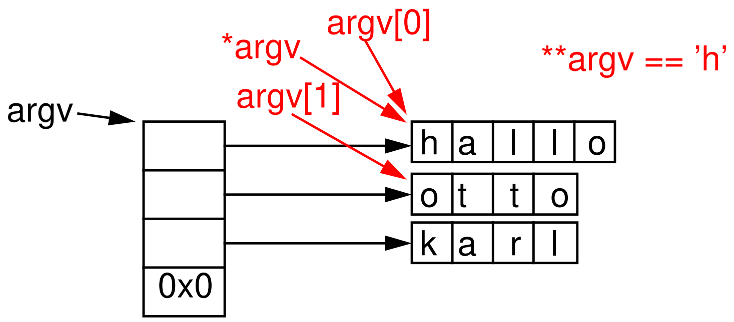
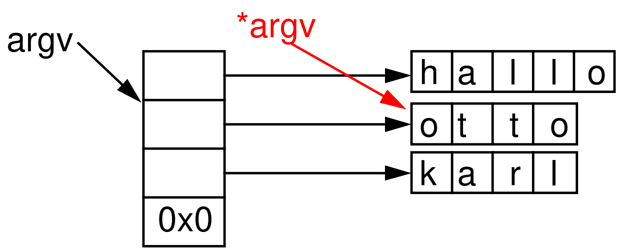
1 Zeiger-Vektoren

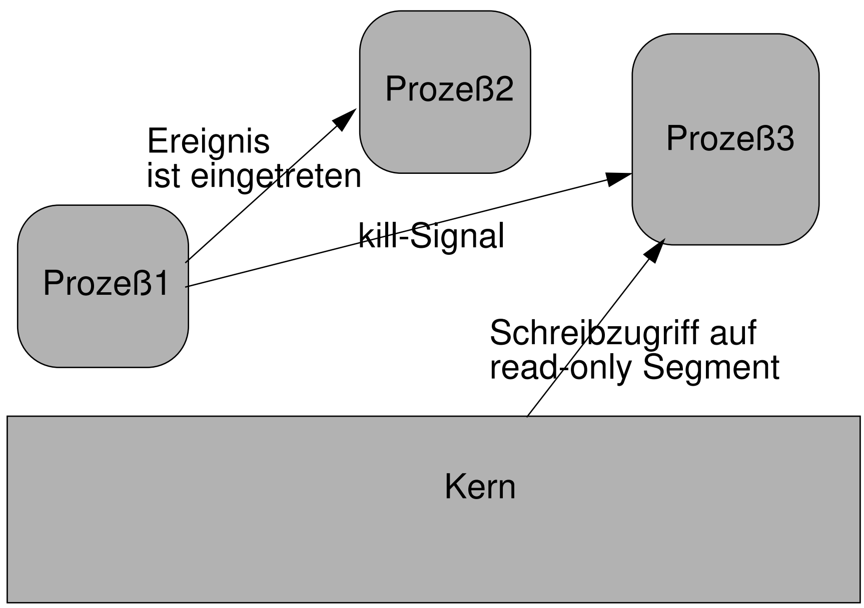
```
int main(int argc, char *argv[]) { ... }
```



- x = *argv++; entspr. x = *(argv++) entspr. { x = *argv; argv = argv + 1; }



2 IPC mit Signalen - Beispiele



3 Reaktion auf Signale: Default-Aktionen

- abort
  - ◆ erzeugt einen Core-Dump (Segmente + Registercontext) und beendet Prozeß
- exit
  - ◆ beendet Prozeß, ohne einen Core-Dump zu erzeugen
- ignore
  - ◆ ignoriert Signal
- stop
  - ◆ Stoppt Prozeß
- continue
  - ◆ setzt gestoppten Prozeß fort

4 Signalbehandlung

- Ignorieren

```
signal(sig, SIG_IGN);
```

- Default

```
signal(sig, SIG_DFL);
```

- Eigene Signalbehandlungsfunktion

```
void func(int sig) { ... }
...
int main(...) {
    ...
    signal(sig, func);
    ...
}
```

SP I
Übung zu Systemprogrammierung I
© Martin Steckermeier, Michael Golm, Universität Erlangen-Nürnberg, IMMD IV, 1997
uebung3.doc 1997-12-01 09.30
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Zurücksetzen des Signalhandlers

- Achtung: Bei System-V (z.B. Solaris, HP-UX) wird der Signalhandler vor dem Aufruf der Signalbehandlungsfunktion auf Default zurückgesetzt. In BSD und POSIX wird der Signalhandler nicht zurückgesetzt.
  - Folgendes Programm gibt bei **`kill -USR1 <PID>`** "Hallo" aus und registriert den Signalhandler erneut..

```
#include <signal.h>
#include <stdio.h>

void func(int s) {
  printf("Hallo\n");
  signal(SIGUSR1, func);
}

int main(int argc, char *argv[]) {

  signal(SIGUSR1, func);

  for(;;);

}
```


5 Wichtige Signale

- SIGABRT
  - Abort Signal; entsteht z.B. durch Aufruf von **`abort()`**
- SIGFPE
  - Floating Point Exception; z.B. Division durch 0 oder Overflow
- SIGILL
  - Illegal Instruction; z.B. break, privilegierte Operation, privilegiertes Register
- SIGINT
  - Interrupt; (Shell: CTRL-C)
- SIGSEGV
  - Segmentation violation; unkorrekter Zugriff auf Segment, z.B. schreiben auf Textsegment

5 Wichtige Signale (2)

- SIGTERM
  - Termination; Default-Signal für **`kill(1)`**
- SIGTSTP
  - Stop Signal; Stoppt Prozeß (Shell: CTRL-Z)
- SIGCONT
  - setzt einen gestoppten Prozeß fort
- SIGQUIT
  - Quit; Shell: CTRL-\
- SIGUSR1, SIGUSR2
  - Benutzerdefinierte Signale

5 Signale (nicht abfangbar)

- SIGKILL
  - beendet den Prozeß
- SIGSTOP
  - stoppt einen Prozeß

6 Senden von Signalen

```
#include <signal.h>
int kill(pid_t pid, int sig);
```

- Beispiel:

```
#include ...

int main(int argc, char *argv[]) {
  int pid;
  int status;

  pid = fork();
  if (pid > 0) {
    sleep(5); /* kind Zeit zum Rechnen geben */
    kill(pid, SIGKILL); /* Asta la vista, Baby! */
    wait(&status);
    if (WIFEXITED(status)) {
      printf("Status: exited\n");
    } else if (WIFSIGNALED(status)) {
      printf("Status: signaled\n");
    } else {
      printf("Status: %d\n",status);
    }
  } else if (pid==0) {
    for(;;) ; /* kindprozess in Endlossschleife */
  } else {
    perror("fork");
  }
}
```


7 Signale und Systemaufrufe

- Wenn ein Signal während eines blockierten Systemaufrufs (z.B. read(), wait()) empfangen wird, wird
  - ◆ der Systemaufruf mit **`errno==EINTR`** abgebrochen (SVR4-Unix: HP-UX, Solaris)
  - ◆ der Systemaufruf nochmals gestartet (BSD-Unix)

8 Verzögerte Signale

- BSD, POSIX (z.B. HP-UX, Solaris)
- wenn ein Prozeß in einer Signalbehandlungsroutine ist und ein neues Signal des gleichen Typs eintrifft wird das Signal zwischengespeichert
- bei Verlassen der Signalbehandlungsroutine wird das Signal deblockiert
- es wird maximal ein Signal zwischengespeichert