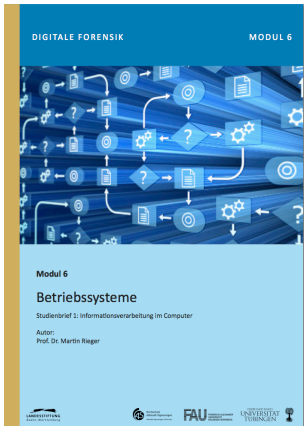
Betriebssysteme

SS 2013

Hans-Georg Eßer
Dipl.-Math., Dipl.-Inform.

SB 1 (14.03.2013)
Überblick zu Betriebssystemen



Über den Dozenten

Hans-Georg Eßer

- Dipl.-Math. (RWTH Aachen, 1997)
  Dipl.-Inform. (RWTH Aachen, 2005)
  Fachjournalist (DFJS Berlin, 2006)
- Chefredakteur Linux-Zeitschrift (seit 2000)
- Autor diverser Linux-Bücher
- seit 2006 Dozent/Lehrbeauftragter an verschiedenen Hochschulen: Betriebssysteme, Rechnerarchitektur, Systemprogrammierung, IT-Infrastrukturen, Informatik-Grundlagen
- Seit 2010 Doktorand (Univ. Erlangen-Nürnberg)

Einführung und Motivation

Wofür Betriebssysteme?

Theorie der Betriebssysteme

- Wie funktionieren Betriebssysteme?
- Konsequenzen für Anwendungsentwickler
- Sicherheitsprobleme
- Auswahl eines geeigneten Betriebssystems

... und das Thema ist auch an sich spannend

Aufgaben von Betriebssystemen (1)

- Abstraktionsschicht zwischen Hardware und Programmen (➞ virtuelle Maschine)
- Verwaltung der vorhandenen Ressourcen
- Einheitlicher Zugriff auf Geräte einer groben Kategorie, z. B.:
  - *Datenträger* (Plattenpartition, CD, DVD, Diskette, USB-Stick, Netzwerk-Volume)
  - *Drucker* (PostScript-Laser, Etikettendrucker, Billig-Tintenstrahler, ...)

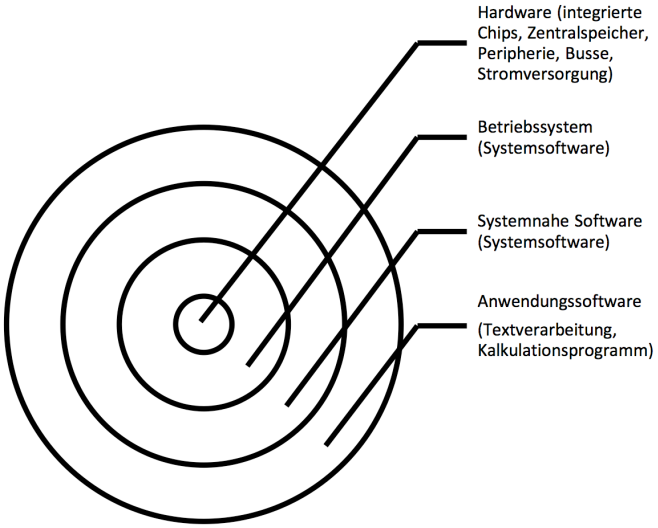
Aufgaben von Betriebssystemen (2)

- Schützt Hardware vor direkten Zugriffen (➞ defekte oder bösartige Software)
- Befreit Software vom Zwang, die Hardware im Detail zu kennen
- Zulassen mehrerer Benutzer und Abgrenzung (Multi-user)
- Parallelbetrieb mehrerer Anwendungen (Multi-tasking): faire Aufteilung der Ressourcen

Aufgaben von Betriebssystemen (3)

- Virtualisierung des Speichers
  - ➞ Anwendungen müssen nicht wissen, wo sie im Hauptspeicher liegen
  - ➞ Speicher über phys. RAM hinaus verfügbar (Swap etc.)

1.3 Schalenmodell



1.3 Schalenmodell

- **Systemsoftware:**
  „Betriebssystem“ vs. „Systemnahe Software“?
- **Betriebssystem** (im engeren Sinne) ist nur der BS-Kern (vmlinuz, ntoskrnl.exe)
- **Systemnahe Software** wird für den sinnvollen Betrieb des BS benötigt, gehört aber nicht zum Kern
  - Shell, bei Desktops auch das Fenstersystem
  - Syslog-Daemon

1.3 Schalenmodell

Ü

Übung 1.1
Ordnen Sie die folgenden Programme einem Ort im Schalenmodell zu:
USB-Treiber, C++-Compiler, Tabellenkalkulations-Programm, Windows Explorer, Linux Korn-Shell, DOS-Command.Com
Nennen Sie für die Betriebssysteme UNIX und MS Windows XP je zwei Beispiele für: Anwendungssoftware, Systemsoftware und systemnahe Software.

	Hard-ware	Betriebs-system	System-nahe Software	Anwen-dungs-Software	Begründung
USB-Treiber	☐	X	☐	☐	Teil des BS-Kerns
C++-Compiler	☐	☐	X	☐	SW im Umfeld des BS
Tabellenkalkulation	☐	☐	☐	X	Kein enger Bezug zum Rechnersystem
Windows Explorer	☐	☐	X	☐	SW im Umfeld des BS, aber nicht Teil des BS
Linux Korn-Shell	☐	☐	X	☐	SW im Umfeld des BS, aber nicht Teil des BS
DOS-Command.Com	☐	☐	(X)	☐	SW im Umfeld des BS, aber nicht Teil des BS

1.3 Schalenmodell

	UNIX	MS Windows XP
Anwendungssoftware	Firefox, LibreOffice	IE, Firefox, MS-Word
Systemsoftware	Kernel, Treiber	Treiber, HAL Manager (Teil des Executive Layer)
systemnahe Software	Compiler, Editor, Debugger	Compiler, Editor, Debugger

1.4.2 Ziele eines BS

Kontrollaufgabe 1.1

Nennen Sie je ein Beispiel für die genannten Arten der Anwenderunterstützung sowie je einen Fall, in dem das Betriebssystem die obigen Anforderungen an die Zuverlässigkeit nicht erfüllt hat.

K

- Abstraktion der HW: Zugriff auf diverse Datenträger (Platte, DVD, USB-Stick) für Anwender transparent und „gleichartig“
- Bereitstellen von Dienstfunktionen: Drucken aus Anwendungen
- Verbergen irrelevanter Details: Blockadressen von Dateien

✗ Schutzmechanismen für Prozesse: Windows 3.1 schützt Speicherbereiche anderer Prozesse nicht

✗ Abfangen von Exceptions: MS-DOS kann das nicht

✗ Rechnerblockade: Windows 3.1 …

1.4.2 Ziele eines BS

K

Kontrollaufgabe 1.2

Was ist der Unterschied zwischen paralleler und quasiparalleler Verarbeitung im Rechner? Welche Voraussetzungen sind an eine CPU zu stellen, wenn im betreffenden Rechner 50 Prozesse – dies ist für einen Arbeitsplatzrechner typisch - gleichzeitig ablaufen?

- **parallel:** jeder Thread/Prozess erhält eine eigene CPU (oder einen CPU-Core)
- **quasiparallel:** es gibt weniger CPUs/Kerne als Threads/Prozesse; Eindruck der Parallelität wird durch schnellen Prozesswechsel erzeugt
- 50 Prozesse: CPU schnell genug, dass Wechsel zwischen den Prozessen nicht auffällt

1.4.3 Aufgaben eines BS

K

Kontrollaufgabe 1.3

Ein Rechner in einem Heißgetränkeautomat hat folgende Aufgaben: Steuerung der Heizung für konstante Temperatur, Annahme von Anforderungen, Annahme von Münzen, Überprüfung der Münzen mit der geforderten Summe, ggf. Geldrückgabe und Restgeldausgabe, Steuerung der Getränkeausgabe, Wartungs- und Diagnosefunktion. Erläutern Sie, welche Aufgaben für das auf dem Rechner laufende Betriebssystem besonders wichtig und welche weniger wichtig sind.

- wichtig: Temperatursteuerung (zu kalt, zu warm?), Ausgabesteuerung (Becher halb leer, Becher läuft über?), Diagnose (ggf. Selbstabschaltung)
- weniger wichtig: alles andere – Versagen der restlichen Komponenten nur ärgerlich...
- Für den Benutzer gibt es natürlich andere Prioritäten

Begriffe

Single-Tasking / Multitasking (Einprogramm- / Mehrprogrammbetrieb):

Wie viele Programme laufen „gleichzeitig“?

- MS-DOS, CP/M: 1 Programm
- Windows, Linux, ...: Viele Programme

Single-Processing / Multi-Processing:

Hilft der Einsatz mehrerer CPUs?

- Windows 95/98/Me: 1 CPU
- Windows 2000, XP, Vista, Win 7/8, Linux, OS X, ...: Mehrere CPUs

1.6.1 Prozesse und Threads

Prozess:

- Konzept nötig, sobald >1 Programm läuft
- Programm, das der Rechner ausführen soll
- Eigene Daten
- von anderen Prozessen abgeschottet
- Zusätzliche Verwaltungsdaten

1.6.1 Prozesse und Threads

Prozess:

Programm, das in den Speicher geladen wurde und ausgeführt wird / werden soll

Mehr als nur der Programmcode:

- Eigene Daten
- Stack
- Programmzähler
- Umgebung

1.6.1 Prozesse und Threads

Thread:

Ähnlich wie Prozess, aber:

- mehrere Threads greifen auf gleichen Speicher zu
- Thread-Verwaltung nicht unbedingt im Kernel (➞ weniger Verwaltungs-Overhead)
- User level / Kernel level

1.6.1 Prozesse und Threads

Prozessliste:

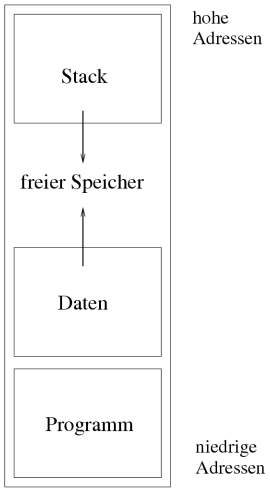
- Informationen über alle Prozesse und ihre Zustände
- Jeder Prozess hat dort einen **Process Control Block** (**PCB**):
  - Identifier (PID)
  - Registerwerte inkl. Befehlszähler
  - Speicherbereich des Prozess
  - Liste offener Dateien und Sockets
  - Informationen wie Vater-PID, letzte Aktivität, Gesamtlaufzeit, Priorität, ...

1.6.1 Prozesse und Threads

Prozess im Detail:

- Eigener Adressraum
- Ausführbares Programm
- Aktuelle Daten (Variableninhalte)
- Befehlszähler (Program Counter, PC)
- Stack und Stack-Pointer
- Inhalt der Hardware-Register (Prozess-Kontext)

1.6.1 Prozesse und Threads



- Daten: dynamisch erzeugt
- Stack: Verwaltung der Funktionsaufrufe
- Details: siehe Kapitel Speicherverwaltung
- Stack und Daten „wachsen aufeinander zu“

1.6.1 Prozesse und Threads

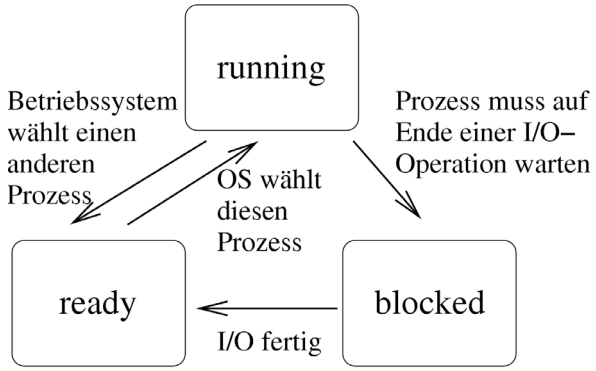
Zustände

- **laufend / running:** gerade aktiv
- **bereit / ready:** würde gerne laufen
- **blockiert / blocked / waiting:** wartet auf I/O

- **suspendiert:** vom Anwender unterbrochen
- **schlafend / sleeping:** wartet auf Signal (IPC)
- **ausgelagert / swapped:** Daten nicht im RAM

1.6.1 Prozesse und Threads

Zustandsübergänge



1.6.3 Prozesse unter Unix

Hierarchien

- Prozesse erzeugen einander
- Erzeuger heißt Vaterprozess (parent process), der andere Kindprozess (child process)
- Kinder sind selbständig (also: eigener Adressraum, etc.)
- Nach Prozess-Ende: Rückgabewert an Vaterprozess (vgl. Funktionsaufruf)

1.6.3 Prozesse unter Unix

```
esser@sony:Folien> emacs test.txt &
[3] 24469
esser@sony:Folien> _

[...]

[3]+ Done                    emacs test.txt
```


1.6.3 Prozesse unter Unix

```
esser@sony:Folien> jobs
[1]- Running                 xpdf -remote sk bs02.pdf &
[2]+ Running                 nedit kap02/index.tex &

esser@sony:Folien> jobs -l
[1]- 8103 Running            xpdf -remote sk bs02.pdf &
[2]+ 20568 Running           nedit kap02/index.tex &

esser@sony:Folien> ps w|grep 8103|grep -v grep
 8103 pts/15 S       5:27 xpdf -remote sk bs02.pdf
```


1.6.3 Prozesse unter Unix

```
> ps auxw
USER     PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root       1  0.0  0.0    720    92 ?        S    Jun24   0:01 init [5]
root       2  0.0  0.0      0     0 ?        SN   Jun24   1:09 [ksoftirqd/0]
root       3  0.0  0.0      0     0 ?        S<   Jun24   0:11 [events/0]
root       4  0.0  0.0      0     0 ?        S<   Jun24   0:00 [khelper]
root       5  0.0  0.0      0     0 ?        S<   Jun24   0:00 [kthread]
root       7  0.0  0.0      0     0 ?        S<   Jun24   0:02 [kblockd/0]
root       8  0.0  0.0      0     0 ?        S<   Jun24   0:00 [kacpid]
root     128  0.0  0.0      0     0 ?        S<   Jun24   0:00 [aio/0]
[....]
esser   5733  0.2 12.2  82420 63428 ?        S    Jul24   4:05 /usr/bin/opera
root    2670  0.3  0.0   1368   300 ?        Ss   08:24   2:39 zmd
/usr/lib/zmd
esser   8037  0.0  0.6   6452  3384 pts/13 S+   11:23   0:05 ssh -X amd64
```


1.6.3 Prozesse unter Unix

```
> pstree -p
init(1)-+-acpid(2266)
        |-auditd(2727)---{auditd}(2728)
        |-cron(3234)
        |-cupsd(2706)
        |-gpg-agent(4031)
        |-hald(2309)-+-hald-addon-acpi(2616)
        |            |-hald-addon-stor(2911)
        |            `-hald-addon-stor(2914)
        |-kded(4079)
        |-kdeinit(4072)-+-artsd(7184)
        |               |-kio_file(4402)
        |               |-klauncher(4077)
        |               |-konqueror(22430)
        |               |-konsole(11064)-+-bash(11065)---ssh(31205)
        |               |                |-bash(11119)---sux(11444)---bash(11447)
        |               |                |-bash(11137)
        |               |                |-bash(25637)-+-ssh(4522)
        |               |                |             `-xmms(7169)-+-{xmms}(7170)
        |               |                |                          `-{xmms}(7171)
        |               |                `-bash(15608)
        |               |-konsole(4773)-+-bash(4774)---ssh(8037)
        |               |               |-bash(8040)---ssh(8058)
        |               |               `-bash(8061)-+-less(15188)
        |               |                            |-nedit(9628)
        |               |                            `-xpdf(8103)
```

1.6.3 Prozesse unter Unix

- Programm unterbrechen: Strg-Z
- Fortsetzen im Vordergrund: **`fg`**
- Fortsetzen im Hintergrund: **`bg`**
- Signale an Prozess schicken: **`kill`**
  - unterbrechen (STOP), fortsetzen (CONT)
  - beenden (TERM), abschießen (KILL)
- Verbindung zu Vater lösen: **`disown`**

1.6.3 Prozesse unter Unix

```
> kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46)
SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-
14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-
10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```


1.6.3 Prozesse unter Unix

Was ist ein Thread?

- Aktivitätsstrang in einem Prozess
- einer von mehreren
- Gemeinsamer Zugriff auf Daten des Prozess
- aber: Stack, Befehlszähler, Stack Pointer, Hardware-Register separat pro Thread
- Prozess-Scheduler verwaltet Threads – oder nicht (Kernel- oder User-level-Threads)

1.6.3 Prozesse unter Unix

Warum Threads?

- Multi-Prozessor-System: Mehrere Threads echt gleichzeitig aktiv
- Ist ein Thread durch I/O blockiert, arbeiten die anderen weiter
- Besteht Programm logisch aus parallelen Abläufen, ist die Programmierung mit Threads einfacher

1.6.3 Prozesse unter Unix

Zwei unterschiedliche Aktivitätsstränge: Komplexe Berechnung mit Benutzeranfragen

Ohne Threads:

```
while (1) {
  rechne_ein_bisschen ();
  if benutzereingabe (x) {
    bearbeite_eingabe (x);
  }
}
```


1.6.3 Prozesse unter Unix

Komplexe Berechnung mit Benutzeranfragen

Mit Threads:

```
T1:

while (1) {
  rechne_alles ();
}
```

```
T2:

while(1) {
  if benutzereingabe (x) {
    bearbeite_eingabe (x);
  }
}
```


1.6.3 Prozesse unter Unix

Server-Prozess, der viele Anfragen bearbeitet

- Prozess öffnet Port
- Für jede eingehende Verbindung: Neuen Thread erzeugen, der diese Anfrage bearbeitet
- Nach Verbindungsabbruch Thread beenden
- Vorteil: Keine Prozess-Erzeugung (Betriebssystem!) nötig

1.6.3 Prozesse unter Unix

Ein Prozess, neun Threads:

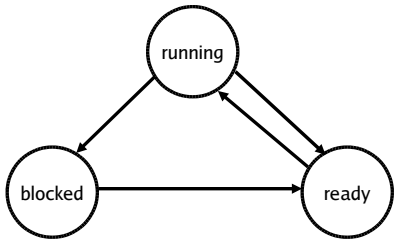
```
[esser:~]$ ps -eLf | grep mysql
UID        PID  PPID   LWP  C NLWP STIME TTY          TIME CMD
-----------------------------------------------------------------------------------------
root     27833     1 27833  0    1 Jan04 ?        00:00:00 /bin/sh /usr/bin/mysqld_safe
mysql    27870 27833 27870  0    9 Jan04 ?        00:00:00 /usr/sbin/mysqld --basedir=/usr
mysql    27870 27833 27872  0    9 Jan04 ?        00:00:00 /usr/sbin/mysqld --basedir=/usr
mysql    27870 27833 27873  0    9 Jan04 ?        00:00:00 /usr/sbin/mysqld --basedir=/usr
mysql    27870 27833 27874  0    9 Jan04 ?        00:00:00 /usr/sbin/mysqld --basedir=/usr
mysql    27870 27833 27875  0    9 Jan04 ?        00:00:00 /usr/sbin/mysqld --basedir=/usr
mysql    27870 27833 27876  0    9 Jan04 ?        00:00:00 /usr/sbin/mysqld --basedir=/usr
mysql    27870 27833 27877  0    9 Jan04 ?        00:00:00 /usr/sbin/mysqld --basedir=/usr
mysql    27870 27833 27878  0    9 Jan04 ?        00:00:00 /usr/sbin/mysqld --basedir=/usr
mysql    27870 27833 27879  0    9 Jan04 ?        00:00:00 /usr/sbin/mysqld --basedir=/usr

[esser:~]$
```

PID: Process ID
PPID: Parent Process ID
LWP: Light Weight Process ID (Thread-ID)
NLWP: Number of Light Weight Processes

Thread-Zustände

- Prozess-Zustände suspended, sleeping, swapped etc. nicht auf Threads übertragbar (warum nicht?)
- Darum nur drei Thread-Zustände



1.6.4 Synchronisation

K

Kontrollaufgabe 1.5

Erläutern Sie die Notwendigkeit der Synchronisation für folgende Fälle:

a) Mehrere Prozesse greifen schreibend und/oder lesend auf die gleiche Datei zu (Leser-Schreiber-Problem)
b) Ein Prozess ist Sender und schickt Daten in einen Datenpuffer mit endlicher Größe (Erzeuger-Verbraucher-Problem)
c) Mehrere Prozesse wollen gleichzeitig auf das LAN zugreifen (gegenseitiger Ausschluss).

a) Lost-Update-Problem

b) Puffer-Überlauf, korrekte Lese-/Schreib-Positionen für Puffer

c) Es gibt auch für Netzwerkpakete einen Puffer...

1.6.5 Betriebsarten

K

Kontrollaufgabe 1.6

Welche Vorteile hat der Einprogrammbetrieb? Welches Ziel wird beim Einprogrammbetrieb prinzipiell verfehlt?

- Programm kann alle Ressourcen des Rechners exklusiv nutzen
- BS kann dem Programm direkten Zugriff auf Hardware erlauben (Abstraktion fällt weg)
- kein Scheduling nötig
- ✗ schlechte Auslastung der HW, z. B. beim Warten auf I/O-Operationen

1.6.5 Betriebsarten

Kontrollaufgabe 1.7

Nennen Sie weitere sinnvolle Einsatzgebiete für den Batchbetrieb.

K

- High Performance Computing (HPC): Komplexe / zeitaufwändige Berechnungen werden als Jobs ausgeführt; Benachrichtigung des Benutzers nach Ausführung
- Nutzung eines Arbeitsplatzrechners zu Nicht-Bürozeiten (ab 20 Uhr: Aufträge bearbeiten)

1.6.6 Benutzerrechte

Unter Linux/Unix:

- Benutzer und Gruppen
- Unix-Dateiattribute
  (rwx für Besitzer, Gruppe, sonstige)
- numerische Rechte
- SUID- und SGID-Bits
- erweiterte Attribute

Dateien, Benutzer, Gruppen

- Jede Datei
  - … gehört einem Benutzer (Besitzer, **u**ser)
  - … und zu einer Gruppe (**g**roup)
- Benutzer können Mitglieder in verschiedenen Gruppen sein
- Zugriffsrechte entscheiden, ob eine Datei gelesen (**r**ead), geschrieben (**w**rite) oder ausgeführt (e**x**ecute) werden darf

Gruppen (1)

- In welchen Gruppen bin ich Mitglied?

```
$ groups
fom cdrom floppy audio dip video plugdev netdev
powerdev scanner
```

- Mitgliedschaft durch Einträge in `/etc/group` geregelt:

```
$ grep forensik /etc/group
cdrom:x:24:forensik
floppy:x:25:forensik
audio:x:29:forensik
...
forensik:x:1002:forensik
```

- Gruppenmitgliedschaft bearbeiten:
  manuell oder (besser!) mit `gpasswd`

Gruppen (2)

- Gruppe mit `gpasswd -a` *`user group`* (**a**dd) ergänzen:

```
# gpasswd -a forensik neugr
Benutzer forensik wird zur Gruppe neugr
hinzugefügt.
# groups fom
forensik cdrom floppy audio dip video plugdev
netdev powerdev scanner neugr
```

- Entfernen einer Gruppenmitgliedschaft:
  `gpasswd -d` *`user group`* (**d**elete)

```
# gpasswd -d forensik neugr
Benutzer forensik wird aus der Gruppe neugr
entfernt.
```

Gruppen (3)

- Jeder Benutzer ist in einer Standardgruppe Mitglied. Welche ist das?

```
$ id
uid=1002(forensik) gid=1002(forensik)
Gruppen=1002(forensik),24(cdrom),25(floppy),
29(audio),30(dip),...
```

- Zwei Standards für Standardgruppe
  - Debian-System: Jeder Benutzer hat seine eigene Standardgruppe (User: `forensik`, Group: `forensik`)
  - andere Systeme: Standardgruppe `users` für alle „normalen“ Benutzer
- Im Namen der Standardgruppe handeln Sie, bis Sie mit `newgrp` die Gruppe ändern.

Gruppen (4)

- Neue Gruppen kann der Administrator mit `groupadd` erzeugen, um Kooperation von Teams zu erleichtern
  - z. B. mit Dateien, die für alle Gruppenmitglieder (und nur diese) les- und schreibbar sind
- Beispielszenario folgt ...

Beispielszenario (1)

- Gruppe `profs`: Mitglieder `prof1`, `prof2`
- Gruppe `studis`: Mitglieder `anna`, `tom`, `fritz` und (!) `prof1`, `prof2`
- Ziele:
  - `profs`-Mitglieder können Daten untereinander austauschen und teilweise auch Studenten zur Verfügung stellen
  - `studis`-Mitglieder können Daten untereinander austauschen und auf die von Profs zur Verfügung gestellten Skripte, Aufgaben etc. zugreifen

Beispielszenario (2)

- Verzeichnisstruktur

```
/srv/profs/
/srv/profs/intern/                ; Austausch der Profs untereinander
/src/profs/intern/klausuren/
/srv/profs/public/                ; Lesezugriff für Studenten möglich
/srv/profs/public/skripte/
/srv/studis/
/srv/studis/mitschriften/
/srv/studis/pruefungsprot/
```

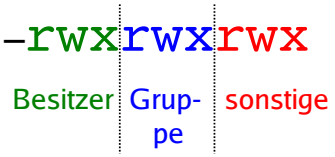
- Gruppenzugehörigkeiten und Zugriffsrechte
  - `/srv/profs/intern`: gehört Gruppe `profs`; lesen und schreiben für `profs` erlaubt, kein Zugriff für `studis`
  - `/srv/profs/public`: gehört Gruppe `profs`; schreiben für `profs` erlaubt, lesen für alle (auch Nicht-Studis)
  - `/srv/studis`: gehört Gruppe `studis`; lesen und schreiben für Gruppenmitglieder erlaubt

Beispielszenario (3)

- Nachteil: keine vernünftige Zugriffsbeschränkung für `/srv/profs/public` möglich
  → ACLs

Unix-Dateiattribute (1)

- Neben Besitzer und Gruppe gibt es noch die sonstigen Systembenutzer (**o**, others)
- ergibt 9 Zugriffsrechte; Notation bei `ls`:



Unix-Dateiattribute (2)

- `chown` (change owner) und `chgrp` (change group) ändern Besitzer und Gruppe einer Datei
- `chmod` (change mode) ändert Zugriffsrechte
- Beispiele:

```
chown user /tmp/log.txt
chgrp www-data /var/www/srv1
chmod o+r /tmp/log.txt
chmod o-rwx,ug+rw /tmp/log.txt
chmod u=rw,g=r,o= /tmp/log.txt
```

- Abkürzung `a` (all) für `ogu` (`chmod a=rw` ...)

Unix-Dateiattribute (3)

- numerische Rechte:
  - Leserecht: 4 (2^2)
  - Schreibrecht: 2 (2^1)
  - Ausführrecht: 1 (2^0)
  - aufaddieren, z. B.: rw = Lesen/Schreiben: 4+2=6
- für Benutzer, Gruppe und Sonstige: **nnn**
  - z. B. **640**:
    - Benutzer: 6 = lesen + schreiben (nicht ausführen)
    - Gruppe: 4 = lesen (nicht schreiben, nicht ausführen)
    - Sonstige: 0 = nichts

Unix-Dateiattribute (4)

- `chmod` mit numerischen Rechten nutzen
  - `rw- r-- ---` = 640 (4+2+0, 4+0+0, 0+0+0)
  - `chmod u=rw,g=r,o= /tmp/log.txt`
    `chmod 640 /tmp/log.txt`
- bei der numerischen Angabe kein „Geben“ und „Nehmen“ von Rechten möglich
  (wie mit `chmod u+x` ..., `chmod o-rwx` ...)

Unix-Dateiattribute (5)

- Beim Erzeugen einer Datei werden Standard-rechte gesetzt – welche das sind, bestimmt die UMASK (**u**ser file creation **mask**)

```
$ umask
0022
$ umask a=rw
$ umask
0111
$ touch Datei; ls -l Datei
-rw-rw-rw-  1 esser users 0 2008-12-04 20:48 Datei
$ umask u=rw,g=r,o=
$ umask
0137
$ touch Test; ls -l Test
-rw-r-----  1 esser users 0 2008-12-04 20:50 Test
```

Standard:
Gruppe: nicht schreiben,
Sonstige: nicht schreiben

Unix-Dateiattribute (6)

- umask wird von 666 (`rw-rw-rw-`: Standard-wert für Dateien) bitweise abgezogen, um konkrete Dateirechte zu berechnen;
- Ausführrecht wird beim Erzeugen einer Datei nie vergeben
- Linux unterstützt diese klassischen Unix-Dateiattribute und einige zusätzliche...

Unix-Dateiattribute (7)

- Dateiattribute nur auf echten Unix-Dateisystemen nutzbar – auf Windows-Datenträgern nur stark eingeschränkt:

```
# mount | grep windows
/dev/sda3 on /windows/D type vfat (rw,gid=100,umask=0002)
# touch /windows/D/Testdatei
# ls -l /windows/D/Testdatei
-rwxrwxr-x  1 root users 0 2006-12-04 21:07 /windows/D/Testdatei
# chmod a-rwx /windows/D/Testdatei
# ls -l /windows/D/Testdatei
----------  1 root users 0 2006-12-04 21:07 /windows/D/Testdatei
# umount /windows/D; mount /windows/D; ls -l /windows/D/Testdatei
-r-xr-xr-x  1 root users 0 2006-12-04 21:07 /windows/D/Testdatei
```

- Windows kennt kein Ausführattribut – wohl aber ein Read-Only-Attribut

Unix-Dateiattribute (8)

- Bedeutung der Attribute für Verzeichnisse:
  - **r**ead: Verzeichnisinhalt lesen (`ls` in einem Verzeichnis ausführen)
  - **w**rite: Verzeichnisinhalt ändern (z. B. neue Datei erzeugen, Datei umbenennen)
  - e**x**ecute: Verzeichnis betreten, also zum aktuellen Arbeitsverzeichnis machen (`cd`)
  - Standardrechte, von denen die umask abgezogen wird, sind bei Verzeichnissen 777 (denn x = execute steht ja für „Verzeichnis betreten“)

Beispielszenario (4)

- Zurück zum Beispielszenario
  - Ersteinrichtung:

```
root# chown -R root /srv/profs /srv/studis
root# chgrp -R profs /srv/profs/intern
root# chmod ug=rwx,o= /srv/profs/intern
root# chgrp -R profs /srv/profs/public
root# chmod ug=rwx,o=rx /srv/profs/public
root# chgrp -R studis /srv/studis
root# chmod ug=rwx,o= /srv/studis
```

  - neue Dateien erzeugen:

```
prof1$ newgrp profs   # als „profs“-Mitglied arbeiten
prof1$ umask 0007     # Neue Dateien nicht für andere
prof1$ cd /srv/profs/intern
prof1$ touch pruefung.doc
prof1$ ls -l pruefung.doc
-rw-rw---- 1 prof1 profs ...     pruefung.doc
```


SUID, SGID (1)

- Problem: Es gibt Dateien, die Benutzer nur „unter kontrollierten Bedingungen“ ändern dürfen, z. B. die Passwortdatei `/etc/shadow`
  - Änderung an der Datei mit dem Tool `passwd`
  - Dafür sind Root-Rechte nötig
  - Normale Anwender haben keine Root-Rechte
- Zwei Lösungen
  - klassisch: SUID (siehe nächste Folie)
  - neuer: `sudo` (behandeln wir hier nicht)

SUID, SGID (2)

- Ausführbare Dateien (nur Binaries) können ein SUID- (Set User ID) und/oder ein SGID-Bit (Set Group ID) haben
  - SUID: Programm läuft immer mit den Rechten des Dateibesitzers, meist root
  - SGID: Programm läuft immer mit den Gruppenrechten der Dateigruppe (seltener verwendet)
  - Beispiel: `passwd` muss Systemdateien ändern

```
$ ls -l /usr/bin/passwd /etc/shadow
-rwsr-xr-x 1 root root    ...  /usr/bin/passwd
-rw-r----- 1 root shadow  ...  /etc/shadow
```

SUID, SGID (3)

- SUID- und SGID-Bits mit `chmod` setzen

```
# cp /usr/bin/passwd /tmp/mypasswd
# chmod u-s,g+s /tmp/mypasswd
# ls -l /usr/bin/passwd /tmp/mypasswd
-rwsr-xr-x 1 root root  ...  /usr/bin/passwd
-rwxr-sr-x 1 root root  ...  /tmp/mypasswd
```

- s-Bits erscheinen in der `ls`-Ausgabe immer an der Stelle, wo sonst das x steht
- Diese Bits sind bei Shell-Skripten wirkungslos (in einigen älteren Unix-Versionen funktionierte das auch mit Skripten)

1.6.8 Systemaufrufe

- Details zu System Calls in SB 4 (Systemprogrammierung)

K

> Kontrollaufgabe 1.9
>
> Vergleichen Sie einen gewöhnlichen Funktionsaufruf in C mit einem Systemaufruf.

- Funktionsaufruf: läuft komplett im User Mode ab:

  Parameter, Rücksprungadresse auf Stack, dann Sprung in Funktion – am Ende Rückkehr mit `RET`
- System call: Wechsel in Kernel Mode durch spezielle Instruktion (z. B. INT), am Ende Rückkehr mit `IRET`

1.7.5 Virtuelle Maschinen

- Idee: Ein Betriebssystem läuft nicht direkt auf der Hardware, sondern als Gast in einer virtuellen Maschine – unter einem Host-Betriebssystem
- Verschiedene Arten der Virtualisierung
  - Full Virtualization (z. B. VMware, VirtualBox)
  - Hypervisor ohne darunter liegendes Betriebssystem (z. B. VMware ESX Server)
  - Paravirtualisierung (z. B. Xen)

Virtualisierung: Warum? (1)

- Ungenutzte Rechenkapazitäten besser ausnutzen (vgl.: Multi-Processing)
- einheitliche Hardware: BS auf virtueller Maschine muss nicht an konkrete Hardware angepasst werden
- Server-Bereich:
  - Konsolidierung – wenige große Maschinen statt vielen kleinen
  - vereinfachtes Backup aller virt. Maschinen

Virtualisierung: Warum? (2)

- Einfachere Bereitstellung einer neuen Maschine, einfaches Duplizieren
- Snapshots erlauben Rückkehr zu funktionierendem Zustand einer Maschine
- Sicherheit: Voreinander zu schützende Anwendungen besser in separaten VMs als auf einem logischen Rechner laufen lassen
- Für Entwickler: Test auf verschiedenen Plattformen, ohne dafür jeweils 1 PC zu benötigen
- Legacy-Support (alte BS)

Emulation vs. Virtualisierung

- Emulation (von Hardware) ist ein anderes Konzept und nicht mit Virtualisierung zu verwechseln
- Emulator ahmt Hardware inkl. CPU vollständig nach
- Instruktionen werden also im Emulator nicht ausgeführt, sondern „interpretiert“
- Emulatoren können auch Hardware mit abweichenden CPUs emulieren (z. B. C64-Emu)

Emulation über Bibliotheken

- Beispiel WINE („WINE Is Not an Emulator“)
- WINE ersetzt auf Linux-Systemen den Programm-Loader und diverse Bibliotheken (DLLs), die Software unter Windows erwartet.
- WINE ist also kein Hardware-Emulator und stellt auch keine virtuelle Maschine bereit.
- Windows-Programme laufen dank WINE fast „nativ“ auf dem Linux-System.

Virtualisierung + Forensik

- Malware Blue Pill: Rootkit, das sich
  - in ein laufendes System als Hypervisor installiert
  - also das laufende System virtualisiert
  - nicht entdeckbar
  - http://en.wikipedia.org/wiki/Blue_Pill_(software)
- Ansatz auch für Forensik-Maßnahmen nützlich
  - HyperSleuth: erlaubt forensische Live-Analysen
  - zieht Speicherabbild und verschickt es über das Netzwerk
  - Gast behält vollen Zugriff auf I/O-Geräte

Vokabular...

- VMX root mode / non-root mode:
  - Hypervisor läuft im *root mode* – Befehlssatz der CPU funktioniert wie gewohnt
  - Beim Umschalten zum Gastsystem in den *non-root mode* wechseln →
    - verändert das Verhalten einiger Instruktionen
    - manche verursachen eine Exception, wodurch das System in den root mode zurück wechselt (und der Hypervisor die Kontrolle übernimmt)
    - VMCS (Virtual Machine Control Structure) legt diese Einstellungen (sog. *Interception Events*) fest
  - Gute Doku: Edgar Barbosa, *Hypervisor Framework*, H2HC 2008, http://www.h2hc.org.br/repositorio/2008/Edgar.pdf

Restliche Übungen

Ü

Übung 1.2
Ein einfacher Rechner mit einem einfachen Betriebssystem, das einen Basic-Interpreter enthält, ist in der Lage im Einprogramm-Betrieb ein Basic-Programm auszuführen. Erläutern Sie welche Ziele eines Betriebssystems damit erreichbar sind und welche nicht.

Ziele	Beschreibung	erfüllt?
Unterstützung des Anwenders	Die Hardware wird erfolgreich abstrahiert, da der Benutzer einen Basic-Interpreter zur Verfügung hat. Der Basic-Interpreter bewirkt auch, dass irrelevante Details des Rechners verborgen werden und dass Dienstfunktionen für den Benutzer bereitgestellt werden	Ja
Optimierung der Rechenauslastung	Da hier nur ein Einprogramm-Betrieb möglich ist, kann die Rechnerauslastung nicht optimal erreicht werden.	Nein

Restliche Übungen

Ziele	Beschreibung	erfüllt?
Zuverlässigkeit	Der Schutz gegen Störung zwischen mehreren Prozessen ist durch den Einprogramm-Betrieb einfach möglich.	Ja
	Ob Ausnahmesituationen abgefangen werden, kommt auf den Basic-Interpreter an. Kann er dies sicherstellen, ist auch dieser Punkt erfüllt.	ggf. Ja
	Da nur ein einziges Programm auf einmal laufen kann, kann es passieren, dass der laufende Prozess bei einem Fehler den Rechner blockiert. In diesem Fall ist es nicht über das Betriebssystem zu bewirken, dass der Prozess abgebrochen wird (z.B. durch Strg + C)	Nein
Portabilität	Durch den Basic-Interpreter ist es ohne weitere Code-Änderungen möglich, ein Programm auch auf einer anderen Plattform mit Basic-Interpreter zu starten und nutzen.	Ja

Restliche Übungen

Ü

Übung 1.3
Ein Bekannter argumentiert, sein Textverarbeitungsprogramm erfülle die Definition des Betriebssystems. Das Programm kann u.a. Texte (von Tastatur und Scanner) erfassen, Texte auf verschiedenen Druckern ausgeben, Texte speichern und Texte per E-Mail versenden. Nennen Sie insgesamt mindestens 4 Argumente, die für oder gegen die Behauptung sprechen.

Die Textverarbeitung erfüllt einige Funktionalitäten eines BS, z. B.

- Bereitstellen von Dienstfunktionen
- Speichern und Laden von Daten
- Laden und Ausführen spezieller Programme
- Abstraktion von Daten in Form von Dateien
- Verbergen irrelevanter Details (z. B. Speicherort der Daten auf der Festplatte)

Aber: …

Restliche Übungen

Andere Funktionalitäten, die ein BS erfüllen muss, erfüllt die Textverarbeitung nicht (oder stützt sich auf Dienste des BS):

- Textverarbeitung nutzt verschiedene Systemfunktionen, die vom darunterliegenden Betriebssystem zur Verfügung gestellt werden (Dateien lesen/schreiben, Tastatureingabe, Bildschirmausgabe, …)
- Die Optimierung der Rechenauslastung kann das Programm selbst nicht erreichen, da es nur als ein Prozess läuft.
- Das Programm kann auch nicht verhindern, dass bei einem Fehler im Programm der Rechner blockiert wird.
  → Zuverlässigkeit durch das Programm selbst nicht gewährleistet.

Restliche Übungen

Ü

Übung 1.4
Was versteht man im Zusammenhang eines Betriebssystems unter Betriebsmittel? Welche Betriebsmittel kennen Sie? Mit welchen Mitteln können Sie die Belegung der Betriebsmittel unter Unix bzw. unter Windows sichten?

Betriebsmittel sind alle Komponenten und Leistungen eines Rechnersystems, die benötigt werden, um Dienste für den Anwender zu erbringen. Zu den Betriebsmitteln zählen z. B. Prozessoren (CPU-Zeit), Arbeitsspeicher, Plattenplatz oder andere Speichermedien, Ein- und Ausgabegeräte.

Sichtung der Betriebsmittel unter Unix: top, ps, df, du ...

Sichtung der Betriebsmittel unter Windows: Taskmanager, Gerätemanager, ...

Restliche Übungen

Ü

Übung 1.5
Nennen Sie die Vor- und Nachteile eines monolithischen Betriebssystems gegenüber einem Mikrokernel-Betriebssystem. Wie müsste der Betriebssystemkern von Linux umgebaut werden, damit ein Mikrokernel-Betriebssystem entsteht? Welche Vor- und Nachteile wären von der Umstellung zu erwarten?

Vorteile monolithisches BS vs. Mikrokernel-BS	Nachteile monolithisches BS vs. Mikrokernel-BS
Hohe Ausführungsgeschwindigkeit gegenüber einem Mikrokernel	Schlechte Speichereffizienz, da auch nicht benötigte Komponenten in den Speicher geladen werden müssen.
Einfache Schnittstellen zur Kommunikation zwischen den einzelnen Komponenten des Kernels, da alle Betriebssystemfunktionen in einem Modul enthalten sind	Struktur unübersichtlich, da alle Komponenten in einem Modul zusammengefasst sind. Keine klare Trennung zwischen z.B. Netzwerkfunktionen und Dateisystemverwaltung.

Restliche Übungen

Vorteile monolithisches BS vs. Mikrokernel-BS	Nachteile monolithisches BS vs. Mikrokernel-BS
Alle Betriebssystemfunktionen sind zusammengefasst, wodurch der Kernel nach außen klar vom Rest des Systems abgetrennt ist	Aktualisierung einzelner Kernel-Komponenten nicht möglich. Es muss immer der gesamte Kernel aktualisiert werden.
	Schlechte Wartbarkeit und Erweiterbarkeit
Es kann via zugeladenen Treibern keine Malware oder instabile Software in den Kernelbereich gelangen	Bei Sicherheitslücken in einer Komponente ist die Sicherheit des gesamten Kernels gefährdet.

Praktische Übungen

Arbeitsblatt 1