Was ist UNIX ?

Geschichte

- Entwickelt 1969 von Ken Thompson bei Bell Laboratories
- Ursprünglich auf einer PDP–7 (Kleinrechner der Firma DEC) in Assembler geschrieben
- 1973 wurde UNIX von Ritchie und Thompson in C umgeschrieben und auf die PDP–11 (DEC) übertragen, nur ein kleiner Teil des Systemkerns (*Kernel*) ist noch in Assembler geschrieben ⇒ UNIX kann relativ einfach auf andere Hardware portiert werden
- Die Weiterentwicklung des Systemkerns und der meisten Dienstprogramme erfolgte seit dieser Zeit in C

Warum ein neues Betriebssystem ?

- Als UNIX entwickelt wurde, war Batch–Betrieb Stand der Dinge.
- Ziel: Ein System zu entwickeln auf dem meherere Programmierer im Team und im Dialog mit dem Rechner arbeiten, Programme entwickeln, korrigieren und dokumentieren können (kein Batch–Betrieb mehr, sondern Time–Sharing).
- Resultat: UNIX als Mehrbenutzersystem (Multitasking–Betriebssystem)
- heute: Sehr viele UNIX Weiterentwicklungen und UNIX ähnliche Betriebssysteme

⇒ Wir werden SunOS 4.1 auf SPARCstation ELC's benutzen

Gemeinsamkeiten von MS–DOS und UNIX

MS–DOS Kommandos

Befehl	Bedeutung
cd	**c**hange **d**irectory
mkdir	**m**a**k**e **dir**ectory
rmdir	**r**e**m**ove **dir**ectory
copy	**copy**
del	**del**ete
type,more	Inhalt einer Datei anzeigen
dir	(**dir**ectory) Inhalt eines Verzeichnisses anzeigen

UNIX – Kommandos

Befehl	Bedeutung
cd	**c**hange **d**irectory
mkdir	**m**a**k**e **dir**ectory
rmdir	**r**e**m**ove **dir**ectory
cp	**c**o**p**y
rm	**r**e**m**ove
cat,more,less	Inhalt einer Datei anzeigen
ls	(**ls**t) Inhalt eines Verzeichnisses anzeigen

Die Befehlsnamen sind meistens Abkürzungen der englischen Befehle.

Die Filesysteme von MS–DOS und Unix sind beide hierarchisch aufgebaut.

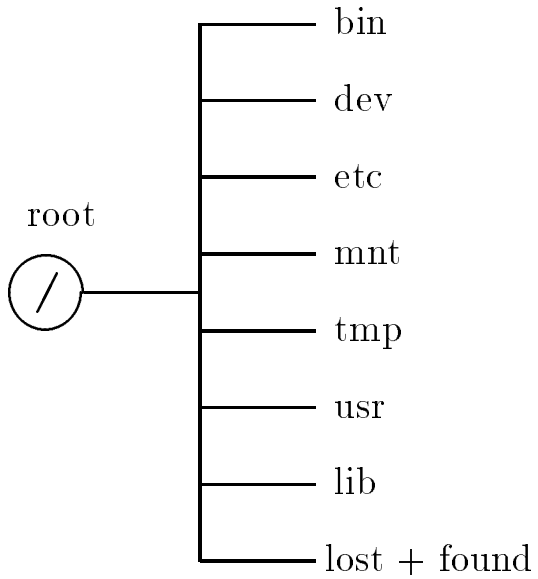
Zu beachten:

Unix unterscheidet zwischen Groß– und Kleinbuchstaben !!!

Beispiel für ein Unix–Dateisystem

Die Wurzel (**root**) des Dateisystems ist das Verzeichnis mit dem Namen '/'. In diesem Verzeichnis liegen neben dem ausführbaren Unixsystem (*Kernel*) in der Regel folgende Unterverzeichnisse:

- **bin**: Wichtige Dienstprogramme
- **dev**: Geräteeinträge (*special files*)
- **etc**: Systemverwaltungsprogramme und Systeminformationsdateien
- **mnt**: Leeres Verzeichnis in das vorübergehen Datenträger eingehängt werden können.
- **tmp**: Temporäre Dateien (werden von zahlreichen Programmen angelegt und existieren nur zu deren Laufzeit)
- **usr**: Weiteres Dateisystem
- **lib**: Enthält einen Teil der Systembibliotheken
- **lost+found**: Das Filesystemcheckprogramm trägt hier Dateien ein, die keine Verweise mehr auf ihren Dateikopf haben.



Einmal Unix und zurück

- Jeder Benutzer erhält einen *login*–Namen und ein geheimes Paßwort. Der *login*–Name ist in den meisten Fällen eine Abkürzung des vollen Names.

- Das Paßwort sollte neben den Buchstaben des Alphabetes noch einige andere Zeichen enthalten (z. B. brg%:-)h).

- Zugang zum System (*login*):
  Der Benutzer wird aufgefordert seinen *login*–Namen einzugeben,

  login:

  danach fordert das System das Paßwort des Benutzers.

  password:

  Das Paßwort erscheint nicht auf dem Bildschirm (kein echo). Falls ein falsches oder falsch geschriebenes Paßwort eingegeben wird, erscheint die Nachricht:

  login failed

  und die *login*–Prozedur wird wiederholt.

- Nach einem erfolgreichen 'Einloggen' erscheint ein *Prompt* z. B. %. (Zur Erinnerung das MS–DOS Prompt war: `C:\`).

- Das Prompt wird von einer *Shell* (Kommandointerpreter) erzeugt. Die *Shell* wartet nun auf die Eingabe eines Befehls.

- Nach getaner Arbeit ... muß man sich vom System abmelden, dazu gibt es den Befehl *logout*:

  % logout

Die Syntax von Unix–Befehlen

- Eine *Kommandozeile* besteht aus einem oder mehreren Wörtern. Ein Wort ist dabei eine Folge von Zeichen ohne Zwischenraum: who am i ⇒ 3 Wörter; whoami ⇒ 1 Wort.
- Die Shell interpretiert das erste Wort als Befehl, alle nachfolgenden Wörter (ausgenommen Sonderzeichen) als Parameter.
- Es gibt zwei Arten von Parametern:
  - normale Parameter, z. B. Dateinamen, Verzeichnisnamen

    `ls *.txt`
    ⇒ listet alle Dateien mit der Endung .txt auf
  - zusätzliche Angaben zur Verarbeitung ⇒ *Optionen.*

    `ls -l`
    ⇒ listet alle Dateien im 'long format' auf

    Optionen werden durch ein vorangestelltes '–' Zeichen markiert (vgl. MS–DOS '/' z. B. `dir/p`).
- Aufbau von Kommandos und Programmaufrufen unter Unix:

```
command_name -options arg_1 arg_2 ...arg_n <CR>
```

Im Unterschied zu MS–DOS gilt für alle Unix–Befehle:

> Zwischen Befehl und Parametern muß ein Leerzeichen stehen!

Der Unix–Befehl 'passwd'

Jeder Benutzer eines Unixsystems besitzt neben seinem login–Namen ein geheimes Paßwort.

- Beim ersten Einloggen an einer Unix–Station sollte man sein Paßwort, das man vom Systemverwalter erhalten hat, ändern. Dazu dient der Befehl `passwd`.

  % `passwd`

  Das System meldet daraufhin

  ```
  Changing NIS password for hugo
  Old password:
  ```

  das alte Paßwort eingeben (wichtig: aus Sicherheitsgründen wird das Paßwort nicht auf dem Bildschirm angezeigt, d. h. kein echo)

  ```
  New password:
  ```

  das neue Paßwort eingeben

  ```
  Retype new password:
  ```

  und weil's so schön war nocheinmal (man kann sich ja schließlich mal vertippen)

- Neben dem login–Namen (user–name) kann der Benutzer auch seinen 'real name' angeben. Der Befehl dafür lautet `passwd -f`.

  % `passwd -f`

  Das System meldet darauf hin

  ```
  Changing NIS finger information for hugo
  Name [Hugo Müller]:
  ```

  neuen 'real name' eingeben

  ```
  Password:
  ```

  Aus Sicherheitsgründen wird wieder das Paßwort abgefragt.

Der Unix–Befehl 'ls'

Syntax des 'ls'–Befehls:

```
ls [-optionen] [datei ...]
```

Beispiele für die Verwendung des 'ls'-Befehls:

a) `ls` ohne Parameter listet Inhalt des aktuellen Verzeichnisses auf.

```
%ls
Calendar   cmd        hugo.txt   pic        tmp
catch      dipl       musi       tex
%
```

b) `ls pic`: Falls es ein Verzeichnis mit dem Namen pic gibt, wird sein Inhalt ausgegeben.

```
%ls pic
coyote.gif   omann.gif    rix-2.gif    rix-4.gif
%
```

c) `ls *.txt` listet alle Dateien mit der Endung `.txt` im aktuellen Verzeichnis auf.

d) `ls -a` gibt alle Dateien des Verzeichnisses einschließlich der Dateien, die mit '.' beginnen aus.

```
%ls -a
.            .cshrc       .logout      cmd          musi
..           .exrc        .xinitrc     dipl         pic
.alias       .rhosts      catch        hugo.txt     tex
%
```

e) `ls -r` listet rekursiv den Inhalt des Verzeichnisse auf, d. h. auch die Dateien in den Unterverzeichnissen.

f) `ls -F` zeigt ein zusätzliches Zeichen nach jeder Datei an. Die Zeichen geben die Art der Datei an, z. B. '$*$' für ausführbare Dateien, '/' für Verzeichnisse, ...

```
%ls -F
Calendar/  cmd/       hugo.txt   pic/       tmp/
catch*     dipl/      musi/      tex/
%
```

g) `ls -l` Listet die Dateien im 'long format' auf. Das 'long format' beinhaltet neben dem Dateinamen zusätzliche Informationen: (von links nach rechts)

- Block von 10 Zeichen, der die Zugriffsrechte angibt,
- Anzahl der Links auf die Datei,
- Besitzer der Datei,
- Größe der Datei,
- Datum der letzten Änderung,
- Uhrzeit und
- Name der Datei.

```
%ls -l
total 241
-rwxrwxr-x  1 jewe       229376 Oct  6 14:27 catch
drwxr-xr-x  2 jewe          512 Jun 24 16:57 cmd
-rw-r--r--  1 jewe         1406 Oct  6 14:26 hugo.txt
drwxr-xr-x  2 jewe          512 Jun  2  1992 pic
brw-r-----  1 root            0 Apr 22  1991 sd0a
drwxr-xr-x  2 jewe          512 Jun 23 17:17 tex
crw--w----  1 jewe            0 Oct  6 14:58 ttyp0
%
```

Das erste der 10 Zeichen gibt an, um welche Datei es sich handelt: normale Datei (-), zeichenorientiertes Gerät (c), blockorientiertes Gerät (b), Verzeichnis (d), Link (l). Die restlichen neun Zeichen geben die Zugriffsrechte an.

Der Unix–Befehl 'chmod'

Zugriffsrechte

Der Benutzer kann die Zugriffsrechte auf seine eigenen Dateien festlegen und bei Bedarf mit dem Befehl `chmod` ändern.

Geräte und Verzeichnisse werden, wie normale Dateien verwaltet !

Man unterscheidet drei Arten von Zugriffen auf eine Datei:

- Lesen (`(r)ead`)
- Schreiben (`(w)rite`)
- Ausführen (`e(x)ecute`)

Das Zugriffsrecht Ausführen (**x**) bedeutet bei normalen Dateien, daß es erlaubt ist, das Programm zu starten. Bei Verzeichnissen zeigt (**x**) an, daß der Zugriff auf das Verzeichnis und die darunter liegenden Dateien erlaubt ist.

Es gibt drei Benutzerklassen:

- den Besitzer (**u**=login **u**ser)
- die Benutzer der gleichen Gruppe (**g**=**g**roup)
- alle anderen Benutzer (**o**=**o**ther users)

Die Zugriffsrechte einer Datei können für die drei Benutzerklassen unabhängig gesetzt werden. Zum Anzeigen der aktuellen Zugriffsrechte benutzt man den Befehl `ls -l`.
Beispiel:

```
-rw-r--r--  1 jewe         1406 Oct  6 14:26 hugo.txt
```

`rw-r--r--` ⇒ Lese– und Schreibrechte für den Benutzer, nur Leserecht für die Gruppe und alle anderen Benutzer.

Syntax von chmod

```
chmod class operation permission file
```

mit
class $\in$ { **u**ser, **g**roup, **o**thers }
operation $\in$ {+, − ,= } Steht für hinzufügen, streichen, setzten von Zugriffsrechten.
permission $\in$ { **r**, **w**, **x** }

Beispiel: Die Zugriffsrechte für `hugo.txt` sind wie folgt gesetzt:

```
-rw-------  1 jewe          1406 Oct  6 14:26 hugo.txt
```

Ändern der Zugriffsrechte für die Gruppe auf Lese– und Schreibrecht:
chmod g+rw hugo.txt $\Rightarrow$

```
-rw-rw----  1 jewe          1406 Oct  6 14:26 hugo.txt
```

Hinzufügen des Leserechts für alle:
chmod o+r hugo.txt $\Rightarrow$

```
-rw-rw-r--  1 jewe          1406 Oct  6 14:26 hugo.txt
```

Es gibt noch eine weitere Möglichkeit die Zugriffsrechte anzugeben:

```
chmod mode file
```

`mode` ist dabei eine dreistellige Oktalzahl. Sie stellt eine Kodierung der Zugriffsrechte dar, wobei die erste Stelle die Rechte für den Benutzer angibt, die zweite für die Gruppe und die dritte für alle anderen.

Beispiel:
chmod 660 hugo.txt $\iff$ **chmod g+rw hugo.txt**

Der Unix–Befehl 'find'

Syntax von `find`

`find [-l`n`] verzeichnis(se) ausdruck`

Der Befehl `find` durchsucht die, im Parameter `verzeichnis(se)` angegebenen Dateibäume, nach Dateien, die den in `ausdruck` angegebenen Kriterien entsprechen. Mit der Option `-l`n kann man die Suchtiefe im Dateibaum auf 'n' beschränken.

Beispiele:

- `find /home/FM/hugo -name brief.txt -print`
  Durchsucht den Dateibaum `/home/FM/hugo` nach der Datei `brief.txt` und gibt alle Unterverzeichnisse, in der sich eine solche Datei befindet aus.
- `find . -name "*.c" -exec ls -l {} \;`
  Durchsucht ab dem aktuellen Verzeichnis alle Unterverzeichnisse nach Dateien mit der Endung `.c` und listet sie im 'long format' auf.

Die Unix–Befehle 'cp' und 'mv'

Syntax von `cp` und `mv` (analog zu copy und move)

`cp datei_1 datei_2` | `cp datei_1 {datei_2 ...} verzeichnis`

`cp` kopiert Dateien, d. h. es wird eine neue Datei erzeugt.
`mv` bewegt Dateien, d. h. es wird ein neuer Verzeichniseintrag eingerichtet und der alte gelöscht.

Beispiel:

```
cp /home1/FM/hugo/test.dat /home1/FM/hugo/tmp
mv brife.txt briefe.txt
```

Die Unix–Befehle 'more' und 'less'

Beide Befehle zeigen den Inhalt einer Datei seitenweise am Bildschirm an. (. . . und sind beide für ihre Aufgabe 'more or less' geeignet.)

Syntax von `more`

```
more [-optionen] datei
```

Wenn `more` die erste Seite der Datei angezeigt hat, kann der Benutzer die Ausgabe der nächsten Information steuern:

Eingabe:	Wirkung
< leertaste >	⇒ Ausgabe der nächsten Seite
n < leertaste >	⇒ Ausgabe der nächsten n Zeilen
< CR >	⇒ Ausgabe einer weiteren Zeile
/	⇒ Suche nach Text
h	⇒ Zeigt alle Kommandos für 'more' an
q	⇒ Beende 'more'

Syntax von `less`

```
less [-optionen] datei
```

Die beiden Programm `less` und `more` sind in ihrer Funktionsweise sehr ähnlich; `less` bietet jedoch einige zusätzliche Befehle, z. B.:

Eingabe:	Wirkung
b	⇒ Blättert eine Seite zurück
{	⇒ Falls sich in der ersten Zeile, die auf dem Bildschirm zu sehen ist eine '{' Klammer befindet, sucht dieses Kommando die korrespondierende schließende Klammer und blättert zu dieser Seite.
}, (,)	⇒ analog

Der Unix-Befehl 'rm'

Syntax von `rm`

```
rm [-optionen] datei ...
```

`rm` (remove) löscht die angegebenen Dateien (analog zu `del` unter MS–DOS). Die wichtigsten Optionen für `rm` sind:

- **-i** (**i**nteractive) Vor dem Löschen jeder Datei wird abgefragt, ob die Datei wirklich gelöscht werden soll. Die Buchstaben `y` oder `Y` löschen die Datei, bei allen anderen Eingaben bleibt sie erhalten.
- **-f** (**f**orce) Löscht die angegebene Datei ohne Sicherheitsabfrage, auch wenn sie schreibgeschützt ist (kein **w**– Recht gesetzt ist).
- **-r** (**r**ecursive) Löscht auch Unterverzeichnisse, wobei rekursiv alle Dateien in diesen Unterverzeichnissen gelöscht werden.

Wichtig:

- Verzeichnisse können nur mit `rmdir` (remove directory) gelöscht werden, wenn sie leer sind, oder aber mit "`rm -r ...`".
- Bei schreibgeschützten Dateien, wird vor dem Löschen noch einmal nachgefragt:
  `rm: override protection 444 for test.dat?`

Der Befehl '`rm -rf` $*$' löscht ohne Rückfragen alle Dateien und Verzeichnisse ab dem aktuellen Verzeichnis.

Der Unix–Befehl 'grep'

Die Programme 'grep', 'fgrep' und 'egrep' erlauben, Dateien schnell auf bestimmte Textmuster zu durchsuchen und die entsprechenden Zeilen auszugeben.

Syntax von 'grep'

```
grep [-optionen] ausdruck [dateien]
```

(**grep** steht für **g**et **r**egular **e**xpression **p**attern)

Beispiele:

- `grep int *.c`
  Durchsucht alle Dateien mit der Endung `.c` auf das Textmuster `int` und gibt die entsprechenden Zeilen aus.
- `grep '*nf' woerterbuch`
  Durchsucht die Datei `woerterbuch` nach allen Wörtern, die auf `nf` enden.

Der Unix–Befehl 'wc'

Der Befehl `wc [datei]` zählt alle Zeilen, Wörter und Zeichen in einer Datei und gibt die Anzahl aus.

Syntax von 'wc'

```
wc [-optionen] [dateien]
```

Falls keine Datei angegeben wird, liest das Programm solange von der Standardeingabe (Tastatur), bis ein EOF (end of file) Zeichen eingegeben wird (`ctrl-d`).

Manual Pages

Es ist nicht anzunehmen, daß irgendein Mensch sich alle Optionen für alle Unix–Befehle merken kann oder will. Daher stellt Unix *online manual pages* zur Verfügung.

Beispiel: `man ls`

Dieses Kommando gibt die *manual pages* für den `ls`–Befehl am Bildschirm aus.

Aufbau der Manual Pages

Das Manual ist in 8 Kapitel mit folgenden Inhalten unterteilt:

- **User Commands**: Alle Benutzerkommandos, die von einer Shell aus verwendet werden können (Utilities).
- **System Calls**: Systemaufrufe (Schnittstellen zum Betriebssystem)
- **C Library Functions**: C–Bibliotheksroutinen
- **Devices and Protocols**: Beschreibung der Gerätetreiber und Geräte–Charakteristika.
- **File Formats**: Beschreibung aller Dateiformate, die in Unix benutzt werden.
- **Games**: :-)
- **Special**: Alles was sich keinem anderen Kapitel zuordnen läßt.
- **Maintainance Commands**: Systemverwaltung

Viele nützliche Hilfen für den Umgang mit den Manual Pages erhält man über den Aufruf `man man`.

Syntax des 'man' Befehls

```
man [kapitel] titel
```

Die Angabe des Kapitels ist optional und nur dann sinnvoll, wenn es den '`titel`' in mehreren Kapiteln gibt. Mit der Option **-k** kann nach einem Schlüsselwort in den Manual Pages gesucht werden.

```
man -k keyword
```

Einen guten Überblick über die einzelnen Kapitel erhält man durch:

```
man kapitel intro
```

Jeder Eintrag in den Manual Pages ist wieder in mehrere Abschnitte unterteilt:

- **Name**: Name des Kommandos mit einer knappen Funktionsangabe.
- **Synopsis**: Syntax des Kommandos.
- **Description**: Detaillierte Beschreibung des Kommandos
- **Files**: Dateien, die von dem Kommando verwendet werden.
- **See also**: Verweise auf Kommandos mit ähnlicher oder ergänzender Funktion.
- **Diagnostics**: Kurze Erklärung der möglichen Fehlermeldungen.
- **Bugs**: Hier werden bekannte Fehler, Inkonsistenzen oder Probleme angegeben.
- **Example:** Beispiele für die Verwendung des Kommandos

Die Shell – ein Kommandointerpreter

Nach dem Einloggen wird eine Shell gestartet. Sie verhält sich ähnlich, wie der Kommandointerpreter unter MS–DOS. Die ursprüngliche Unix–Shell hieß einfach *sh*. Inzwischen sind viele unterschiedliche Shells entstanden, die den Bedienkomfort erheblich verbessert haben. Wir werden in den Übungen die *tcsh* verwenden (weitere Beispiele sind csh, bsh, bash, ksh).

Aufgaben des Kommandointerpreters unter MS–DOS

Der Kommandointerpreter von MS–DOS heißt `command.com`.

- Liefert das Prompt (z. B. C:\).
- Interpretiert Befehle (z. B. `dir`).
- Ruft Programme auf (z. B. `turbo`).

Aufgaben der Shell unter Unix

- Liefert ein Prompt (z. B. %) $\Rightarrow$ Wartet auf Eingaben
- Interpretiert Befehle: Die Shell unterscheidet zwei Arten von Kommandos, sogenannte interne Kommandos (builtin commands) und externe Kommandos.
  - interne Kommandos: Kommandos, die von der Shell selbst ausgeführt werden, d. h. im Sourcecode der Shell implementiert sind (z. B. : `cd, set, pwd`).
  - externe Kommandos: (Programme) Für diese Kommandos erzeugt die Shell einen eigenen Prozeß, der das entsprechende Programm ausführt.

Interne Kommandos der Shell

set und setenv

- Das Kommando `set [var[=value]]` weist der Shellvariablen `var` den Wert `value` zu.
- `set` ohne Parameter listet alle Shellvariablen mit ihren aktuellen Werten auf.
- Beispiele für Shellvariablen mit spezieller Bedeutung:
  - **user** — enthält den login–Namen (user–Name)
  - **path** — enthält den Suchpfad für externe Kommandos
  - **term** — enthält den Typ des Terminals an dem der Benutzer gerade arbeitet
  - **home** — enthält das Home–Verzeichnis des Benutzers

  Diese Shellvariablen sind automatisch auch Environmentvariablen.
- Auf Environmentvariablen kann jedes Programm, das von der Shell aus gestartet wurde zugreifen. Auf die einfachen Shellvariablen kann nur die Shell selbst zugreifen.
- Das Kommando `setenv [VAR[=word]]` weist der Environmentvariablen `VAR` den Wert `word` zu, wobei `word` ein einzelnes Wort oder eine, durch einfache Anführungszeichen zusammengefaßte Folge von Wörtern ist.

MS–DOS kennt auch einige spezielle Variablen, z. B. `PROMPT` und `PATH`. Sie haben die gleiche Bedeutung wie unter Unix.

change directory (`cd`)

- Fast analog zum cd–Kommando unter MS–DOS zu verwenden.
- Ausnahme: `cd` ohne Parameter ⇒ Benutzer wechselt in sein `home`–Verzeichnis.
- '`cd /`' wechselt ins `root`–Verzeichnis.
- '`cd ..`' wechselt in das nächst höhere Verzeichnis.

Zu beachten:

> In Unix wird als Verzeichnisseperator das Zeichen '/' anstelle von '\' verwendet.

print working directoy (`pwd`)

Das `pwd`–Kommando gibt das aktuelle Verzeichnis aus.

alias–Substitutionen

Mit Hilfe des `alias`–Kommandos kann der Benutzer Abkürzungen für lange oder häufig verwendete Befehle definieren.

```
alias abkürzung kommando_text
```

Beispiele für Abkürzungen:

- `alias ll 'ls -l'`
- `alias dir 'ls -l'` (für alle MS–DOS Freaks)
- `alias del rm`

Beim Starten der `tcsh` wird die Datei `.cshrc` automatisch gelesen und die Befehle darin ausgeführt. Alias Kommandos, die man in die Datei `.cshrc` eingeträgt werden demnach bei jedem Einloggen ausgeführt.

Das Unix–Prozeßkonzept

Unix muß als ein Multitasking–BS eine Möglichkeit zum Erzeugen, Verwalten und Beenden von Prozessen zur Verfügung stellen.

Was ist ein Prozeß ?

Man kann einen Prozeß vereinfacht als ein Programm + Programmumgebung betrachten. Zur Programmumgebung gehören u. a. Speicherbelegung, Registerinhalte, das aktuelle Verzeichnis und die für den Prozeß sichtbaren Umgebungsvariablen (*environment*), z. B. $HOME, $PATH.

Ein Prozeß kann sich in drei Grundzuständen befinden:

- **running**, d. h. der Prozeß ist gerade aktiv,
- **suspended**, er ist rechenwillig, besitzt aber die CPU nicht,
- **waiting**, er wartet auf ein Ereignis.

Verwaltung von Prozessen

Da in einem Unix–System mehrere Prozesse um die Zuteilung der CPU konkurrieren, muß im System eine Steuerung dafür existieren.

- Die Steuerung wird *Scheduling–Algorithmus* genannt,
- die Terminierung bzw. zeitweise Verdrängung *Suspendierung.*
- Die Aktivierung des nächsten rechenbereiten Prozesses *Scheduling.*

Beenden von Prozessen

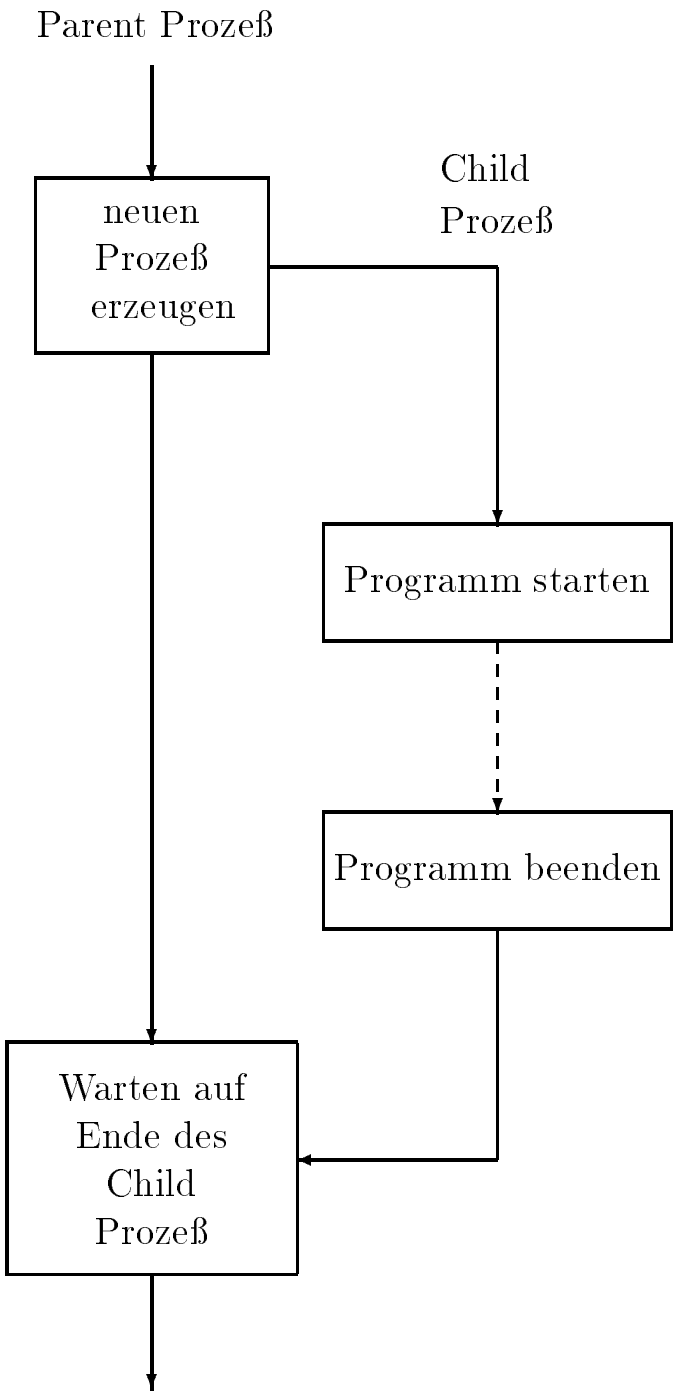
Eine Prozeß kann beendet werden, wenn

- das zugehörige Programm auf ein *exit*–Statement trifft,
- der Benutzer über ein Shellkommando (`kill`) ein Signal an den Prozeß sendet (bzw. wenn er ein Signal empfängt).

Erzeugen von Prozessen

Wenn ein UNIX–System neu gestartet wird existiert nur ein einziger Prozeß, alle anderen Prozesse werden nach folgendem Muster erzeugt:

- Jeder Prozeß (`parent`) kann durch einen Systemaufruf einen neuen Prozeß (`child`) erzeugen.
- Der `child`–Prozeß ist eine fast identische Kopie des `parent`–Prozesses. Sie unterscheiden sich nur durch die PID (**P**rocess **ID**entifier).
- Durch einen weiteren Systemaufruf (*exec(ute)*) lädt und startet der `child`–Prozeß ein Programm.
- Soll der `child`–Prozeß nach Ausführen des Programms beendet werden, geschieht das durch den Systemaufruf *exit()*.
- Der `parent`–Prozeß kann währenddessen die Ausführung seines Programmes fortsetzen oder auf die Terminierung des `child`–Prozesses warten (*wait()*).



Zurück zur Shell

Externe Shell–Kommandos

Um ein externes Kommando (Programm) auszuführen, erzeugt die Shell einen neuen Prozeß. Der neue Prozeß startet dann das auszuführende Programm.

Beispiel: Editieren der Datei hugo.cc `% vi hugo.cc`

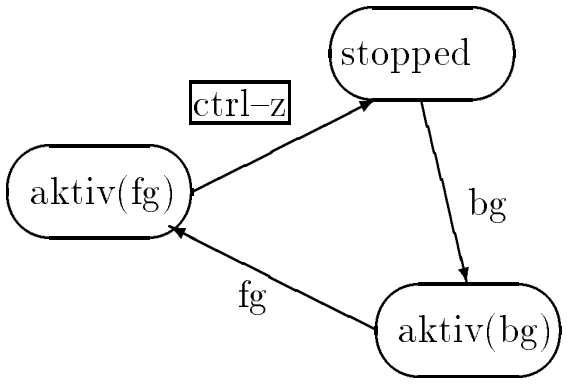
Hintergrund–Prozesse

- Der Benutzer kann bestimmen, ob ein Prozeß im *Hintergrund* oder *Vordergrund* laufen soll.
- Einen Prozeß im *Hintergrund* starten bedeutet, daß die Shell weiter aktiv bleibt und dem Benutzer ein Prompt für weitere Eingaben bietet. Dazu muß der Benutzer ein '&' Zeichen an den Befehl anhängen.
- Das '&' Zeichen signalisiert der Shell, daß dieser Befehl im Hintergrund (*background*) ausgeführt werden soll.

Beispiel: Compilieren einer Datei `% g++ hugo.cc &`

Jobkontrolle durch die Shell

- Prozeß aktiv im Vordergrund
- Tastenkombination ctrl–z
  ⇒ Prozeß stoppen
- Mit `bg` den Prozeß „in den Hintergrund legen“
- Mit `fg` den Prozeß „in den Vordergrund holen“



Ein–/Ausgabeumlenkung

Viele Unix–Kommandos arbeiten mit drei Standarddateien:

- Sie lesen von der *Standardeingabe* (meistens die Tastatur),
- und schreiben auf die *Standardausgabe* (den Bildschirm).
- Falls Fehler auftreten werden sie auf die *Standardfehlerausgabe* geschrieben (im Normalfall ebenfalls der Bildschirm).

Die Ein–/Ausgabe kann vom Benutzer der Shell 'umgelenkt' werden. Dazu stehen einige Sonderzeichen zur Verfügung:

Sonderzeichen	Bedeutung
<	Umlenken der Standardeingabe
>	Umlenken der Standardausgabe
> &	Umlenken der Standardfehlerausgabe
|	Umlenken der Standardausgabe zum nächsten Befehl

Beispiele:

- Ausgabeumlenkung:

  `ls -l / > ausgabe.txt`
  Die Ausgabe des `ls`–Kommandos wird in die Datei `ausgabe.txt` umgelenkt.

- Eingabeumlenkung:

  `wc < ausgabe.txt`
  `wc` nimmt als Eingabe, den Inhalt der Datei `ausgabe.txt`.

- Ausgabeumlenkung (mit 'anhängen'):

  `ls >> ausgabe.txt`
  Falls die Datei `ausgabe.txt` schon existiert, wird der Inhalt des aktuellen Verzeichnisses an die Datei angehängt.

(Auch unter MS–DOS ist Ein–/Ausgabeumlenkung möglich, z. B. `more < hugo.cc` oder `type hugo.txt > muell.txt`)

Pipelinemechanismus

Wenn man die Ausgabe eines Programmes als Eingabe für ein anderes Programm benutzen möchte, gibt es zwei Möglichkeiten:

- Entweder speichert man die Ausgabe des ersten Programmes in einer Pufferdatei und benutzt diese dann als Eingabe für das zweite Programm:

  ```
  ls -l > inhaltsverzeichnis.txt
  wc < inhaltsverzeichnis.txt
  ```

- oder man benutzt das sogenannte Pipe–Zeichen '`|`':

  ```
  ls -l | wc
  ```

  Die letzte Methode erspart das Anlegen einer Datei auf der Festplatte und ist daher schneller.

  Beispiel: Wieviele Zeilen in allen TeX–Dateien des aktuellen Verzeichnisses enthalten das Wort Unix ?

  ```
  grep Unix *.tex |wc -l
  ```

Prozeßorientierte Befehle

Das `ps`–Kommando

ps steht für print **p**rocess **s**tatus. Das Kommando liefert Informationen über den Status von Prozessen. Die Art und Menge der Informationen wird über die Optionen gesteuert.

Die Syntax des `ps`–Kommandos:

```
ps -optionen
```

- **ps** : Zeigt Informationen über die eigenen Prozesse an:
  - Prozeßnummer (PID)
  - Name der Dialogstation (TT)
  - Status des Prozesses (STAT)
  - verbrauchte Rechenzeit (TIME)
  - expandierter Kommandoaufruf (COMMAND)
- **ps -a** : Informationen über alle Prozesse anzeigen.
- **ps -au** : Zeigt zusätzliche, benutzerorientierte Informationen über alle Prozesse an, z. B.:
  - Benutzer (USER)
  - reine CPU–Zeit in % (% CPU)
  - Benutzter Speicher in % (% MEM)
  - Größe des Prozesses zu 512 Byte Blöcken (SZ)
  - Startzeit (START)
- **ps -aux** : Zeigt zusätzlich alle Prozesse an, die keine Dialogstation haben an (z. B. lpd, nfsd . . .).

Das `jobs`–Kommando

Das `jobs`–Kommando zeigt die aktiven Hintergrundprozesse (Programme die mit `command &` gestartet wurden oder mit `ctrl-z` und '`bg`' in den Hintergrund geschickt wurden.) an. Die Syntax des `jobs`–Kommandos:

```
jobs [-l]
```

- `jobs` zeigt die Jobnummern & Kommandos.
- `jobs -l` zeigt zusätzlich die PID an.

Das `kill`–Kommando

Das `kill`–Kommando kann ein beliebiges *Signal* an einen Prozeß senden.
Die Syntax des `kill`–Kommandos:

```
kill [-signal] pid ...
```

- Die PID bezeichnet den Prozeß, an den das Signal gesendet werden soll. Falls es sich um einen Hintergrundprozeß handelt, kann die 'Jobnummer' (% n) an Stelle der PID angegeben werden.
- Falls kein Signal angegeben ist, wird Signalnummer 15 angenommen, wobei Signal 15 bedeutet: Prozeß beenden (daher der Name `kill`).
- Falls eine Signalnummer angegeben ist, wird das entsprechende Signal an den Prozeß gesendet.
- Mit Signalnummer 9 (`kill -9 pid`) kann man jeden (eigenen) Prozeß abbrechen.

Das Mailsystem

Zum Austausch von Nachrichten zwischen den Benutzern steht ein Mailsystem (*email*, electronic mail) zur Verfügung. Man kann es mit dem Senden von Briefen über die Post vergleichen.

Syntax des 'mail'–Kommandos

```
mail -options name ...
```

Dem `mail`–Kommando wird als Zieladresse der *user*–Name des Empfängers übergeben. Wenn eine *email* ankommt, wird sie in einer Datei (*mailbox*) gespeichert. Der Empfänger wird bei Ankunft einer Nachricht informiert. Das Kommando dient nicht nur zum Versenden, sondern auch zum Lesen der eigenen *email*.

Beispiel zum Versenden einer *email* :

```
% mail hugo@gipsy9.cs.uni-sb.de
Hallo Hugo!

Gehst Du morgen in Uni--Film?

Gruß Emil

.
CC:
```

- `hugo` bezeichnet den Empfänger,
- `gipsy9.cs.uni-sb.de` den Zielrechner.
- Der einzelne Punkt '.' in der letzten Reihe markiert das Textende und veranlaßt, das Versenden der *email* .
- Das System fragt mit '`CC:`' nach, ob diese *email* noch an andere Benutzer geschickt werden soll (`CC` steht für *carbon copy*).

Bei größeren Briefen ist es sinnvoll den Text in einem Editor (z. B. vi) zu schreiben. Diesen Editor kann man aus dem Mailprogramm heraus

mit `~v` aufrufen. Eine Übersicht über alle Tilde–Kommandos erhält man mit `~?`.

Beispiel zum Lesen der *mail* :

```
You have mail.
% mail

Mail version SMI 4.1-OWV3 Mon Sep 23 07:17:24 PDT
"/usr/spool/mail/jewe": 1 message 1 new
>N  1 simmet         Fri Oct  8 12:56   11/212   test
&
```

Das Mailkommando ist ein interaktives Programm und besitzt ein eigenes Prompt '&'.
Das Mailkommando akzeptiert in diesem Zustand folgende Befehle:

- `p [number]`: Zeige Mail mit Nummer `number` am Bildschirm an.
- `d [number]`: Lösche Mail.
- `s`: Speichere Mail.
- `r`: Antworte auf die Mail (Die Antwort wird automatisch an den Absender der Mail geschickt.).
- `?`: Gibt eine kurze Übersicht, über die möglichen Kommandos mit Erklärung aus.