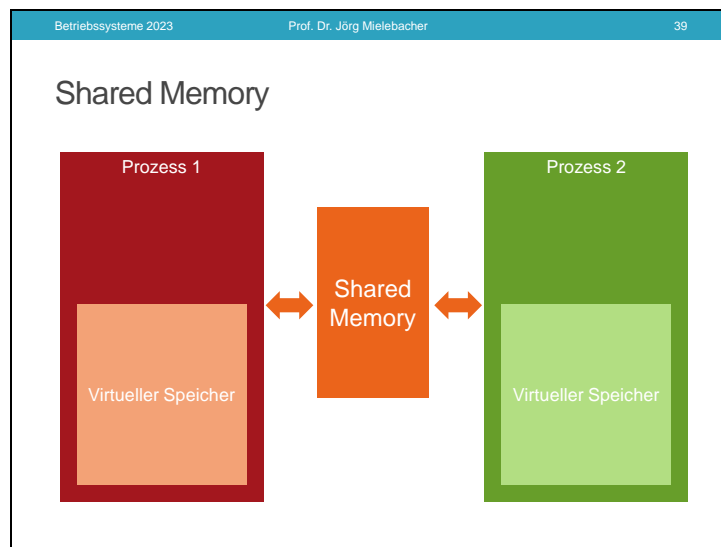
Das ist umso wichtiger, wenn man bedenkt, dass viele Anwendungen nicht nur mehrere Threads, sondern auch mehrere Prozesse verwenden. Beim Absturz eines Threads, ist der ganze Prozess betroffen. Wenn jedoch einzelne Prozesse ausfallen, können die übrigen weiterhin ausgeführt werden und der fehlende Prozesse neugestartet werden. Dies sieht man heutzutage z.B. bei vielen Browsern, die jeden Browser-Tab als einzelnen Prozess ausführen.

Betriebssysteme 2023 Prof. Dr. Jörg Mielebacher 38

Typische Anwendungen der IPC:

- Gemeinsam verwendete Daten
- Weiterreichen von Daten
- Synchronisation von Prozessen
- Zugriffskoordination

Es gibt eine Reihe von Anwendungen für IPC, z.B. wenn zwei oder mehr Prozesse Daten gemeinsam verwenden oder wenn Daten von einem Prozess an einen anderen weitergereicht werden sollen, z.B. `history | grep test`. Auch kann es notwendig sein, dass sich unabhängig ablaufende Prozesse synchronisieren, z.B. wenn der eine Prozess auf den anderen warten muss. Eine weitere Anwendung betrifft die Zugriffskoordination bei gemeinsam genutzten Ressourcen, z.B. wenn mehrere Prozesse auf einem Drucker Ausdrücke erzeugen wollen.

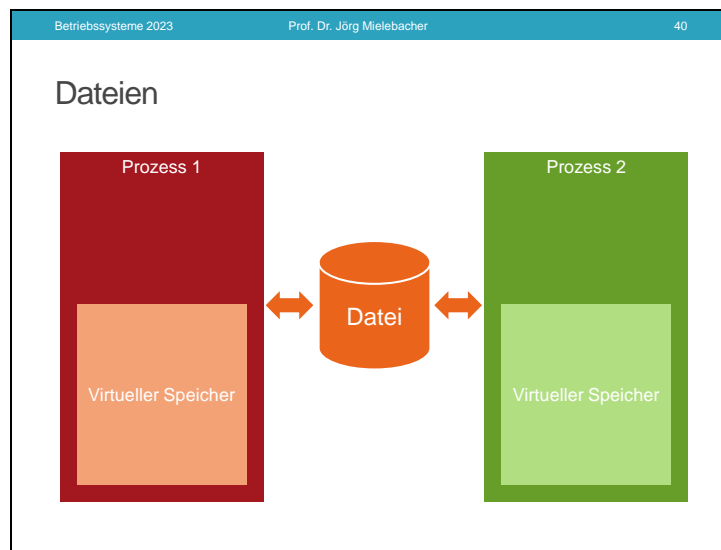


Da die Speicherbereiche von Prozessen voneinander isoliert sind, braucht es eine explizite Unterstützung gemeinsamer Speicherbereiche. Diese Art der IPC ist schnell, da es sich um direkte Speicherzugriffe handelt, allerdings fehlt hier eine angemessene Form der Abstraktion und Kapselung.

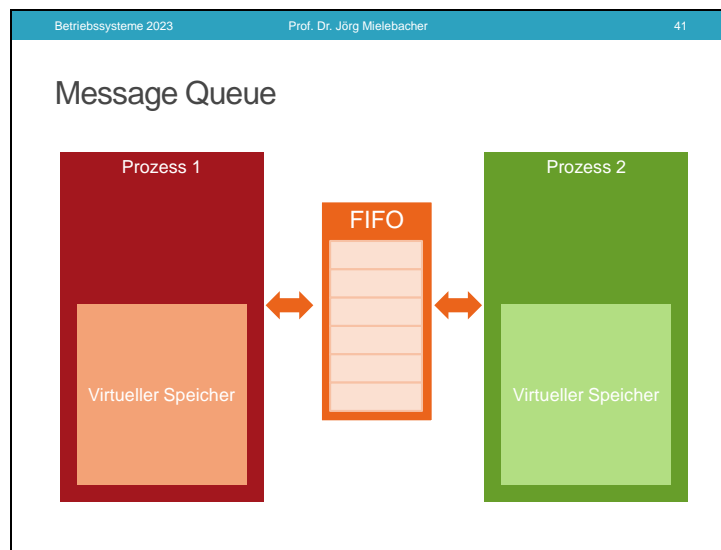
Unter Linux kann man ein Shared-Memory-Segment mit `shmget()` anlegen oder öffnen, mit `shmctl()` bearbeiten und mit `shmat()` an einen Prozess binden. Anschließend kann man unter der betreffenden Adresse auf den Speicherbereich zugreifen. Allerdings braucht man für die Zugriffskoordination zwischen den Prozessen Hilfsmittel wie Semaphoren. `Shmget` ist jedoch eher überholt.

Daneben Linux bietet mit `/dev/shm` ein im Hauptspeicher angelegtes Verzeichnis, das von allen Prozessen zugegriffen werden kann. Man sieht hier, dass eine strenge Trennung zwischen gemeinsamem (Haupt)Speicher und der IPC mit Dateien nicht immer möglich ist.

Es gibt außerdem plattformübergreifende Techniken, z.B. bietet Qt mit `QSharedMemory` eine Shared-Memory-Klasse, die auf mehreren Betriebssystemen eingesetzt werden kann und dabei die jeweiligen Plattformbibliotheken verbirgt.



Bei der IPC durch Dateien werden Dateien für den Informationsaustausch genutzt. Wie gerade gesehen nutzen auch Shared-Memory-Ansätze oftmals Dateien als Grundlage, wobei hier die Nutzung über den Hauptspeicher im Vordergrund steht. Bei der Nutzung von Dateien für die IPC können Prozesse in die Datei schreiben, während andere aus der Datei lesen. Da das Betriebssystem ein exklusives Öffnen von Dateien unterstützt, ist hierdurch eine Zugriffskoordination möglich. Datei-IO ist vergleichsweise langsam, was ein Nachteil dieser Art der IPC ist.



Bei einer Message Queue wird zwischen Prozessen eine FIFO-Warteschlange angelegt, in die der eine Prozesse Nachrichten (typischerweise Text) schreiben kann, während der andere Prozess diese Nachricht entnimmt. Üblicherweise gibt es mehrere Warteschlangen, die unter einem Namen (oder einer anderen Kennung) angesprochen werden können. Auch Prioritätswarteschlangen sind hierfür möglich.

Unter Linux können Message Queues mit `msgget()` angelegt oder geöffnet werden, mit `msgsnd()` lassen sich Nachrichten in die Queue einfügen, während `msgrcv()` die nächste Nachricht entnimmt.

Betriebssysteme 2023Prof. Dr. Jörg Mielebacher42

Pipes

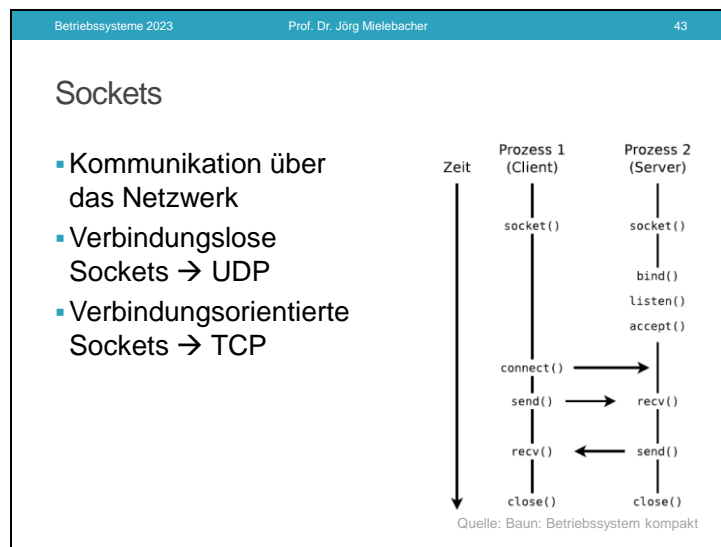
- Unidirektionale Kommunikation
- FIFO
- Beispiel:
history | grep test
- (Anonyme) Pipe
- Benannte Pipe

Pipes erlauben eine unidirektionale Kommunikation zwischen Prozessen, üblicherweise nach dem Prinzip FIFO. Man kennt Pipes vor allem von der Unix-Kommandozeile: Ein Befehl wie `history | grep test` sorgt dafür, dass die Ausgabe von `history` an `grep` weitergereicht wird, das diese Ausgabe als Eingabe verwendet.

Es gibt „normale“ (unbenannte) Pipes, die mit dem Ende des Prozesses, der sie erzeugt hat, auch wieder zerstört werden. Wichtig ist außerdem, dass Pipes nur zwischen Prozessen möglich sind, die miteinander verwandt sind, also z.B. denselben Elternprozess haben.

Andererseits gibt es mit named Pipes eine Erweiterung, bei der jede Pipe einen Namen besitzt und über den Prozess hinaus fortbesteht. Diese können auch zwischen nicht verwandten Prozessen verwendet werden.

In Linux gibt es für das Erzeugen von Pipes den Systemaufruf `pipe()`, die Windows API wiederum nutzt `CreatePipe`.



Während Pipes letztlich nur zwischen Prozessen desselben Rechners genutzt werden können, erlauben Sockets auch eine Kommunikation über das Netzwerk. Man unterscheidet dabei verbindungslose Sockets über UDP, die sehr effizient sind, aber u.U. weniger zuverlässig. Verbindungsorientierte Sockets nutzen TCP, was zuverlässiger ist, aber mehr Verwaltung erfordert. Die Abbildung zeigt die notwendigen Linux-Systemaufrufe im Zusammenhang mit der Nutzung einer verbindungsorientierten Socket. Diese wird zunächst von Prozess 2 mit `socket()` angelegt, mit `bind()` an einen Port gebunden und mit `listen()` und `accept()` so aktiviert, dass auf eingehende Anfragen von Clients reagiert wird. Anschließend ist nach Verbindung mit `connect()` ein Austausch über `send()` und `recv()` möglich. Am Ende wird die Socket mit `close()` geschlossen.

Betriebssysteme 2023 Prof. Dr. Jörg Mielebacher 44

Zwischenfazit

- Sehr grundlegende, kleinteilige IPC-Unterstützung
- Kommunikation auf demselben Rechner vs. Kommunikation über das Netzwerk
- Wiederkehrende Probleme erfordern stets ähnliche Lösungsansätze

→ Notwendigkeit für Abstraktion

Die gezeigten Ansätze für IPC sind zweifelsohne hilfreich, allerdings auch eher kleinteilig. Sie unterstützen einerseits die Kommunikation auf demselben Rechner (z.B. Shared Memory oder Pipes), andererseits aber auch die Kommunikation über das Netzwerk (z.B. Sockets). Allerdings zeigt sich, dass bei diesen sehr traditionellen Ansätzen immer wieder dieselben Probleme auftauchen (z.B. Zugriffscoordination), die für jede Anwendung erneut gelöst werden müssen. Daher sind Ansätze notwendig, die ein höheres Abstraktionsniveau bieten. Beispiele für dieses Vorgehen folgen auf den nächsten Folien.

Betriebssysteme 2023 Prof. Dr. Jörg Mielebacher 45

Beispiel: D-Bus

- IPC für Desktop-Umgebungen
- Ursprünglich für freedesktop.org entwickelt
- Ersetzt Punkt-zu-Punkt-Kommunikation durch Nachrichtenbus
- Kommunikationsmodi:
 - Request-response
 - Publish/Subscribe

D-Bus wurde seit 2002 – ursprünglich für freedesktop.org – für die IPC speziell in Desktop-Umgebungen entwickelt. Ziel war es, die sonst übliche Punkt-zu-Punkt-Kommunikation zu ersetzen und in Form eines Nachrichtenbusses zu abstrahieren.

Mit D-Bus ist einerseits die Request-Response-Kommunikation zwischen zwei Partnern möglich, andererseits Publish/Subscribe-Kommunikation, bei der eine Nachricht an alle Teilnehmer eines Kanals weitergegeben wird. Hierbei ist eine Antwort jedoch nicht möglich.

Betriebssysteme 2023

Prof. Dr. Jörg Mielebacher

46

Beispiel: ZeroMQ

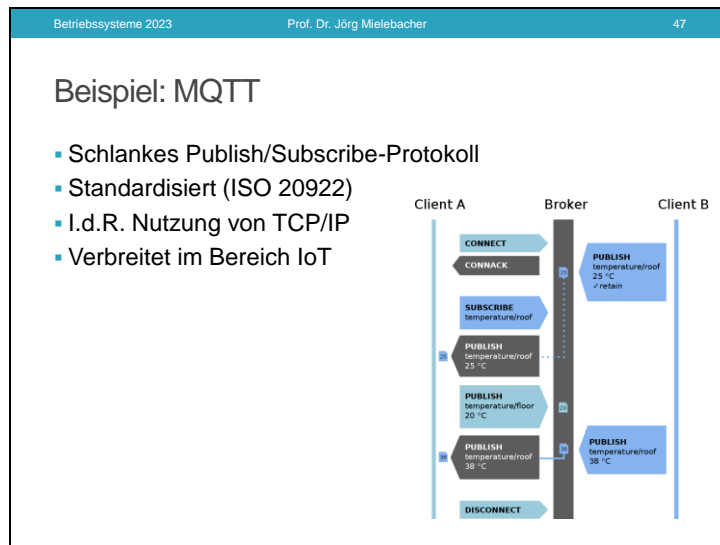
- Message Queue
- Verwendbar ohne eigenen Message Broker
- Kommunikationsmuster:
 - Request – Reply
 - Publish/Subscribe
 - Push-Pull
 - Exclusive Pair

ZeroMQ stellt eine Message Queue für verteilte Systeme zur Verfügung. Die Queue ist ohne eigenen Message Broker verwendbar.

ZeroMQ bietet auf der Basis von Sockets vier verschiedene Kommunikationsmuster:

- Request – Reply, um Clients und Services miteinander zu verbinden
- Publish/Subscribe, mit dem Teilnehmer (Subscriber) die Nachrichten erhalten, die in einem Kanal von einem Publisher veröffentlicht werden
- Push – Pull, mit dem Aufgaben an mehrere Knoten verteilt und verarbeitet werden können
- Exclusive Pair, mit dem zwei Sockets auf unterer Ebene miteinander verbunden werden

ZeroMQ steht unter einer Open-Source-Lizenz zur Verfügung.



MQTT ist ein Publish/Subscribe-Protokoll, das nach ISO 20922 standardisiert ist. Es nutzt einen Broker, über den Nachrichten in Kanälen veröffentlicht(Publish) werden. Diese Kanäle lassen sich abonnieren (Subscribe), um die veröffentlichten Nachrichten zu erhalten. MQTT ist sehr schlank und ist daher auch im Embedded-Umfeld sehr beliebt. Es wird z.B. im Kontext Internet-of-Things für intelligente Sensoren oder Edge Computing eingesetzt.

Zusammenfassung

- Prozesse sind Programme in Ausführung. Das Betriebssystem verwaltet sie anhand der PCBs in der Prozesstabelle.
- Prozesse unterscheiden sich in Häufigkeit und Dauer von CPU- und IO-Bursts.
- Die Umschaltung zwischen Prozessen erfordert den Wechsel des Kontextes. Diese Aufgabe übernimmt der Dispatcher.
- Der Scheduler teilt die Prozesse dem Prozessor zu; hierfür gibt es unterschiedliche Verfahren (z.B. FCFS, SJF, SRT oder RR).
- Threads sind Teil von Prozessen und erlauben die nebenläufige Ausführung auf mehreren Prozessoren.
- Systemaufrufe erlauben die Ausführen von Systemroutinen durch sog. Traps.
- IPC ist u.a. möglich durch Shared Memory, Message Queues, Pipes und Sockets.
- Das Betriebssystem stellt in Form von Systemaufrufen Techniken für die IPC zur Verfügung.

Aufgaben

1. Welche Scheduling-Verfahren eignen sich für Stapelverarbeitung, welche für interaktive Verarbeitung? Welche sind preemptive, welche non-preemptive?
2. Erläutern Sie die einzelnen Spalten, die unter Linux bei `ps -eF` ausgegeben werden.
3. Welche Möglichkeiten der IPC gibt es a) auf demselben Rechner und b) zwischen unterschiedlichen Rechnern?
4. Wie läuft ein Systemaufruf ab?
5. Wieso nutzt man in C `fopen()`, um eine Datei zu öffnen, und nicht direkt `open()`?

Aufgaben

6. Vergleichen Sie FCFS, SJF und RR (Quantum: 3). Geben Sie jeweils für die folgenden Prozesse an, wann sie die CPU nutzen können. Berechnen Sie die tatsächliche Laufzeit, die Wartezeit und die mittlere Wartezeit.

Prozess	Start	Dauer
P1	0	5
P2	0	2
P3	1	4
P4	3	3