FH | JOANNEUM
University of Applied Sciences

Systemnahe Programmierung
SS 2024

Unit 7

Interprozesskommunikation

Mechanismen

Die Interprozesskommunikation ermöglicht den Datenaustausch zwischen zwei oder mehreren Prozessen.

- Signale
- Pipes
- FIFO
- Message-Queues
- Semaphore
- Shared Memory

Für einige IPC Mechanismen existieren zwei verschiedene API Versionen, die ursprüngliche (System-V) und eine neuere POSIX Variante. Wir verwenden immer die POSIX Variante.

Signale

Signale

Signale sind asynchrone Ereignisse und bewirken eine Unterbrechung auf der Prozessebene (Software-Interrupt).

Signale haben eine Bezeichnung die mit SIG... beginnt.

Beispiele:

SIGILL ... illegal Instruction
SIGINT ... User generated Interrupt (Ctrl+C)
SIGKILL ... Terminate process
usw.

Signale können

- vom Kernel an einen Prozess
- von einem Prozess an einen anderen Prozess (Bsp.: kill -9)
- von einem Prozess an sich selbst

gesendet werden.

Signale

Signalhandling

Wie kann ein Prozess auf Signale reagieren:

- Ignorieren
  Das Signal wird verworfen. Die beiden Signale SIGKILL und SIGSTOP können nicht ignoriert werden.

- Blockieren
  Das Signal wird solange in einer Queue abgelegt, bis der Prozess die Blockierung aufhebt. Das Signal ist im Status „Pending“.
  Welche Signale blockiert werden, wird durch die Signalmaske des Prozesses bestimmt. SIGKILL und SIGSTOP können nicht blockiert werden.

Signale

Signalhandling

Wie kann ein Prozess auf Signale reagieren:

- Die Default-Aktion ausführen
  In vielen Fällen ist die Default-Aktion das Programm zu beenden. Signale, die für sehr spezielle Situationen gedacht sind, werden ignoriert.

Folgende Default Actions sind definiert:

- Term ... Prozess terminieren
- Ign ... Das Signal ignorieren
- Core ... Den Prozess terminieren und einen Core-Dump speichern.
- Stop ... Den Prozess stoppen
- Cont ... Den Prozess fortsetzen, wenn er aktuell gestoppt ist

Signale

Signalhandling

Wie kann ein Prozess auf Signale reagieren:

- Auf das Signal mit einem Handler reagieren (Das Signal "abfangen")
  Für ein bestimmtes Signal kann eine Funktion als Signal-Handler registriert werden. Wenn das Signal empfangen wird, dann wird das Programm unterbrochen und die als Handler registrierte Funktion wird ausgeführt. Anschließend wird das Programm an der Stelle der Unterbrechung fortgesetzt.

Signalhandler werden mit `fork()` and den Kindprozess vererbt, nach einem `exec()` Call werden sie gelöscht.

Signale

Wichtige Signale

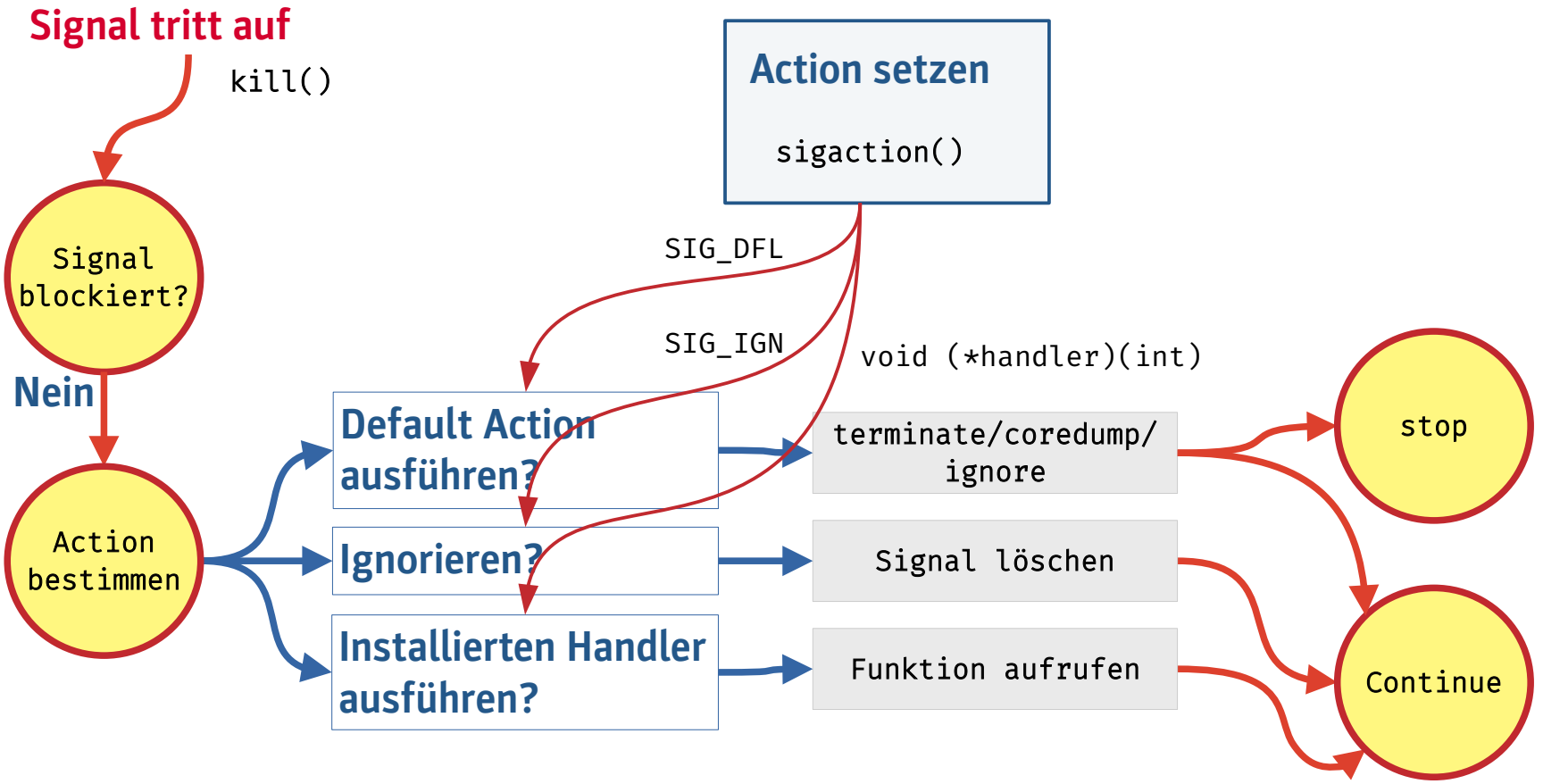
Eine Liste aller Signale kann mit dem Befehl `kill -l` angezeigt werden.

Einige Signale:

- SIGKILL
  SIGKILL terminiert den Prozess sofort.
- SIGINT
  Unterbrechung, Ctrl+C
- SIGTERM
  Wenn es abgefangen wird, ist ein kontrolliertes Beenden möglich.
- SIGCHLD
  Wird an den Elternprozess gesendet wenn ein Kindprozess beendet wurde.
- SIGUSR1, SIGUSR2
  Frei zur eigenen Benutzung

Signale

Signalhandling



Signale

Operationen

Das Verhalten eines Prozesses beim Auftreten eines Signals festlegen:

```
int sigaction(int signum, const struct sigaction *act,
              struct sigaction *oldact);
```

Auslösen (senden) von Signalen an Prozesse:

```
int raise(int sig_nr);
int kill(pid_t pid, int sig)
```

Manipulation von Signalmengen:

```
int sigemptyset(sigset_t *sig_set);
int sigfillset(sigset_t *sig_set);
int sigaddset(sigset_t *set, int signum);
int sigdelset(sigset_t *set, int signum);
int sigismember(const sigset_t *set, int signum);
```

Signale

Verhalten bei Signalen festlegen

Die Funktion `sigaction()` ermöglicht es, zu spezifizieren wie ein Prozess auf ein Signal reagieren soll, bzw. die aktuellen Settings abzufragen.

```
#include <signal.h>
int sigaction(int signum, const struct sigaction *act,
              struct sigaction *oldact);
```

`signum` ... Nummer des Signals, dessen Settings bearbeitet werden sollen. Konstante dafür sind definiert (z.B.: `SIGINT`, `SIGHUP` ...)
`act` ... neue Settings oder `NULL`, wenn nur abgefragt werden soll
`oldact` ... aktuelle Settings oder `NULL`, wenn nur neue gesetzt werden sollen.

Die Settings werden mit dieser Struktur gesetzt/abgefragt:

```
struct sigaction {
  void     (*sa_handler)(int); → Handler Funktion
  void     (*sa_sigaction)(int, siginfo_t *, void *);
  sigset_t sa_mask; → Maske für Signale die während des Handlers auftreten
  int      sa_flags; → Details wie Signale behandelt werden.
  void     (*sa_restorer)(void);
};
```

Signale

Verhalten bei Signalen festlegen

Verwendung der Felder in der Struktur `sigaction`.

```
struct sigaction {
  void     (*sa_handler)(int);
  void     (*sa_sigaction)(int, siginfo_t *, void *);
  sigset_t sa_mask;
  int      sa_flags;
  void     (*sa_restorer)(void);
};
```

Handler(Funktion) die für das Signal eingerichtet werden soll:

`SIG_DFL` ... Default Action
`SIG_IGN` ... Signal ignorieren

oder eigener Handler mit der Signatur

```
void (*sa_handler)(int);
```

Als Parameter wird die Nummer des Signals übergeben, das den Handler ausgelöst hat.

Signale

Verhalten bei Signalen festlegen

Verwendung der Felder in der Struktur `sigaction`.

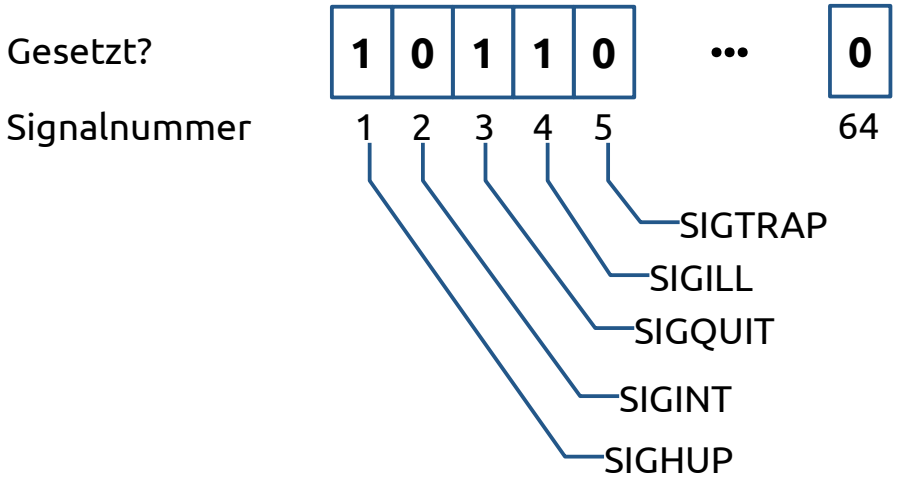
```
struct sigaction {
  void     (*sa_handler)(int);
  void     (*sa_sigaction)(int, siginfo_t *, void *);
  sigset_t sa_mask;
  int      sa_flags;
  void     (*sa_restorer)(void);
};
```

`sa_mask` ist die Signalmenge jener Signale, die während der Ausführung des Signalhandlers blockiert werden sollen.
Das Signal das den Handler ausgelöst hat wird blockiert, außer das Flag `SA_NODEFER` ist gesetzt.

Signale

Signalmengen

Signalmengen sind eine Bitmenge, in der jedes Bit einem Signal zugeordnet ist.



Wozu werden sie verwendet:
In den Funktionen mit denen die Signalbearbeitung verwaltet wird, können mit einer Signalmenge mehrere Signale auf einmal bearbeitet werden (z.B.: Signalmaske)

Signale

Signalmenge

Funktionen um Signalmengen zu bearbeiten:

```
#include <signal.h>

int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);
int sigdelset(sigset_t *set, int signum);
```

`sigset_t` ist eine Bitmaske (64 Bit), in der jedes Bit für ein Signal steht.

`sigemptyset()` initialisiert `set` als leere Menge -> kein Signal gesetzt.
`sigfillset()` initialisiert `set` so, dass alle Signale gesetzt sind.
`sigaddset()` setzt das Signal mit der Nummer `signum` in der Menge `set`.
`sigdelset()` löscht das Signal mit der Nummer `signum` in der Menge `set`.

Die Funktionen retournieren 0 wenn OK, sonst -1.

Signale

Verhalten bei Signalen festlegen

Verwendung der Felder in der Struktur `sigaction`.

```
struct sigaction {
  void     (*sa_handler)(int);
  void     (*sa_sigaction)(int, siginfo_t *, void *);
  sigset_t sa_mask;
  int      sa_flags;
  void     (*sa_restorer)(void);
}
```

Die Flags sind nur wirksam beim Hinzufügen eines Handlers

`SA_RESTART` ... restartbare Systemcalls nach dem Signalhandler restarten
`SA_NODEFER` ... Das Signal das die Signalbehandlung ausgelöst hat, auch in der Ausführung des Signalhandlers zulassen.
`SA_RESETHAND` ... nachdem der Signalhandler aufgerufen wurde, die Default-Action wiederherstellen

Signale

Beispiel

Eine Handlerfunktion für SIGINT setzen

```
void signalhandler(int signal) {
  char *msg = „Handle Signal“;
  write(STDOUT_FILENO, msg, strlen(msg));
}



 ...
struct sigaction sigAction;

sigAction.sa_handler = signalhandler; // Handler-Funktion setzen
sigemptyset(&sigAction.sa_mask); // keine Signale blockieren
sigaddset (&sigAction.sa_mask, SIGINT); // Auslösersignal blockieren
sigAction.sa_flags = SA_RESTART;

if (sigaction (SIGINT, &sigAction, NULL) < 0) // Handler registrieren
   ...
```

Signale

Anforderungen an Signal-Handler

In einem Signalhandler dürfen nur Funktionen aus der Standardbibliothek aufgerufen werden, die `async-signal-safe` sind.
`stdio`-Funktionen gehören nicht dazu!

Eine Liste jener Funktionen, die `async-signal-safe` sind, erhält man mit

```
man signal-safety
```

Der Signalhandler selbst muss in Hinsicht auf die Verwendung von globalen Variablen reentrant sein.

Signale

Signale senden

Dem eigenen Prozess ein Signal senden - raise()

```
#include <signal.h>
int raise(int sig_nr);
```

Ein Signal an einen anderen Prozess senden - kill()

```
int kill(pid_t pid, int sig)
```

`pid > 0` ... das Signal wird an den Prozess mit der Prozess-ID pid gesendet.
`pid == -1` ... das Signal wird an alle Prozesse mit Ausnahme von init gesendet (nur Superuser).

Signale

Übung

1) Implementieren sie eine Funktion in der Art

```
int setSignalHandler (int signal, void (*signalhandler)(int)) { … }
```

welche den Funktionsparameter `signalhandler` als Signalhandler für das Signal `signal` einrichtet.

Verwenden sie dazu die vorher besprochenen Funktionen `sigemptyset()`, `sigaddset()`, `sigaction()` und die Struktur `sigaction`.

Signale

Übung

2) Schreiben sie ein Programm, das folgende Schritte umsetzt:

- Einen Signalhandler für das Signal `SIGINT` mit der Funktion `setSignalHandler()` einrichten.
  Der Signalhandler soll den Text `"SIGINT aufgetreten."` auf `stdout` ausgeben.
  Verwenden sie zur Ausgabe im Signalhandler die sichere Low-level Funktion `write(STDOUT_FILENO,…)`!

- Nach dem Einrichten des Signalhandlers soll mit `sleep(2)` etwas gewartet und anschließend mit `raise()` das Signal `SIGINT` an den Prozess gesendet werden. Ungefähr so:

```
int main(int argc, char* argv[]){
  setSignalHandler(SIGINT, ...
  sleep(2);
  raise ...
  exit(EXIT_SUCCESS);
}
```

Filesystem

Informationen über Dateien und Verzeichnisse

Bisher haben wird mit Standard-I/O und Low-Level I/O aus Dateien gelesen, bzw. darauf geschrieben.

Im folgenden werden die zusätzlichen Informationen die uns das Filesystem bietet untersucht:

- Auslesen von Attributen einer Datei
- Auslesen der Berechtigungen einer Datei
- Prüfen der aktuellen Zugriffsrechte
- Durchsuchen eine Verzeichnishierarchie

Filesystem

stat - Informationen über Dateien und Verzeichnisse

Die Funktion `stat()` liefert Informationen über ein Element des Dateisystem (Datei, Verzeichnis, Link ...).

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int stat(const char *pathname, struct stat *statbuf);
```

`pathname` ... Pfad der Datei
`statbuf` ... Struktur mit den Ergebnissen der Abfrage

Returnvalue: 0 if OK, -1 Error

Filesystem

Informationen über Dateien und Verzeichnisse

Die Felder der `struct stat` enthalten Informationen über den Filesystem-Eintrag.

Wesentliche Felder:

`st_size`	Filesize in Bytes
`st_mode`	File-Typ (Datei, Link, Verzeichnis ...) und Zugriffsrechte
`st_mtime`	Modification Time

...

Für die Abfrage des Dateityps im Feld `st_mode` sind Macros definiert:

```
struct stat s;
```

`S_ISREG(s.st_mode)` ... reguläre Datei
`S_ISDIR(s.st_mode)` ... Verzeichnis
`S_ISLNK(s.st_mode)` ... Link

usw.

Filesystem

Dateiberechtigungen

Im Feld `st_mode` sind auch die Dateiberechtigungen kodiert, diese können mit dem Access-Mode Macros aus Low-Level I/O abgefragt werden:

```
struct stat s;
if (s.st_mode & S_IRUSR) // kann der Owner Lesen?
```

Beispiel:

```
struct stat s;
char *path = "/home/user";

if (stat(path, &s) == 0) {
  if (S_ISREG(s.stmode)) printf("%s is a regular file.", path);
  else if (S_ISDIR(s.st_mode) printf("%s is a directory.", path);

  if (s.st_mode & S_IRUSR && s.st_mode & S_IWUSR)
    printf("%s is readable and writeble for the owner.", path)
}
else
  perror(path);
```

Filesystem

Zugriff prüfen

Um zu prüfen, welche Zugriffsrechte der aktuelle Prozess auf eine Datei/ein Verzeichnis hat, kann die Funktion `access()` zu verwendet werden.

```
#include <unistd.h>

int access(const char *pathname, int mode);
```

`pathname` … Pfad der Datei

`mode`:

- `F_OK` … test if file exists
- `R_OK, W_OK, X_OK` …

  “test if file exists and the calling process has read, write or execute permission on file“ (mit bitweisem ODER verknüpfen)

Beispiel:

```
if (access("/home/user", W_OK|R_OK|X_OK) == 0) {
…
}
```

Filesystem

Verzeichnisse

Ein Verzeichnis kann Dateien und Verzeichnisse enthalten. Die Funktion `readdir()` ermöglicht es, auf alle in einem Verzeichnis enthaltenen Einträge zuzugreifen.

```
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char *name);

struct dirent *readdir(DIR *dirp);

int closedir(DIR *dirp);


struct dirent {
  ino_t          d_ino;       /* Inode number */
  ...
  char           d_name[256]; /* Null-terminated filename */
};
```

`d_name` ist immer relativ zum abgefragten Verzeichnis!

Filesystem

Beispiel – alle Einträge im aktuellen Verzeichnis ausgeben

```
#include <sys/types.h>
#include <dirent.h>
#include <stdio.h>

int main() {
  DIR *dir;
  struct dirent *de;

  if ((dir = opendir("./")) ≠ NULL) {
    while(de = readdir(dir))
      puts(de→d_name);
    closedir(dir);
  }
  else
    perror ("directory ./");
}
```

Zeit

Zeit abfragen

Die aktuelle Zeit kann mit dem Aufruf von `time()` abgefragt werden.

```
#include <time.h>
time_t time(NULL);
```

Aufruf liefert die Anzahl von Sekunden seit „The Epoch“ zurück, das sind die vergangenen Sekunden seit

`1970-01-01 00:00:00 +0000 (UTC)`

Im Fehlerfall wird -1 als Rückgabewert geliefert.

Umwandlung in einen String mit einem Default-Format:

```
char *ctime(const time_t *timep);
```

Beispiel: `Wed Apr 24 08:13:47 2024`

Zeit

Zeit umwandeln

Umwandlung in eine Struktur unter Berücksichtigung der aktuell eingestellten Zeitzone:

```
struct tm *localtime(const time_t *timep);
```

```
struct tm {
    int tm_sec;    /* Seconds (0-60) */
    int tm_min;    /* Minutes (0-59) */
    int tm_hour;   /* Hours (0-23) */
    int tm_mday;   /* Day of the month (1-31) */
    int tm_mon;    /* Month (0-11) */
    int tm_year;   /* Year - 1900 */
    int tm_wday;   /* Day of the week (0-6, Sunday = 0) */
    int tm_yday;   /* Day in the year (0-365, 1 Jan = 0) */
    int tm_isdst;  /* Daylight saving time */
};
```

Returnvalue: `NULL` im Fehlerfall

Zeit

Zeit formatieren

Die Funktion `strftime` wandelt einen Zeitwert vom Typ `struct tm` in einen formatierten String um.

```
#include <time.h>
size_t strftime(char *buffer, size_t size, const char *format,
                const struct tm *tm);
```

Returnvalue: Anzahl geschriebener Bytes oder 0 im Fehlerfall

Ausgewählte Format-Specifier:

- `%a` … Wochentag
- `%d` … Tag des Monats (1-31)
- `%b` … abgekürzter Name des Monats
- `%y` … Jahr, abgekürzt
- `%T` … Zeit (24h Format)
- `%z` … Timezone-Offset von UTC

Zeit

Beispiel

Das folgende Beispiel fragt die aktuelle Zeit ab und gibt sie im Default-Format aus.

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

int main() {
  time_t currentTime;

  if ((currentTime = time(NULL)) ⩾ 0)
    printf("Time: %s\n", ctime(&currentTime));
  else
    perror("Time");
}
```

Verzeichnisse

Übung

Schreiben sie ein Programm `DIR` welches das aktuelle Verzeichnis ausliest und alle darin enthaltenen Einträge auf `stdout` ausgibt.
Verwenden sie folgendes Format:

```
Dateiname                                          Typ Permission Zeit
```

Dateiname: padded auf z.B: 50 Zeichen links ausgerichtet.
Typ: `'F'` Datei, `'D'` Verzeichnis, `'L'` Link
Permission: Wenn für den Besitzer ausführbar dann `'X'` sonst leer.
Zeit: Modification time

Beispiel:

```
popen.c                                            F   Mon Jan 15 15:11:29 2024
..                                                 D X Tue Apr 23 20:29:16 2024
.                                                  D X Wed Apr 24 08:23:06 2024
DIR                                                F X Wed Apr 24 08:23:06 2024
dir.c                                              F   Wed Apr 24 08:23:02 2024
signal.c                                           F   Mon Jan 15 15:11:29 2024
pipe.c                                             F   Mon Jan 15 15:11:29 2024
```