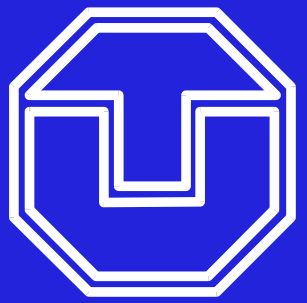
Fallbeispiel Unix

Betriebssysteme
WS 15/16

Hermann Härtig
TU Dresden



Lernziel

- grobes Kennenlernen einer konkreten Schnittstelle für die Bausteine (s. Einführung)

Wegweiser

Geschichte und Struktur von Unix

Vom Programm zum Prozess

Unix-Grundkonzepte
- Dateien
- Prozesse

Prozess-Kommunikation
- Signale
- Pipes
- Sockets

Rechte und Schutz

Unix Story

196x	MULTICS (MIT)	wichtige Ideen, „Fehlschlag“ ?
1971	Ken Thompson	„UNICS" auf PDP-7 (First “Edition”)
1973	Dennis Ritchie + KT	C, rewrite in C
1974	TR74	The Unix Time-Sharing System
1975		Sixth “Edition”, weite Verbreitung
1977	Richards	Portierung auf Interdata (32 Bit)
1979		Bourne-Shell, PCC
1980	Bill Joy, et. al.	Berkeley SD 4, „vi“
198x		virtueller Speicher, Netzwerke
1982	Randell et al.	Newcastle Connection

Dateien in Unix

Pfadnamen …

Kernschnittstelle:

- fd = open(name, flags, mode)
- bytes = read(fd, but, size)
- bytes = write(fd, bud, size)
- lseek(fd, position)

Unix-Objekte haben Schnittstelle analog zu Dateien
fd file descriptor 0..n

Prozesse in Unix

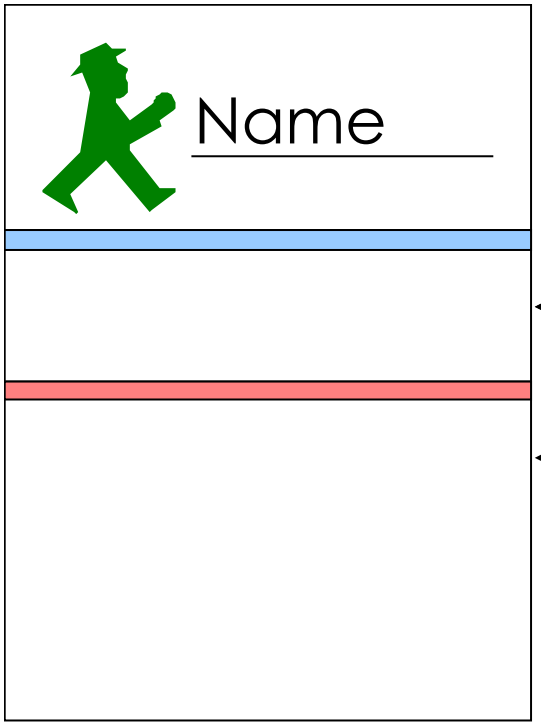
Unix-Prozess

- ein Programm
- ein Thread
- ein Adressraum ...
- „is a program in execution“
- „Besitzer" aller Betriebsmittel (Speicher, Dateien, ...)
- repräsentiert *Prinzipale* (durch UId/GId)

Viele Prozesse pro Rechner

- Benutzerprozesse
- Hintergrund-Systemprozesse („daemons“)

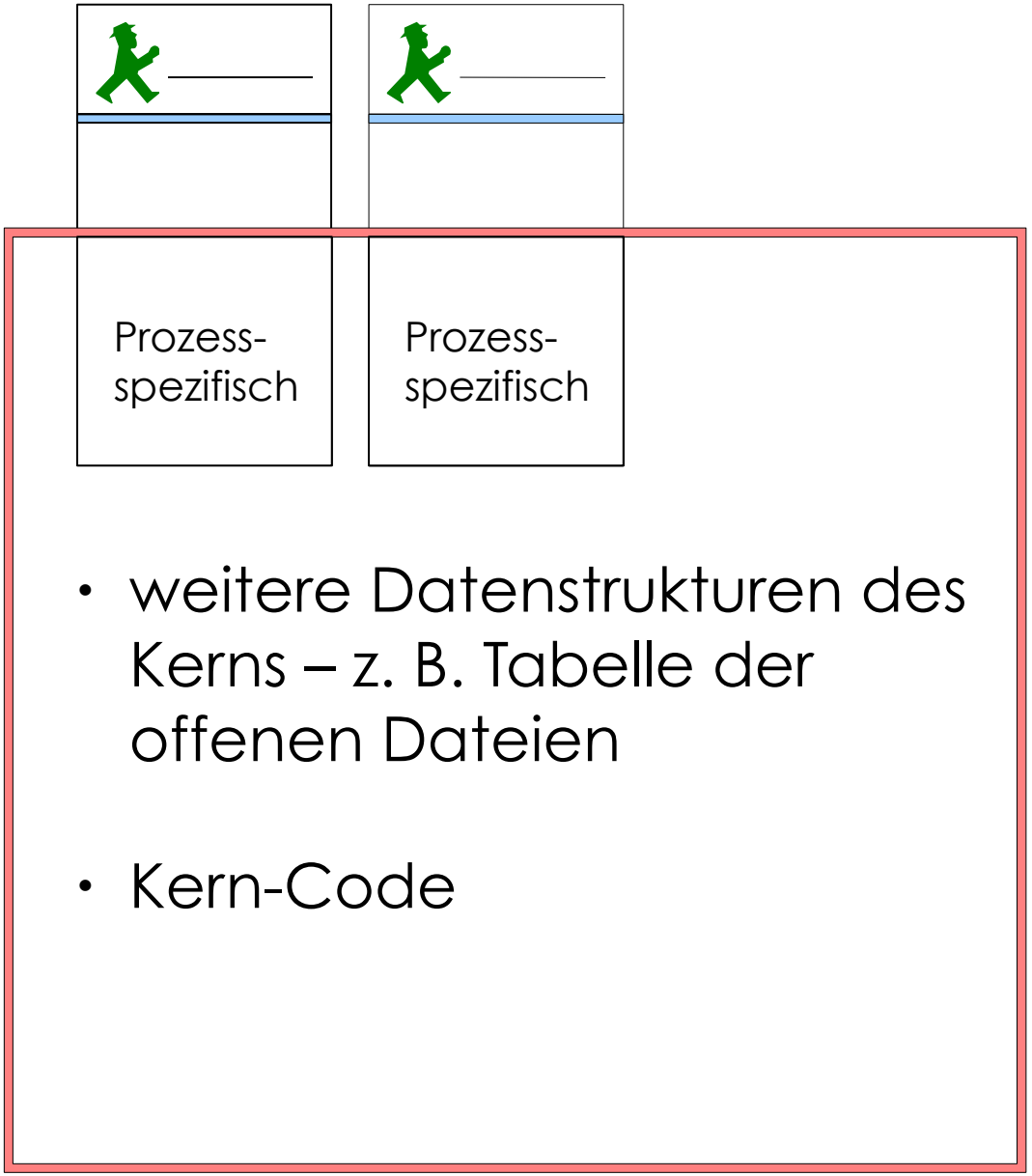
Darstellung Unix-Prozesse im Folgenden



User-Mode Daten

dem Prozess zugeordnete Kern-Strukturen werden nur im Kern manipuliert

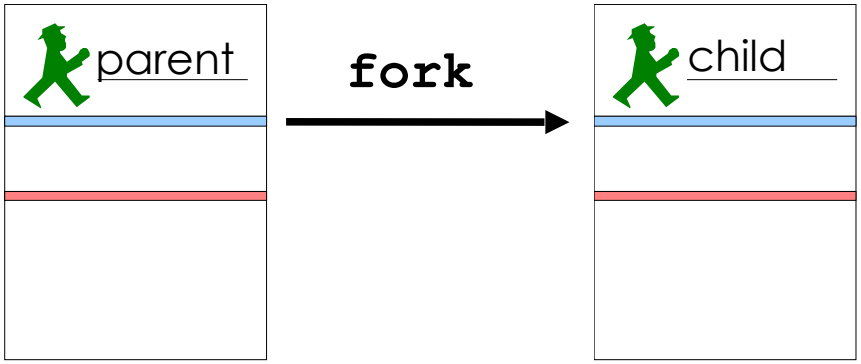
Kern-Adressraum



- weitere Datenstrukturen des Kerns – z. B. Tabelle der offenen Dateien
- Kern-Code

Kernaufruf: Erzeugung von Prozessen

```
X = fork();
//Erstellen einer exakten Kopie des Aufrufers
//inklusive Adressraum, aller Dateideskriptoren ...
```




```
if (X == 0) {
  //child code
} else {
  // parent code, X==pid of child
  printf(„new child: PID = %d\n", X);
}
```

Weitere Kernaufrufe

`s = exec(file, argument, environment)`

- ersetzt Speicherinhalt durch Inhalt von `file` und führt `file` aus
- schreibt Argumente und Umgebungsvariablen an Anfang des Kellersegments

`exit(status)`

- existiert noch (als „Zombie“), bis Eltern-Prozess `wait` ausführt
- überträgt Ergebnis zum Eltern-Prozess

`s = waitpid(pid, status, block or run)`

- wartet auf Ende des Kindprozesses `pid`
  bei `pid` = -1 auf irgendein Kind
- Ergebnis des Kindprozesses in `status`

Beispiel: Shell mittels fork/exec

```
read (command, params);

X = fork();
// erzeugt Kopie des Aufrufers (d.h. der Shell)
// Kind erhält fd des Eltern-Prozesses
// beide Prozesse setzen Abarbeitung hinter fork fort

if (X < 0){
   // Fehlerbehandlung
} else if (X != 0) {
   // Parent-Prozess
   waitpid(X, &status, 0); // warte auf Kind-Prozess
} else {
   // Child
   exec(command, params, env);
}
```

Beispiel: Shell mittels fork/exec

```
read (command, params);

X = fork();




if (X < 0){

} else if (X != 0) {
  // Parent
  waitpid(X, &status, 0);
} else {
  // Child
  exec(command,params,env);
}
```

```
read (command, params);

X = fork();




if (X < 0){

} else if (X != 0) {
  // Parent
  waitpid(X, &status, 0);
} else {
  // Child
  exec(command,params,env);
}
```

Beispiel: Shell mittels fork/exec

```
read (command, params);

X = fork();




if (X < 0){

} else if (X != 0) {
   // Parent
   waitpid(X, &status, 0);
} else {
   // Child
   exec(command,params,env);
}
```

```
// Program-Code for command

.......




exit
```

Threads (neuere Unix Versionen)

- Bibliotheksfunktionen (z.B. „pthreads“)
- Linux syscall; clone(….)

```
main()
{
  p1 = pthread_create(thread_function);
  p2 = pthread_create(thread_function);

  // do something else

  pthread_join(p1);
  pthread_join(p2);
}
```

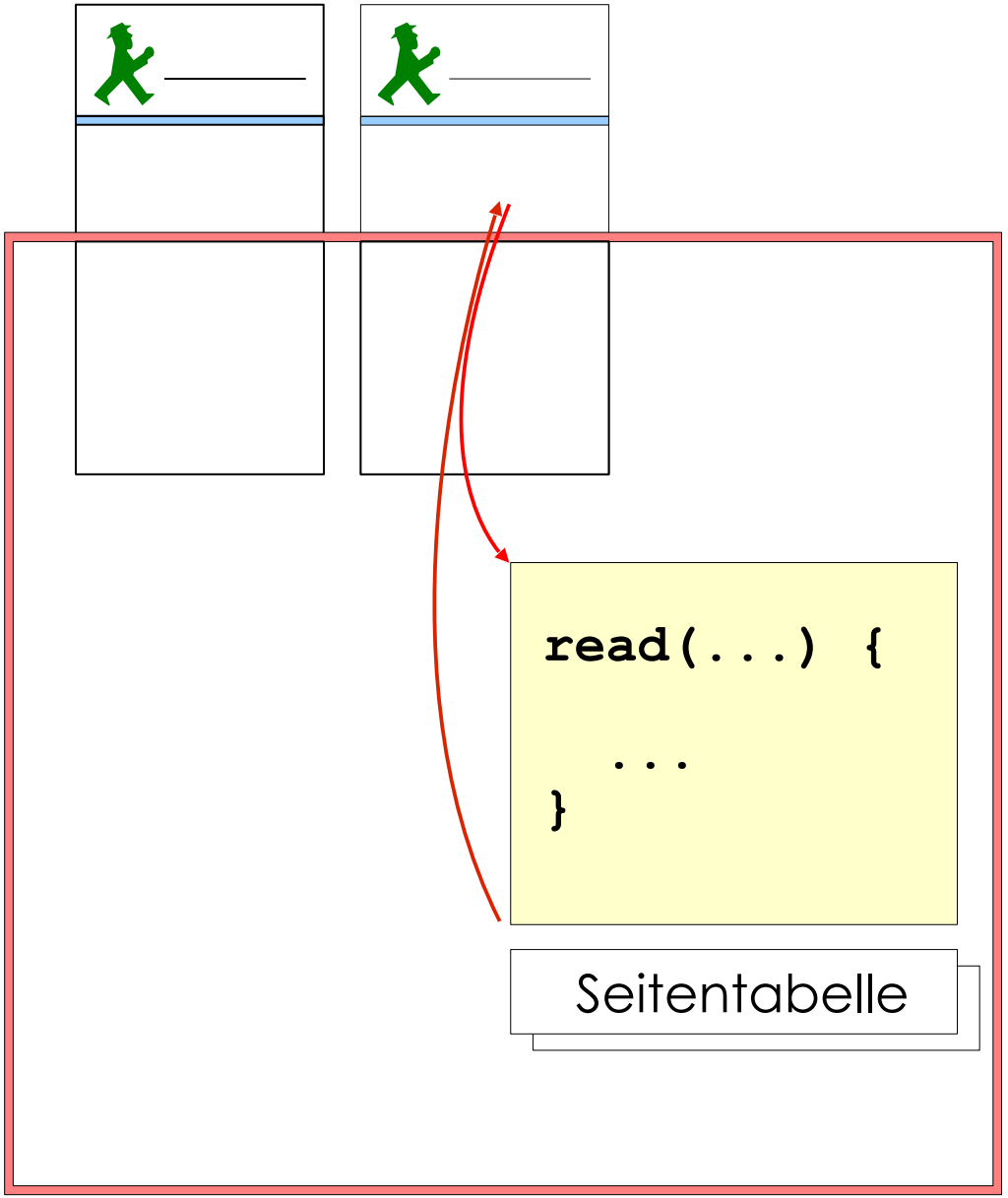
Kernaufrufe (System-Calls)

Beispiel

```
status = read (10, buf, anzahl);
```

- Ergebnis: Anzahl der gelesenen Bytes
- Konvention: -1 → Fehler
- Meldung der Fehlerursache: **`errno`**

Schutz des Kerns?



- Wie wird der Kern sicher aufgerufen?
- Wie werden Kern-Strukturen geschützt? (Beispiele später)

Prozessor-Modi: usermode/kernelmode

Kernelmode	Usermode
Alle Instruktionen	Teilmenge
User- und Kernel-Mode Speicher	User-Mode Teil des Adressraums
Umschalten des Adressraums	

Kernaufruf im Detail

Benutzerprozess

```
read(...) {

   //Parameteraufbereitung
    ...
   call = read;
   INT 0X80 //trap (alt)
```

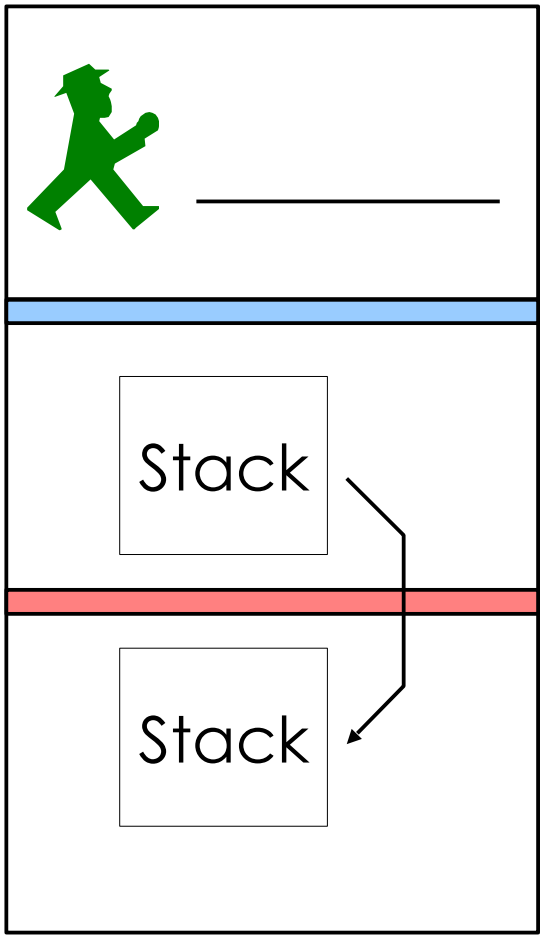
```
   //weiter geht's
}
```

Kern

```
//TRAP-Entry
 switch (call) {
  case read:
    ...
    Iret //return from trap
  case write: ...
```

- User-Mode: kein Zugriff auf Kern-Adressraum
- Kernel-Mode: Zugriff auf Kern- und Benutzer-Adressraum

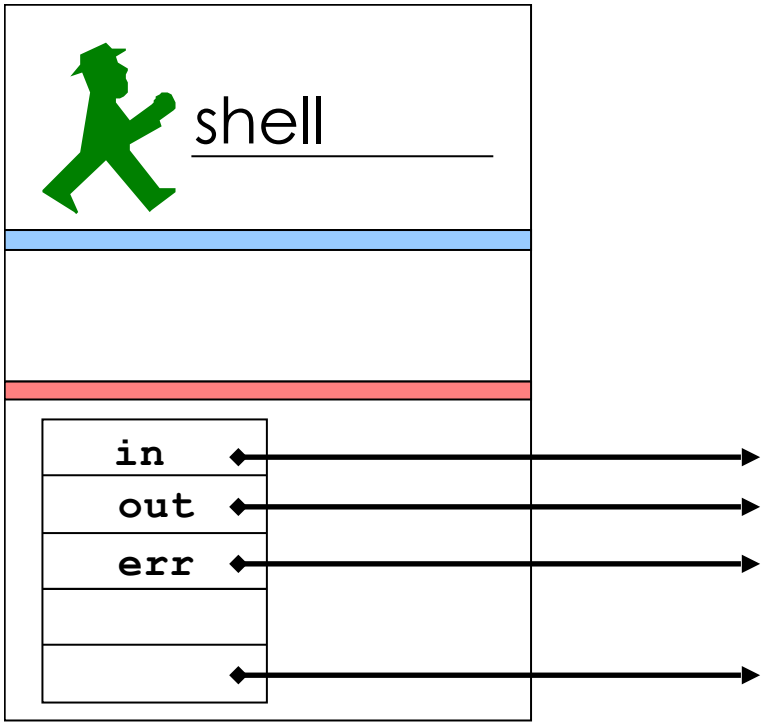
Zwei Keller pro Prozess: User, Kernel



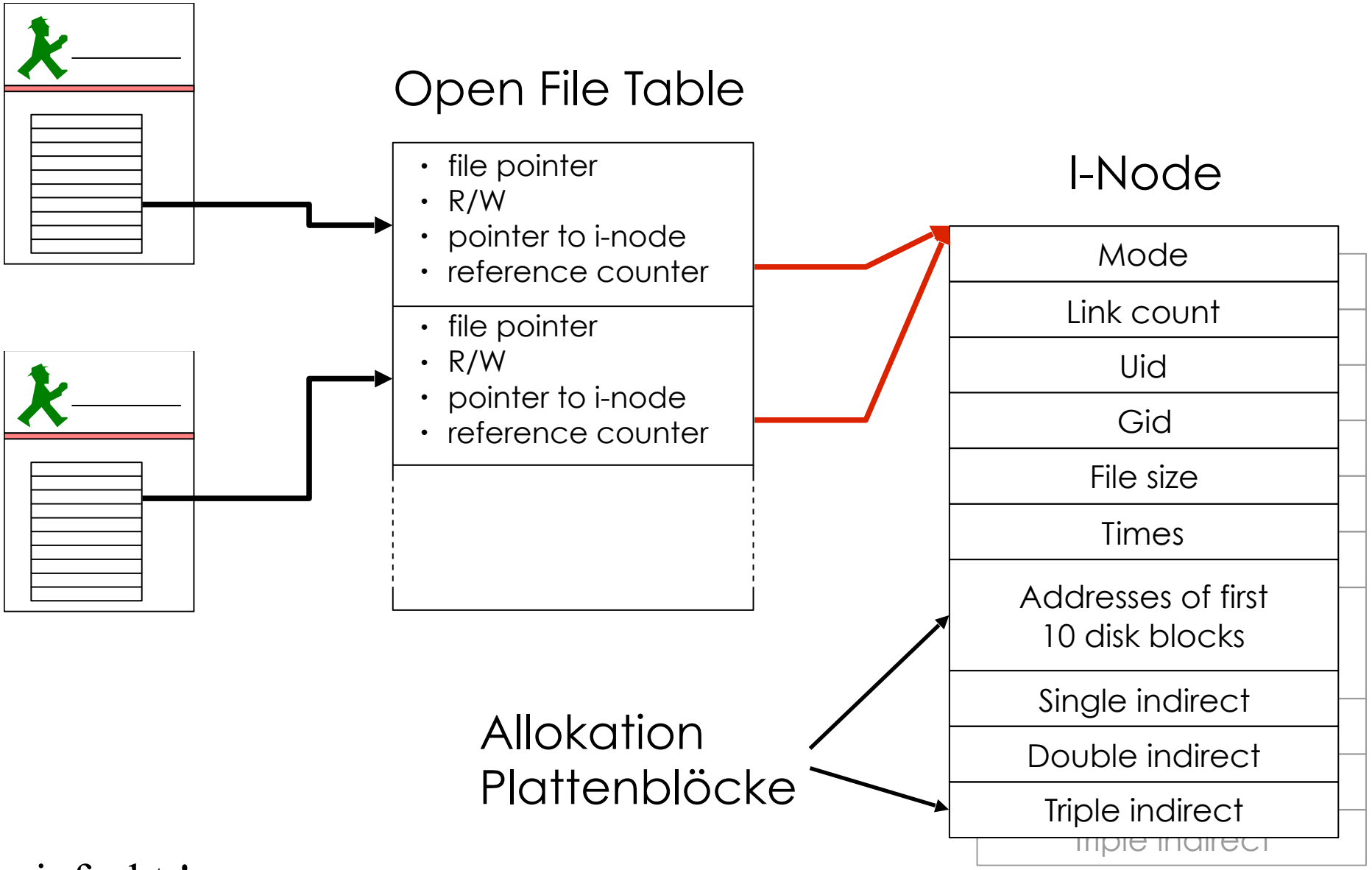
Bei Systemcalls wird

- auf den Kern-Keller geschaltet
- der Kernmodus eingeschaltet
  dadurch wird der Kern-Adressraum sichtbar
- an eine feste Einsprungstelle gesprungen und von dort kontrolliert verzweigt

Kernschnittstelle: Datei-Deskriptoren

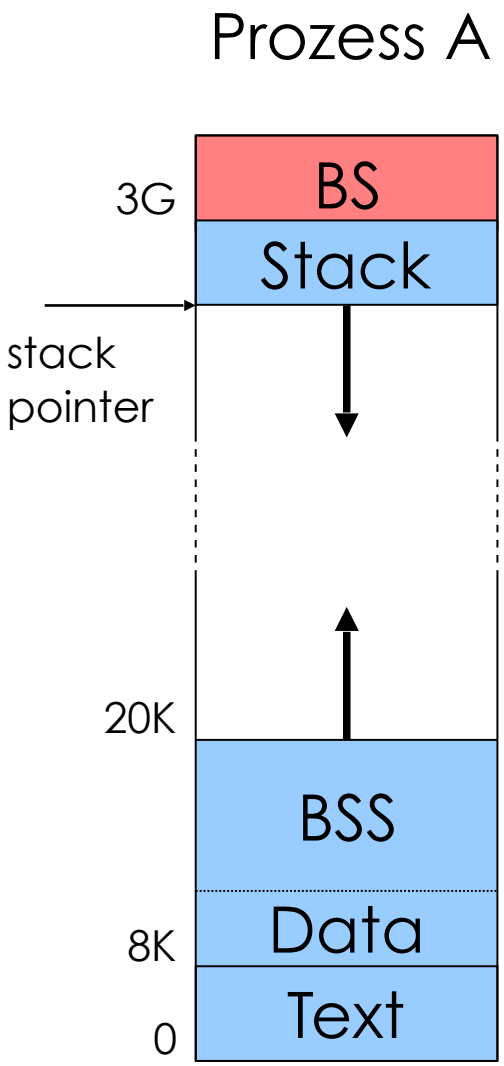


Kerninterne Datenstrukturen



vereinfacht !

Das Unix-Speichermodell



Daten-Segment

- globale Daten eines Programms
  - **Data**: initialisierte Daten
  - **BSS**: per Konvention mit 0 initialisiert
    erweiterbar durch Systemaufruf `brk`

Textsegment

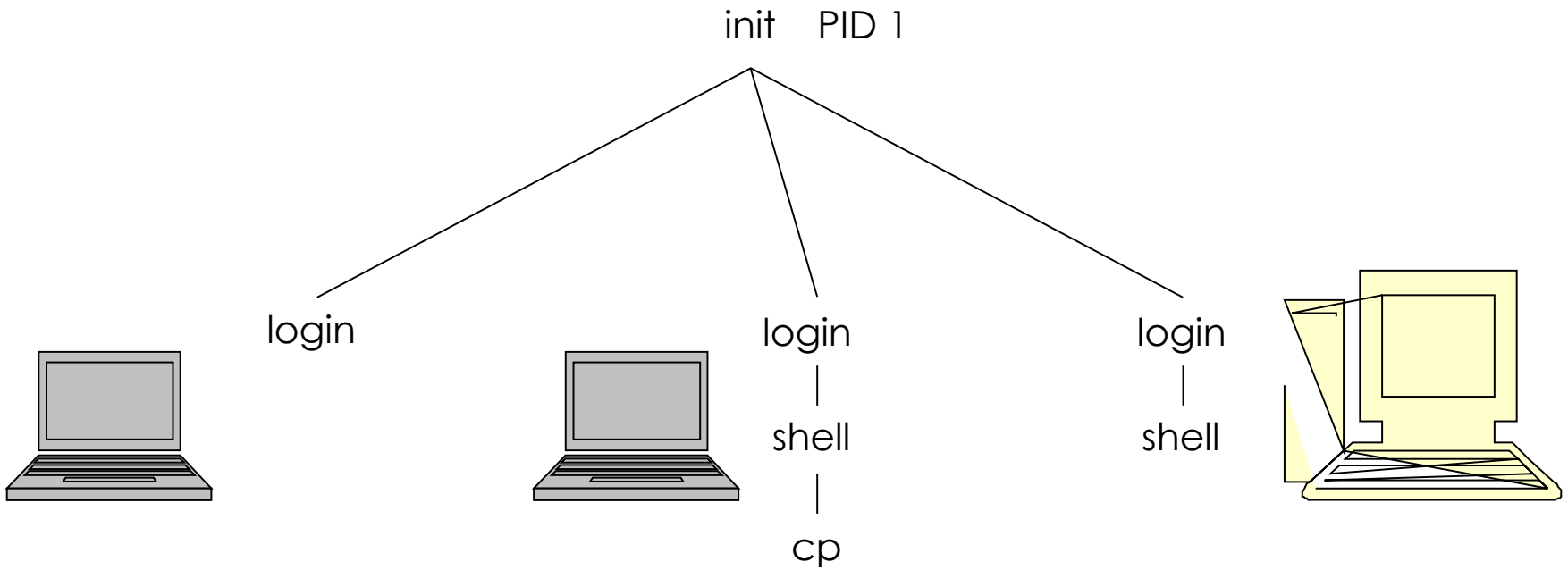
- enthält Maschinencode
- read only
- erste Seite frei zum Entdecken nicht-initialisierter Pointer
- "shared text"

Keller-Segment

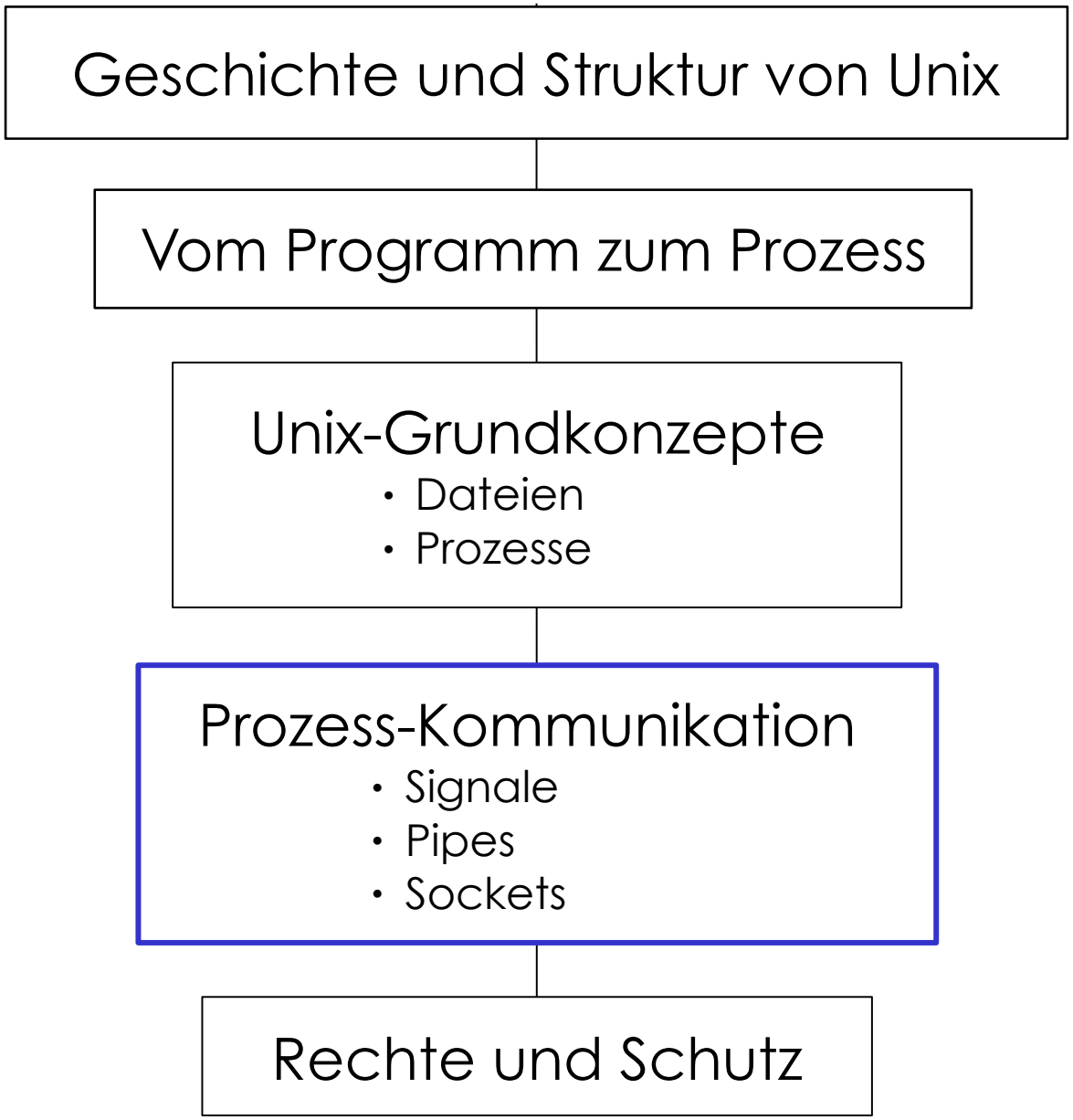
- Keller (Stack)
- enthält Parameter und Kontext (environment)

Systemstart und Login

- **`init`** – der erste Prozess



Wegweiser


Geschichte und Struktur von Unix
Vom Programm zum Prozess
Unix-Grundkonzepte
• Dateien
• Prozesse
Prozess-Kommunikation
• Signale
• Pipes
• Sockets
Rechte und Schutz

Prozesskommunikation: Signal, Pipe und Socket

Signale

- Senden von Signalen: z. B.
  - kill-Kernaufruf (**`kill(pid, SigNo)`**)
  - Terminaltreiber

- Disponieren:
  - gar nichts: Default-Verhalten, z. B. Abbruch
  - ignorieren: Signal verpufft
  - blockieren: Signal wird später zugestellt (nach unblock)
  - zustellen: Signalhandler wird aufgerufen

Signale

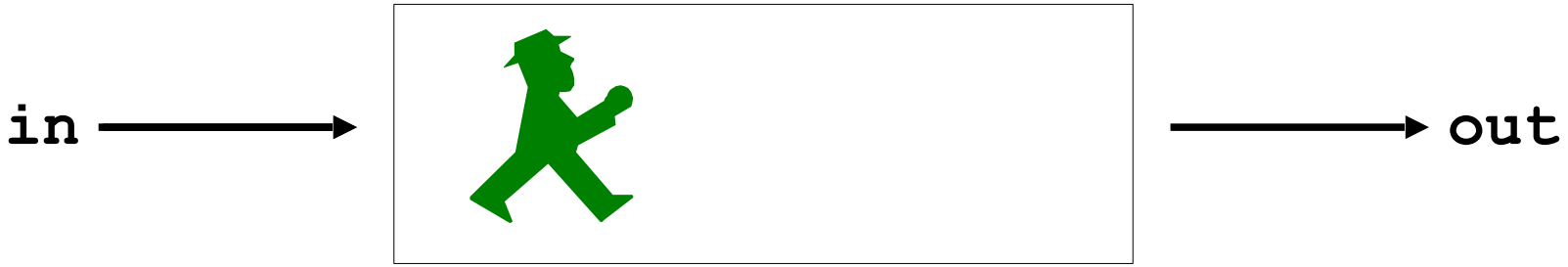
Signal	Cause
SIGABRT	Sent to abort process and force a core dump
SIGALARM	The alarm clock has gone off
SIGFPE	A floating point error has occurred (e.g., division by 0)
SIGHUP	The phone line the process was using has been hung up
SIGILL	The process has executed an illegal machine instruction
SIGINT	The user has hit the DEL key to interrupt the process
SIGQUIT	The user has hit the key requesting a core dump
SIGKILL	Sent to kill a process (cannot be caught or ignored)
SIGPIPE	The process has written on a pipe with no readers
SIGSEGV	The process has referenced an invalid memory address
SIGTERM	Used to request that a process terminate gracefully
SIGUSR1	Available for application-defined purposes
SIGUSR2	Available for application-defined purposes

Pipes und Filterketten

- Programme lesen von STDIN und schreiben nach STDOUT
- Kein Unterschied, ob lesen/schreiben von/in Datei oder über pipe zu einem anderen Prozess.

```
cat a > b

cat < a  > b

cat a | lpr

cat a | sort | lpr
```

Shell-Ebene: Prozesse als Filter



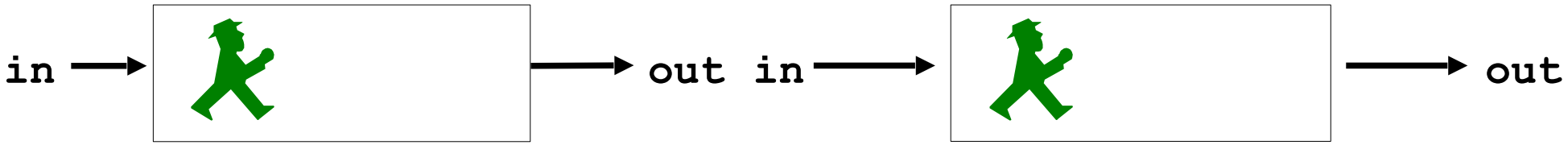
`cat < src.txt | sort`

`sort <in >out`

- Datenstrom als durchgängiges Konzept, spezielle Dateien per Konvention (`stdin`, `stdout`)
- Normalfall:
  - Tastatur als "standard in"
  - Terminal als "standard out"

→ Ein/Ausgabe als Spezialfall von Dateien

Shell-Ebene: Prozesse als Filter



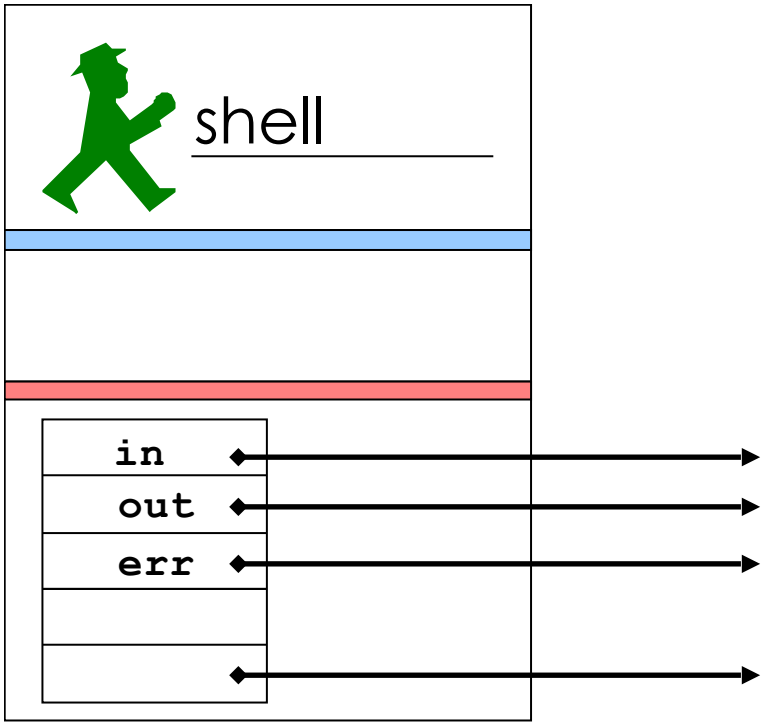

```
sort < mylist.txt | lpr
```

```
cat | sort | lpr
```

"Pipes" als spezielle Datenstrom-Dateien

→ Datenstrom als durchgängiges Konzept !

Kernschnittstelle: Datei-Deskriptoren



Beispiel: cat <in >out

```
read (command, params);

X = fork();

 if (X < 0) {
 // Fehlerbehandlung
 } else if (X != 0) {

  waitpid(X, &status, 0); // warte auf Kind-Prozess
 } else {

  close(0); open(in, ...);  // ersetze vorh. fd
  close(1); open(out, ...); // durch in/out

 exec(command, params, env);
 }
```

Pipes und Filterketten

z. B. **`cat | lpr`**

- **`pipe`**

erzeugt pipe mit 2 fd (fd1, fd2)

- **`fork`**: Kind1 für **`cat`**
- **`fork`**: Kind2 für **`lpr`**
- Elternteil (shell):

schliesst fd1, fd2

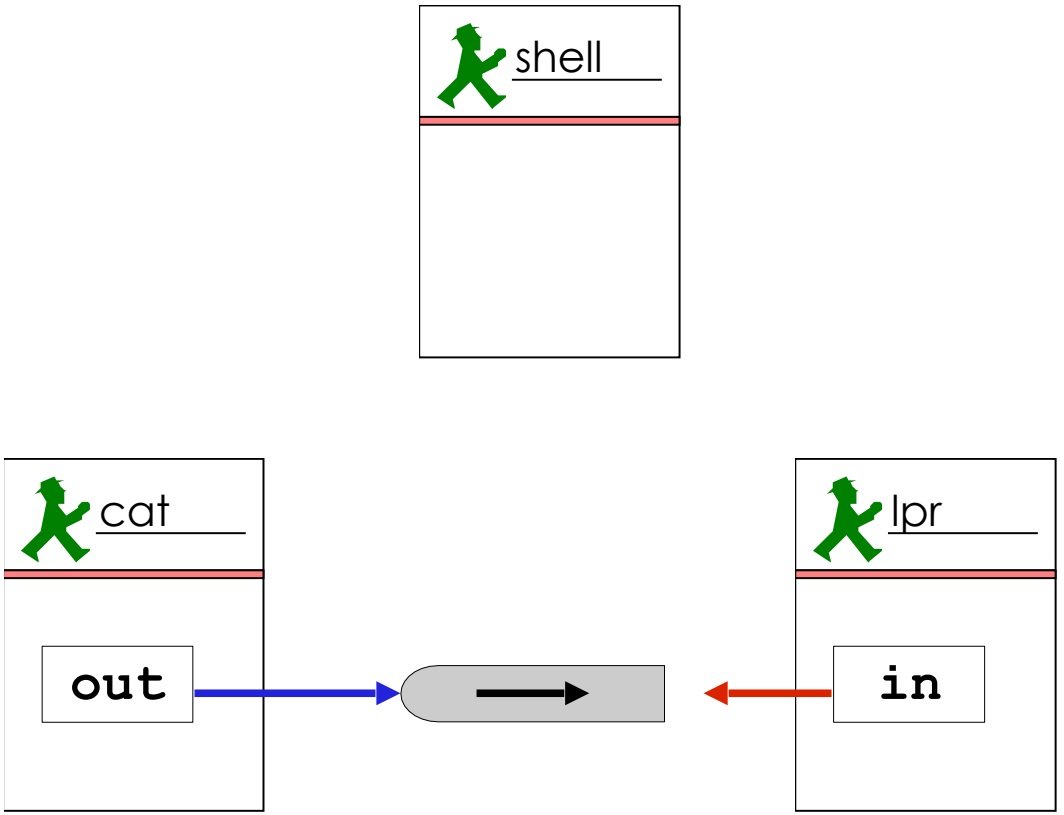
- Kind1:
  schließt fd2
  schließt stdout

fd1 → stdout

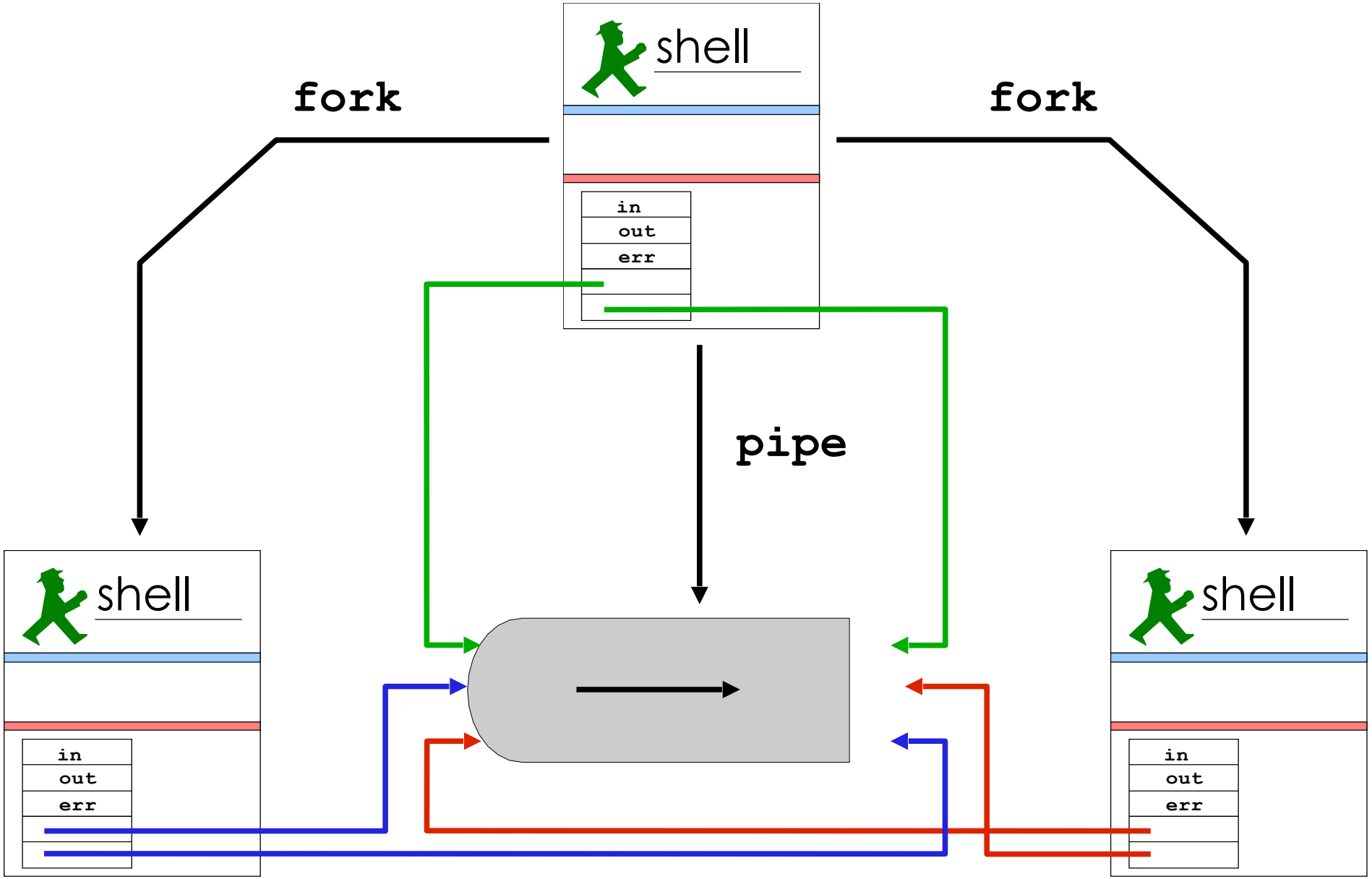
schließt fd1

`exec cat`

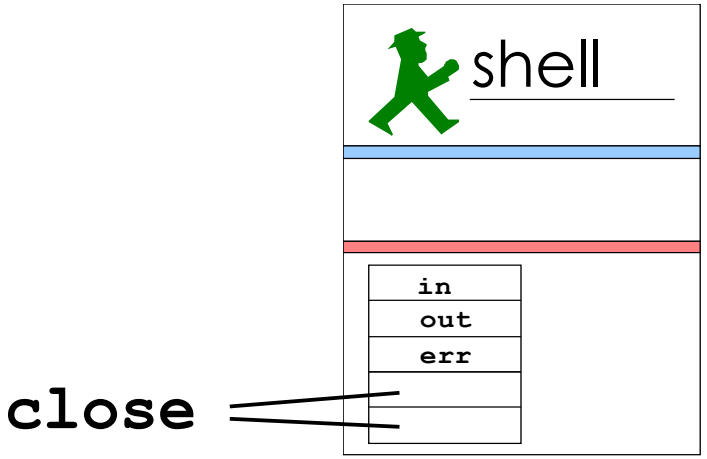
- Kind2: spiegelbildlich

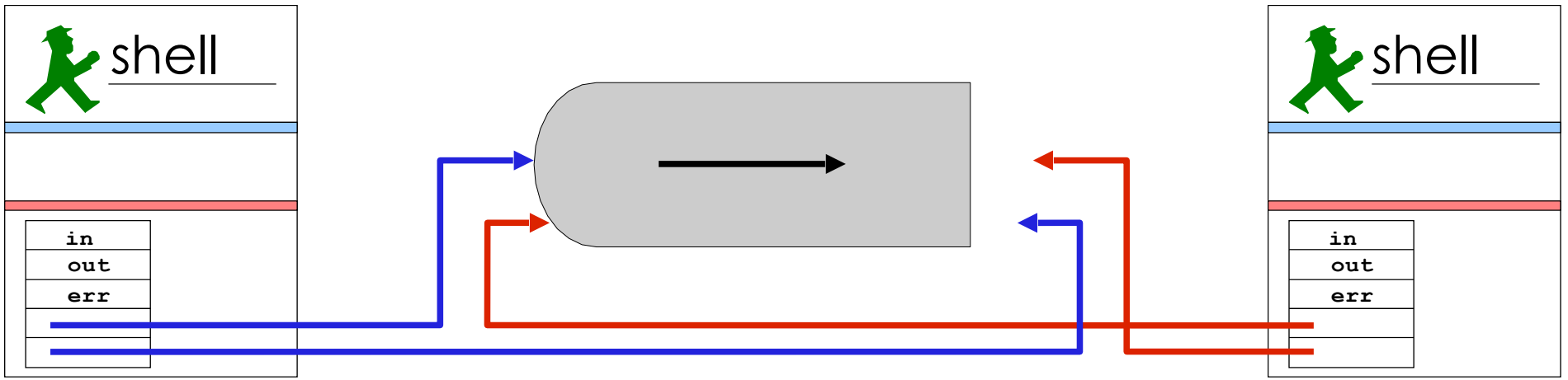


Aufbau einer Filterkette mit Pipes

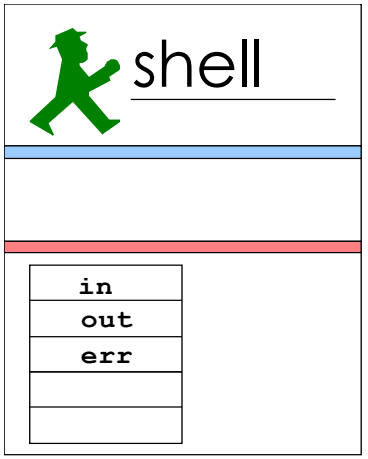


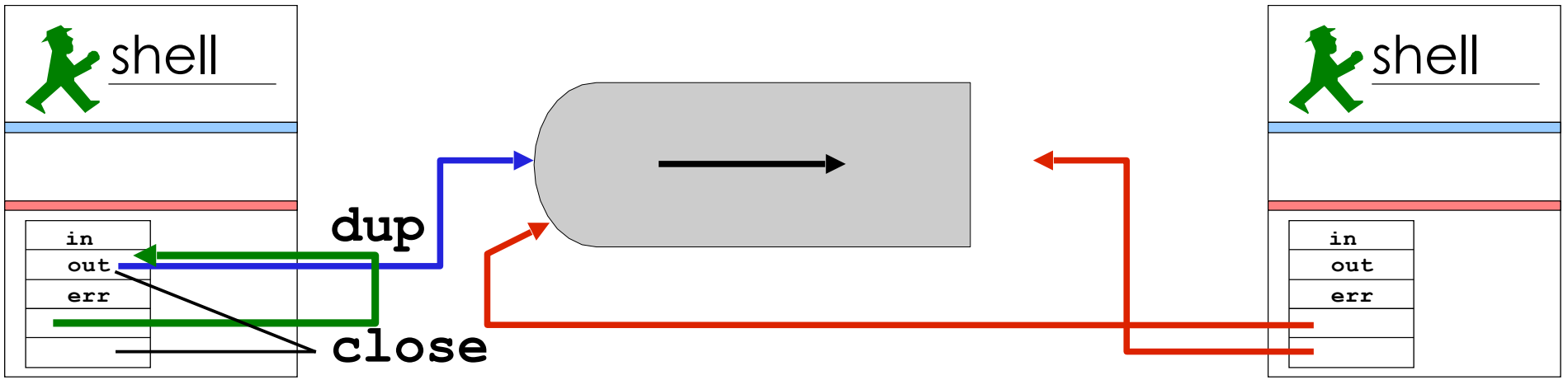
Aufbau einer Filterkette mit Pipes



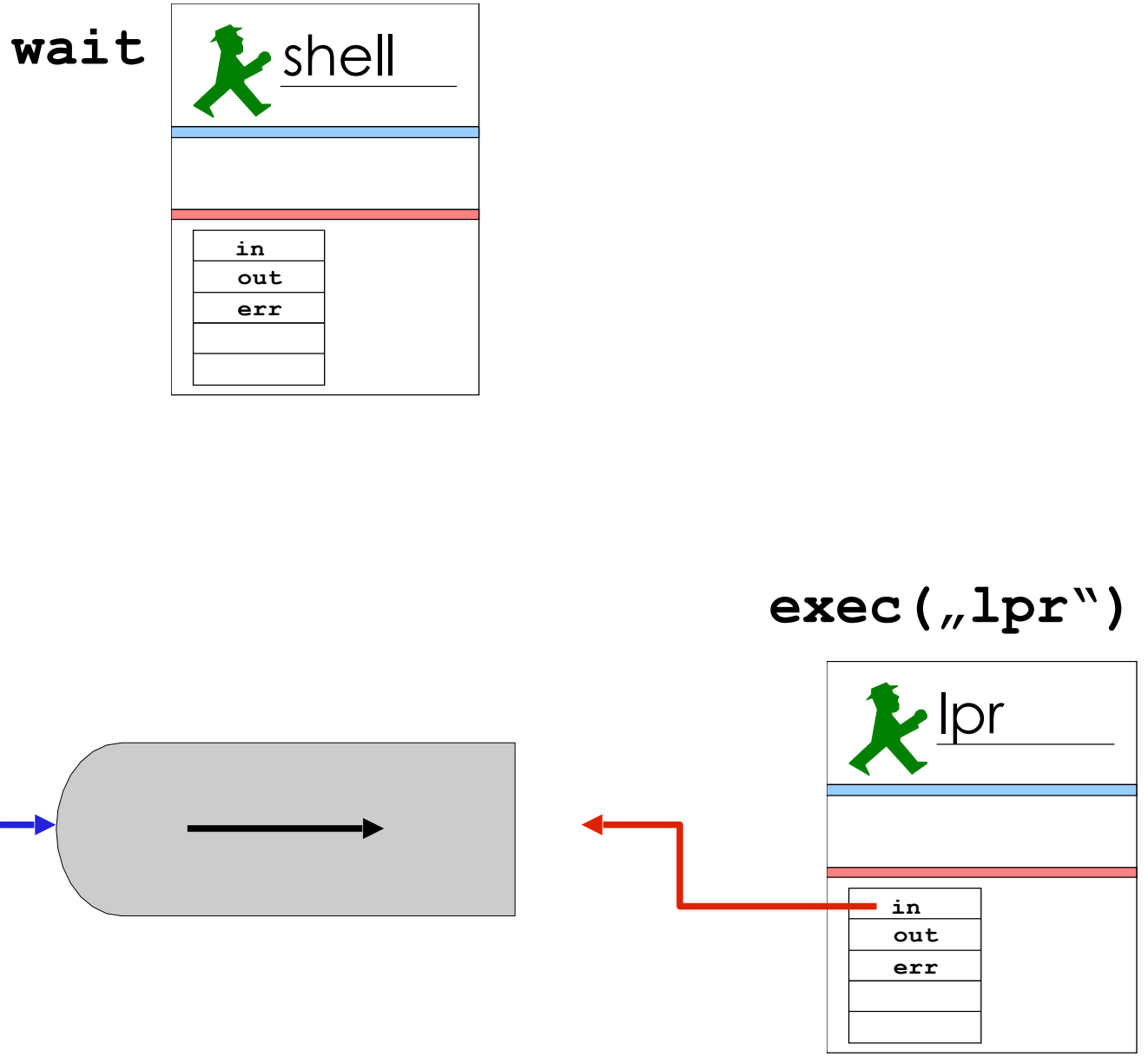


Aufbau einer Filterkette mit Pipes

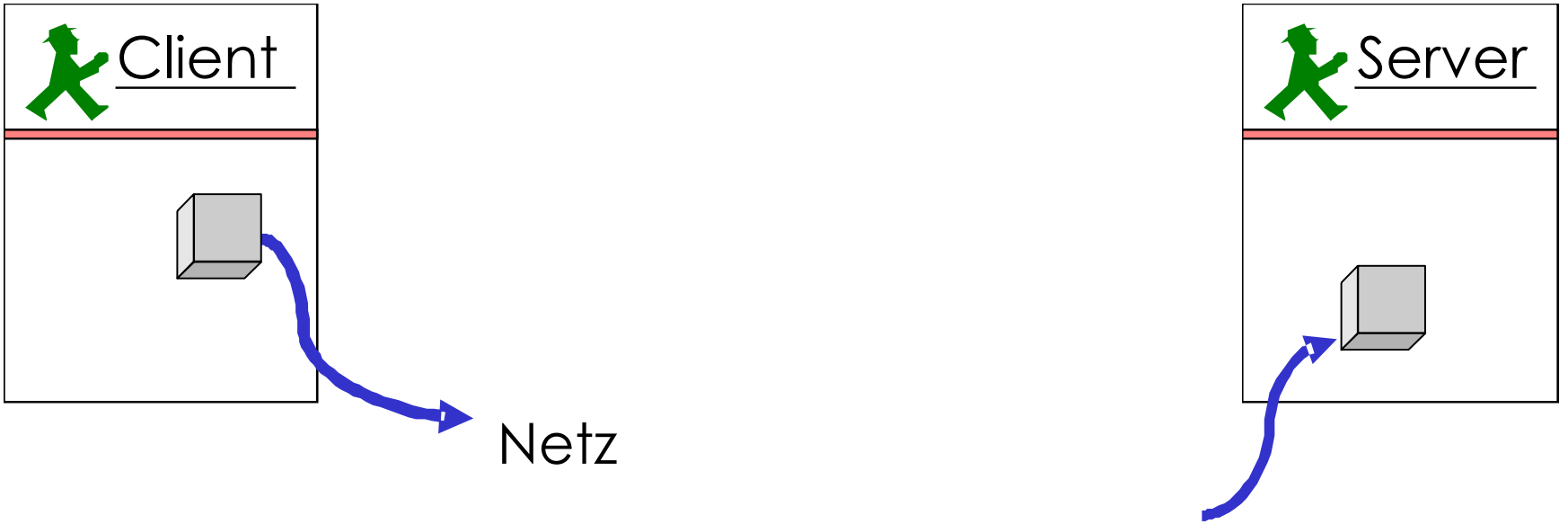




Aufbau einer Filterkette mit Pipes



Sockets



Client	Server

`create sock(protocol type)`
`connect(Adresse)`
	`create sock(protocol type)`
	`bind(Adresse)`
	`listen`
	`accept`

Wegweiser

Geschichte und Struktur von Unix

Vom Programm zum Prozess

Unix-Grundkonzepte
- Dateien
- Prozesse

Prozess-Kommunikation
- Signale
- Pipes
- Sockets

Rechte und Schutz

Rechte: Benutzer

- In Unix werden Benutzer (Prinzipale) dargestellt durch Uld (User-Id) und Gld (Group-Id)
- Zuordnung (klassisch): /etc/passwd
  (jeder kann zugreifen, verschlüsselt)
- Benutzer gehören zu einer (oder mehreren) Gruppen
  Zuordnung: /etc/group

Benutzer und Gruppen in Unix

/etc/passwd

```
root:x:0:0:Björn:/root:/bin/bash
sqrt:x:0:0:Mario:/root:/bin/tcsh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:100:sync:/bin:/bin/sync
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
irc:x:39:39:ircd:/var:/bin/sh
christiane:x:1000:100:Christiane:
   home/users/christiane:
   /bin/bash
johannes:x:1001:100:Johannes:
   /home/users/johannes:
   /bin/bash
hen:x:1002:100:Hendrik:
   /home/users/hen:
   /bin/tcsh
micha:x:1006:100:Michael:
   /home/users/micha:
   /bin/tcsh
...
```

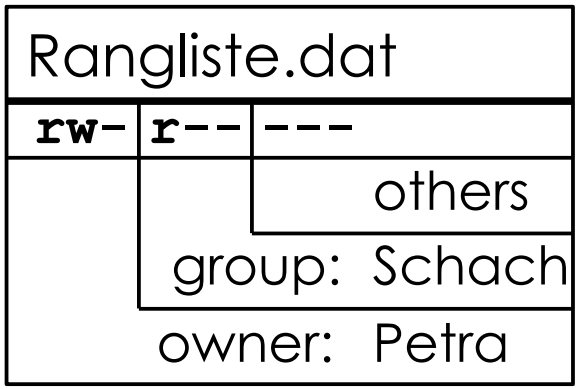
/etc/group

```
offline:x:102:ulli,iwer,veritaz

oea:x:103:veritaz,hen,bd1,iwer,
bjoern,robert,johnny,johannes,
ulli,nico

ese:!:104:iwer,veritaz,hen,chris,
benjamin,keiler,mario,ralf,bd1

www:!:105:chris,bjoern,iwer,
veritaz,hen,robert,anatol

ftp:x:106:chris,iwer

ifc:x:107:hen,reinhold,ulli,iwer,
micha
```

Rechte: Benutzer und Dateien

Zugriffsrechte zu Dateien festgelegt in Bezug auf Benutzer

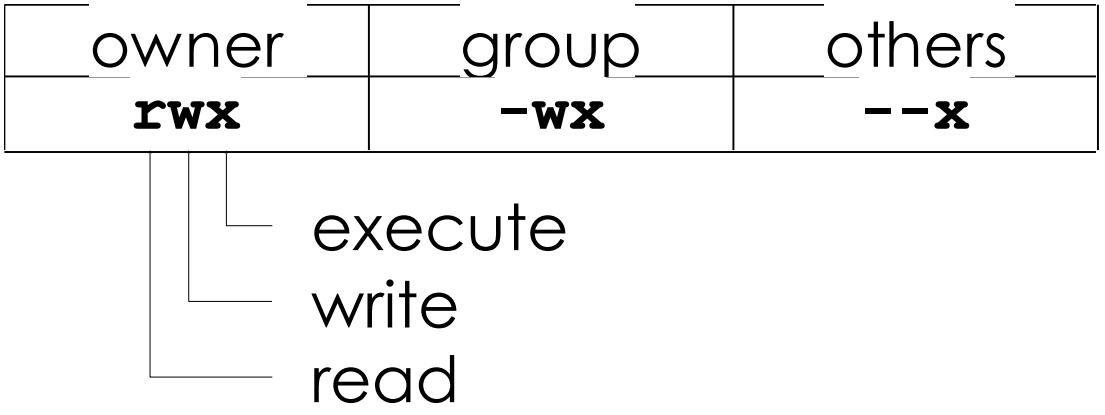
- jede Datei hat Attribute für Besitzer
  owner: Uld
  group: Gid

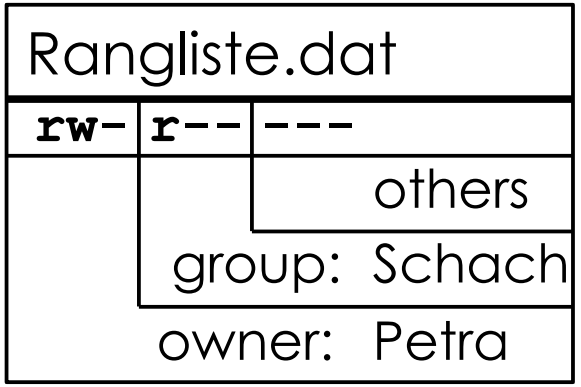


- Rechte an einer Datei werden festgelegt in Bezug auf
  owner
  group
  others (= Rest der Welt)

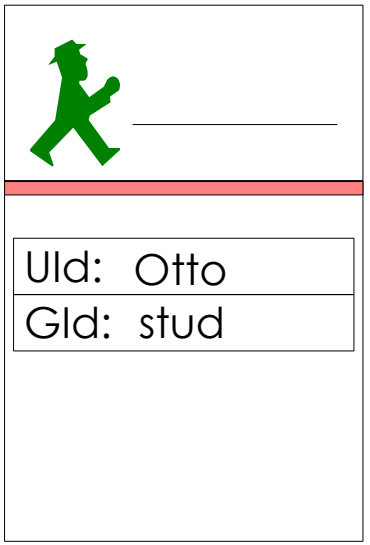
Benutzer und Prozesse

Rechte an Daten werden festgelegt in Bezug auf



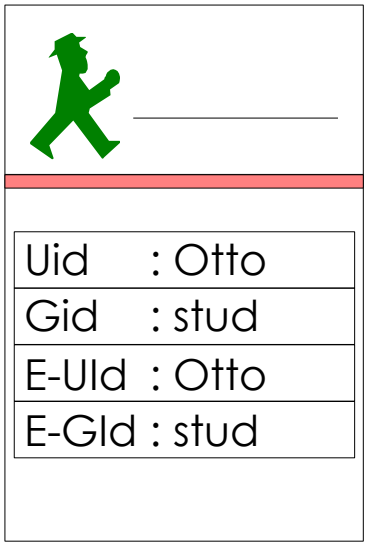


- Jeder Prozess übernimmt Uld und Gld vom „Eltern“-Prozess, die Rechte eines Benutzers leiten sich von Uld, GID ab



Benutzer und Prozesse

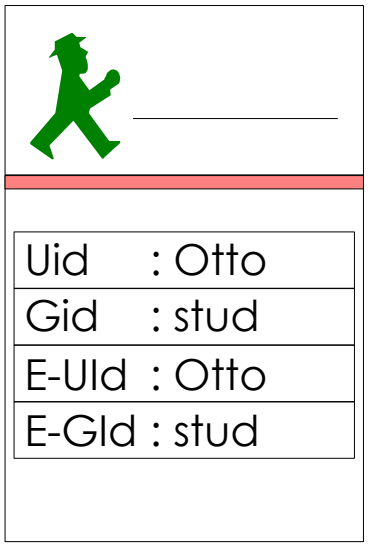
- Jeder Prozess repräsentiert einen Benutzer.
- Prozess-Attribute:
  - Uld, Gld
  - Effective-Uld, Effective-Gld

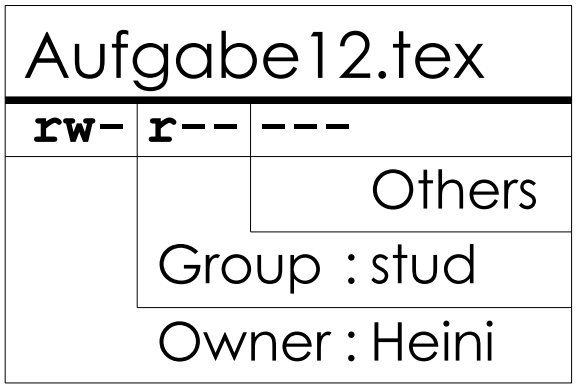


- Nur wenige hochprivilegierte Prozesse dürfen Uld und Gld manipulieren, z.B. Login-Prozess.
- Nach Überprüfung des Passwortes setzt Login-Prozess Uld, Gld, Eff-Uld, Eff-Gld.
- Alle anderen Prozesse: Kinder des Login-Prozesses.
- Kinder erben Attribute von Eltern.

Prozesse und Dateien

Die Attribute E-Uid und E-Gid bestimmen beim Zugriff auf Dateien die Rechte eines Prozesses.

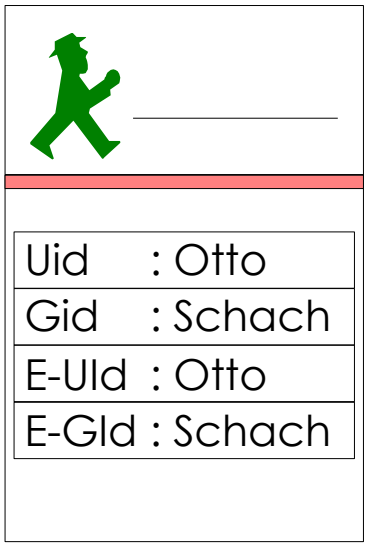


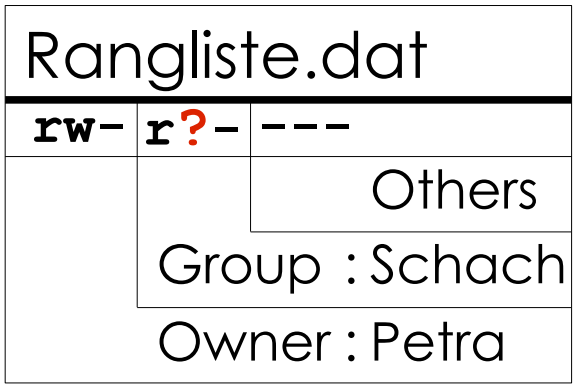


Problem: Rechteerweiterung

Beispiel: Schachrangliste

- Jeder Teilnehmer soll lesen können.
- Jeder Teilnehmer soll seine Ergebnisse schreiben können.
- Kein Teilnehmer soll darüber hinaus schreiben dürfen
  Fälschung der Rangliste verhindern.

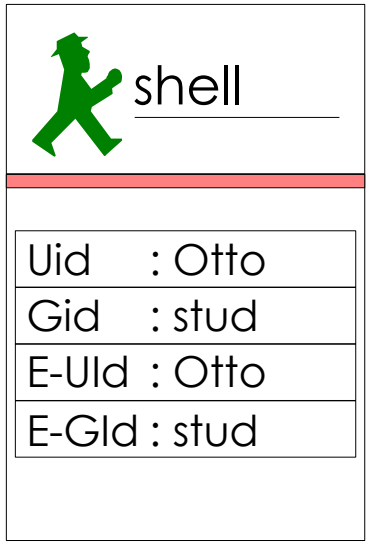


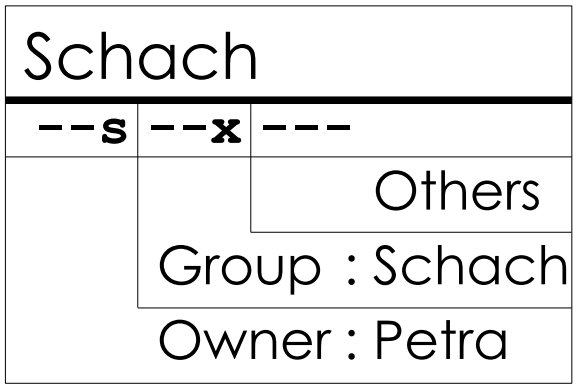


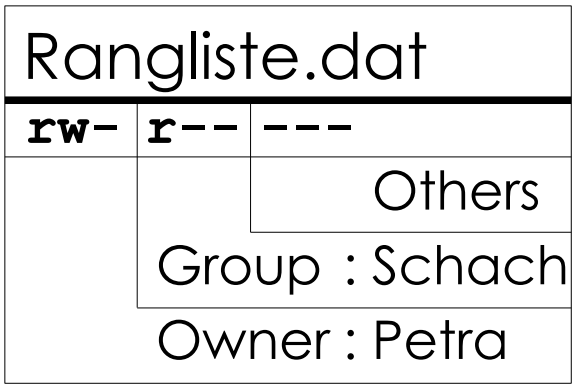
Unix-Lösung: SetUId-Mechanismus

- Datei, die vertrauenswürdigen Programmcode (z. B. Schach) enthält, besitzt Kennzeichnung als „Set-UID“ (s).
- Bei **`exec`** auf Set-UId Programme erhält ausführender Prozess als Effektive UId die UId des Installateurs (Owners) des Programms (genauer: der Datei, die Programm enthält).

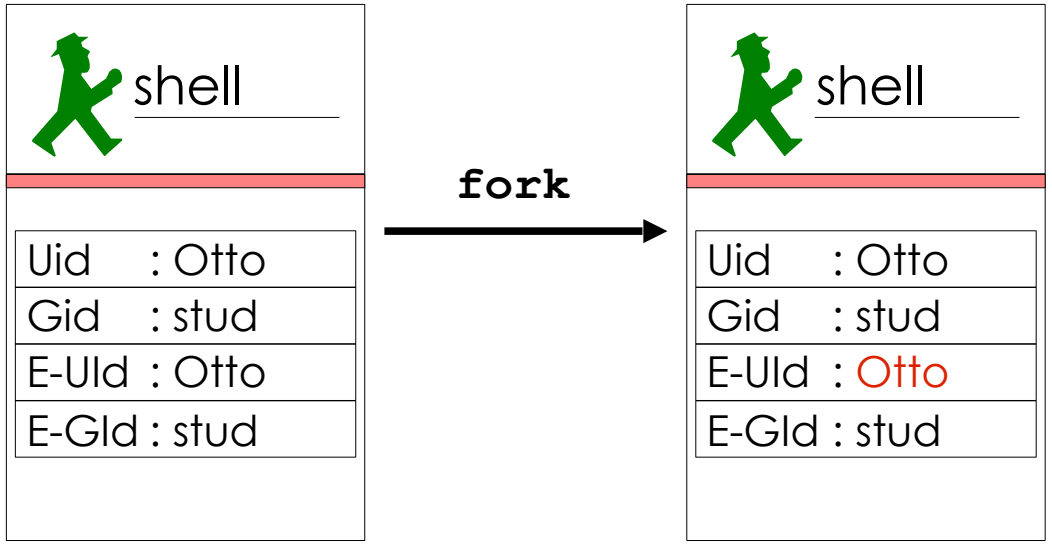
SetUId am Beispiel Rangliste

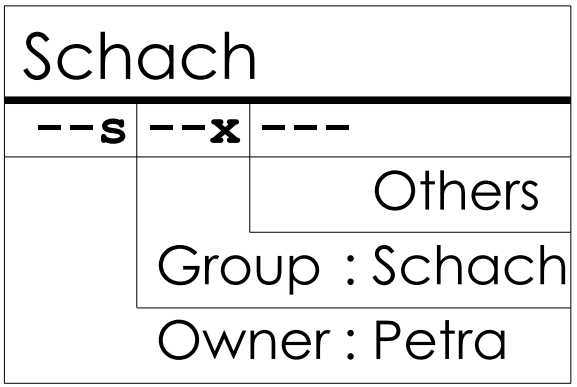


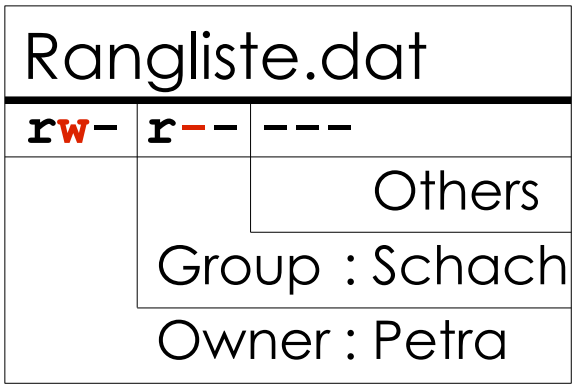




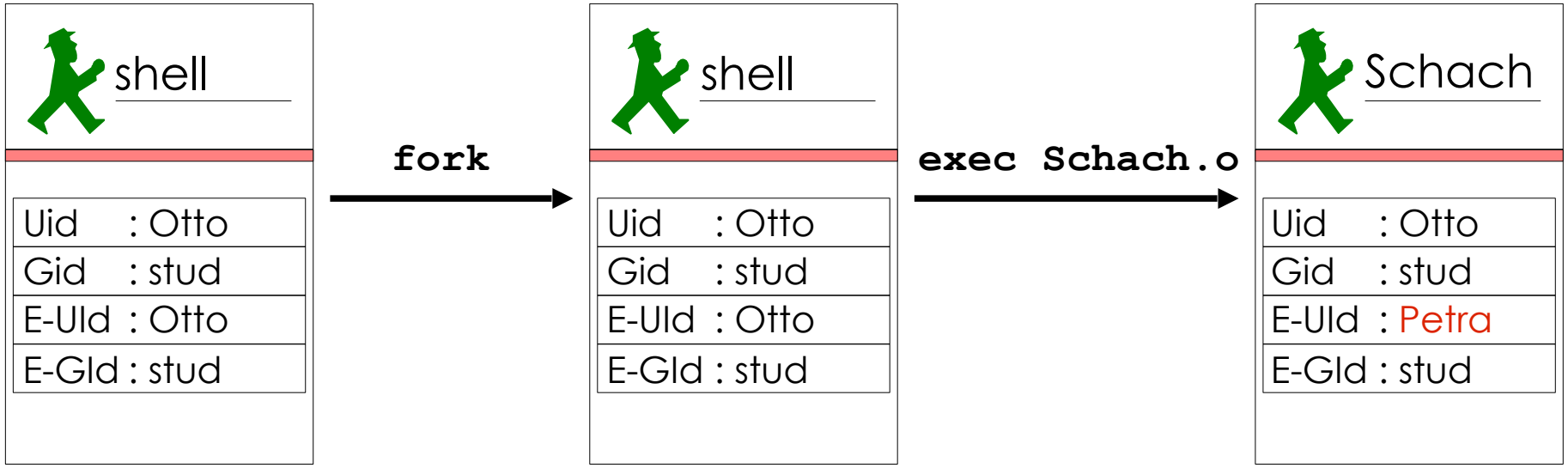
SetUId am Beispiel Rangliste

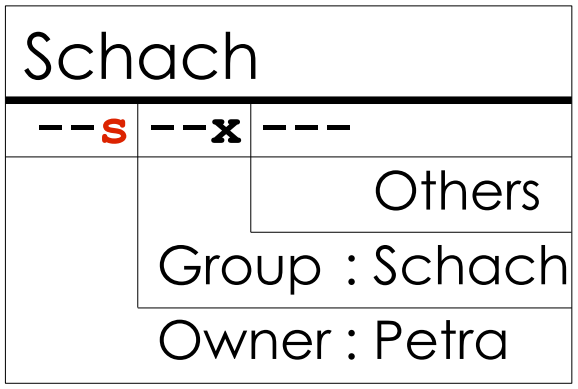


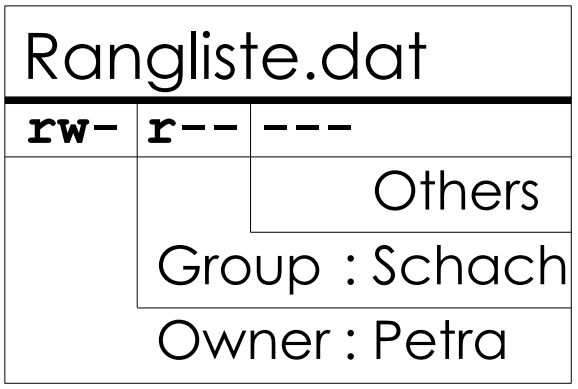




SetUId am Beispiel Rangliste







SetUId

- Erweiterung der Rechte eines Benutzers genau für den Fall der Benutzung dieses Programms.
- Installateur vertraut dem Benutzer, wenn er dieses Programm nutzt.

Probleme:

- Programmfehler führen zu sehr großen Rechteerweiterungen
- Bsp.: shell-Aufruf aus einem solchen Programm heraus

Zusammenfassung/Weiterführung

- Erfolgreiches Betriebssystem (akademisch, Workstations, Server)
- Wenige, einfache Designprinzipien
- Viele Versionen

Reimplementierungen/Ableger:

- Linux (Server, Desktop, eingebettete Systeme), Android
- Solaris (Server)
- Mac OS X, iOS