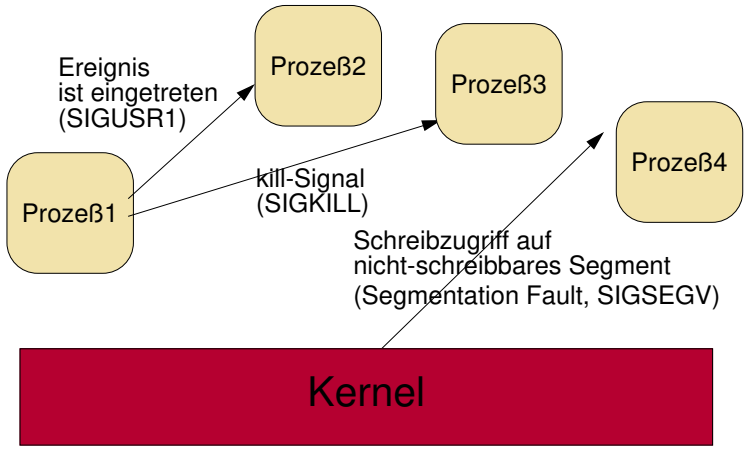
5. Tafelübung

- Lösung der ldir-Aufgabe
- Erläuterung der jsh-Aufgabe
- POSIX-Signale

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

IPC mit Signalen - Beispiele



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

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

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Posix Signalbehandlung

- sigaction
- sigprocmask
- sigsuspend
- sigpending

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Signalhandler installieren (sigaction)

- Prototyp

```
#include <signal.h>

int sigaction(int sig, /* Signal */
              const struct sigaction *act, /* Handler */
              struct sigaction *oact /* Alter Handler */ );
```

- Handler bleibt solange installiert, bis neuer Handler mit **`sigaction`** installiert wird
- sigaction Struktur

```
struct sigaction {
    void (*sa_handler)(int);
    sigset_t sa_mask;
    int sa_flags;
}
```


© Michael Golm, Universität Erlangen-Nürnberg, IMMD 4, 1999/2000
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

sigaction Handler (sa_handler)

- ist ein Funktionspointer oder einer der vordefinierten Werte
  - **`SIG_DFL`**: Default Signalbehandlung
  - **`SIG_IGN`**: Signal ignorieren

© Michael Golm, Universität Erlangen-Nürnberg, IMMD 4, 1999/2000
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

sigaction Maske (sa_mask)

- verzögerte Signale
  - während der Ausführung der Signalhandler-Prozedur wird das auslösende Signal blockiert
  - bei Verlassen der Signalbehandlungsroutine wird das Signal deblockiert
  - es wird maximal ein Signal zwischengespeichert
- mit **`sa_mask`** in der **`struct sigaction`** kann man zusätzliche Signale blockieren
- Auslesen und Modifikation der Signal-Maske vom Typ **`sigset_t`** mit:
  - **`sigaddset()`**: Signal zur Maske hinzufügen
  - **`sigdelset()`**: Signal aus Maske entfernen
  - **`sigemptyset()`**: Alle Signale aus Maske entfernen
  - **`sigfillset()`**: Alle Signale in Maske aufnehmen
  - **`sigismember()`**: Abfrage, ob Signal in Maske enthalten ist

© Michael Golm, Universität Erlangen-Nürnberg, IMMD 4, 1999/2000
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

sigaction Flags (sa_flags)

- Durch Flags läßt sich das Verhalten beim Signalempfang beeinflussen.
- Kann für jedes Signal gesondert gesetzt werden.
- **`SA_NOCLDSTOP`**: SIGCHLD wird nur erzeugt, wenn Kind terminiert, nicht wenn es gestoppt wird (POSIX, SVID, BSD)
- **`SA_RESTART`**: durch das Signal unterbrochene Systemaufrufe werden automatisch neu aufgesetzt (kein errno=EINTR) (nur SVID und BSD)
- **`SA_SIGINFO`**: Signalhandler bekommt zusätzliche Informationen übergeben (nur SVID)

```
void func(int signo, siginfo_t *info, void *context);
```

- **`SA_NODEFER`**: Signal wird während der Signalbehandlung nicht blockiert (nur SVID)

© Michael Golm, Universität Erlangen-Nürnberg, IMMD 4, 1999/2000
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Beispiel

- Beispiel:

```
#include <signal.h>
void my_handler(int sig) { ... }
...
struct sigaction action;
sigemptyset(&action.sa_mask);
action.sa_flags = 0;
action.sa_handler = my_handler;
sigaction(SIGUSR1, &action, NULL); /* return abfragen ! */
```

- Signal schicken mit **`kill -USR1 <pid>`** oder mit

```
int kill(pid_t pid, int signo);
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Ändern der prozessweiten Signal-Maske

```
int sigprocmask(int how, /* Verknüpfung der Masken */
                const sigset_t *set, /* neue Maske */
                sigset_t *oset /* Speicher für alte Maske */ );
```

- how:
  - **`SIG_BLOCK`**: Vereinigungsmenge zwischen übergebener und alter Maske
  - **`SIG_SETMASK`**: Setzen der Maske ohne Beachtung der alten Maske
  - **`SIG_UNBLOCK`**: Schnittmenge zwischen inverser übergebener Maske und alter Maske
- Beispiel

```
sigset_t set;
sigemptyset(&set);
sigaddset(&set, SIGUSR1);
sigprocmask(SIG_BLOCK, &set, NULL);
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Warten auf Signale

- **`sigsuspend(mask)`** wartet auf Signale, die in **`mask`** enthalten sind
- **`mask`** wird damit zur aktuellen Signal-Maske
- kehrt nach Bearbeitung des Signalhandlers zurück

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Abfrage blockierter Signale

- **`sigpending(set)`** speichert alle Signale, die blockiert sind, aber empfangen wurden, in **`set`** ab

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

POSIX Signale

Das Defaultverhalten bei den meisten Signalen ist die Terminierung des Prozesses, bei einigen Signalen mit Anlegen eines Coredumps.

- SIGABRT (core): Abort Signal; entsteht z.B. durch Aufruf von **`abort()`**
- SIGALRM: Timer abgelaufen (**`alarm(), setitimer()`**)
- SIGFPE (core): Floating Point Exception; z.B. Division durch 0 oder Overflow
- SIGHUP: Terminalverbindung wird beendet (Hangup)
- SIGILL (core): Illegal Instruction; z.B. privilegierte Operation, privilegiertes Register
- SIGINT: Interrupt; (Shell: CTRL-C)
- SIGKILL (nicht abfangbar): beendet den Prozeß

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

POSIX Signale (2)

- SIGPIPE: Schreiben auf Pipe oder Socket nachdem der lesende terminiert ist
- SIGQUIT (core): Quit; (Shell: CTRL-\)
- SIGSEGV (core): Segmentation violation; inkorrekter Zugriff auf Segment, z.B. Schreiben auf Textsegment
- SIGTERM: Termination; Default-Signal für **`kill(1)`**
- SIGUSR1, SIGUSR2: Benutzerdefinierte Signale

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Jobcontrol-Signale

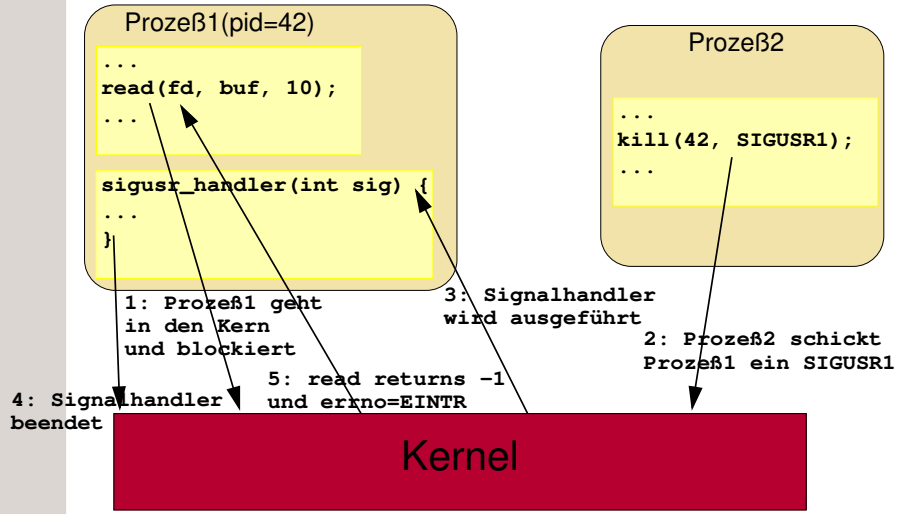
Diese Signale exisitieren in einem POSIX-konformen System nur, wenn das System Jobkontrolle unterstützt (**`_POSIX_JOB_CONTROL`** ist definiert).

- SIGCHLD (Defaultaktion ist Ignorieren): Status eines Kindprozesses hat sich geändert
- SIGCONT: setzt den gestoppten Prozeß fort
- SIGSTOP (nicht abfangbar): stoppt den Prozeß
- SIGTSTP: stoppt den Prozeß (Shell: CTRL-Z)
- SIGTTIN, SIGTTOU: Hintergrundprozeß wollte vom Terminal lesen bzw. darauf schreiben

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Unterbrechen von Systemcalls

- Signale können die Ausführung von Systemaufrufen unterbrechen



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

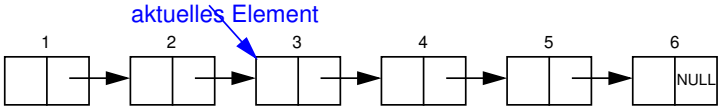
Unterbrechen von Systemcalls

- dies betrifft nur "langsame Systemcalls" (welche sich über einen längeren Zeitraum blockieren können, z.B. **`wait()`**, **`waitpid()`** oder **`read()`** von einem Socket oder einer Pipe)
- der Systemcall setzt dann **`errno`** auf **`EINTR`**
- in einigen UNIXen (z.B. 4.2BSD) werden unterbrochene Systemcalls automatisch neu aufgesetzt
- bei einigen UNIXen (SVR4, 4.3BSD), kann man für jedes Signal einstellen (**`SA_RESTART`**), ob ein Systemcall automatisch neu aufgesetzt werden soll
- POSIX.1 läßt dies unspezifiziert

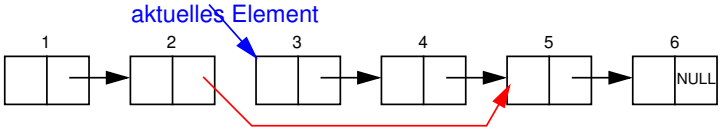
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Signale und Race Conditions

- Signale erzeugen Nebenläufigkeit innerhalb des Prozesses
- diese Nebenläufigkeit kann zu Race-Conditions führen
- Beispiel:
  - main-Funktion läuft durch eine verkettete Liste



  - Prozeß erhält Signal; Signalhandler entfernt Elemente 3 und 4 aus der Liste und gibt den Speicher dieser Elemente frei



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

Signale und Race Conditions

- Lösung: Signal während Ausführung des kritischen Abschnitts blockieren!
- weiteres Problem:
  - Aufruf von Bibliotheksfunktionen, z.B. **`getpwuid()`**, wird durch Signal unterbrochen und nach Ausführung des Signalhandlers fortgesetzt
  - Signalhandler ruft auch **`getpwuid()`** auf-> Race Condition!
- Lösung:
  - in Signalhandlern nur Funktionen aufrufen, die in POSIX.1 als reentrant gekennzeichnet sind (**`getpwuid`** und **`malloc`**/**`free`** sind z.B. nicht reentrant, **`wait`** und **`waitpid`** sind reentrant)
  - wenn in einem Signalhandler Funktionen verwendet werden, die errno verändern, muß der Wert von errno vorher gesichert und vor Beendigung des Signalhandlers wieder zurückgesetzt werden

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

signal()-Funktion

- ANSI-C definiert die signal()-Funktion zum Installieren von Signalhandlern
  - Problem: sehr ungenaue Spezifikation, da Prozeßkonzept in ANSI-C nicht definiert
- BSD- und SystemV-Unixe enthalten die signal-Funktion
  - Problem: Prozeßkonzept jetzt definiert, aber signal-Semantik ist von Unix Version 7 abgeleitet und unzuverlässig (*unreliable signals*) (Signalhandler bleibt nicht installiert,Signale können nicht blockiert werden)
- signal() ist deshalb in POSIX.1 nicht enthalten und sollte auch nicht mehr benutzt werden

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.