Inhaltsverzeichnis

1. OS: posix

1.1. Ziele: Betriebssystem - Interface

☑ Wir wollen hier die im **POSIX** (http://de.wikipedia.org/wiki/POSIX)
(**Portable Operating System Interface**) beschriebenen Methoden und Konzepte zum

- ☐ **Multitasking** (Nebenläufigkeit) und der
- ☐ **Prozesskommunikation/-Synchronisation** (IPC=InterProcessCommunication) üben.

☑ Im Detail werden folg. Themen/Begriffe besprochen:

- ☐ Signale (signal(),alarm(), ...)
- ☐ Prozesse (fork(), exec(), wait(), ...)
- ☐ Kommunikation (pipe(), popen(),fdopen(), streams)
- ☐ Synchronisation: Lock-Files

☑ Workshop:

- ☐ http-server mit fork()
- ☐ Parallelles Sortieren (MergeSort)

☑ Quellen:

- ☐ https://computing.llnl.gov/tutorials/pthreads/

1.2. Was bedeutet POSIX

(*http://de.wikipedia.org/wiki/POSIX*)

Das **Portable Operating System Interface** (**POSIX** ['pɒzɪks]) ist ein

gemeinsam von der IEEE und der Open Group

für *Unix* entwickeltes standardisiertes Application Programming Interface, das die Schnittstelle zwischen Applikation und demBetriebssystem darstellt.

Die internationale Norm trägt die Bezeichnung ISO/IEC/IEEE 9945.

1.2.1. POSIX für Windows User

http://www.dogsbodynet.com/openr/install_cygwin.html

1.3. alarm, signal, kill – Signale versenden (4)

Signale werden zur einfachen **Kommunikation zwischen Programmen** verwendet.

Mögliche Anwendungen sind:

☑ auf **Programmabbruch durch Ctrl-C** reagieren können

☑ einem Programm **während seiner Laufzeit mitteilen**, es möge seine Konfiguration neu auslesen

☑ auf einen **Timeout** reagieren können

…

☑ Hinweis:
alarm ... set an alarm clock for delivery of a signal
signal … set signalhandler routine for a sig-number

man 3 alarm
http://www.manpagez.com/man/3/Signal/

1.3.1. Beispiel: demo_selfalarm.c

- Das folgende Programm bricht automatisch nach 10 Sekunden ab (siehe alarm(10)).
- Da das Programm ein Zeichen (argv[1]) ausgibt, sieht man das Time-Slicing-Verfahren des Schedulers, wenn man das Programm 2 mal startet.

```
// demo_selfalarm.c

#include <stdio.h>
#include <stdlib.h>

#include <unistd.h>

int main(int argc, char * argv[]){
    if (argc==1){
        printf("usage: selfalarm.exe #"); exit(1);
    }

    alarm(10);    //nach 10 sekunden, bricht das Programm ab

    while(1)
        printf("%s", argv[1]);

     return 0;
}

// gcc demo_selfalarm.c -o demo_selfalarm.exe
// starte mit:
// ./demo_selfalarm.exe 1 & ./demo_selfalarm.exe 0
```

Starten Sie das Programm selfalarm.exe 2× gleichzeitig mit

```
./demo_selfalarm.exe 1 & ./demo_selfalarm.exe 0
```

Die Eingabe "x & y" startet die beiden Kommandos x und y gleichzeitig.

Signal-Bearbeitungsroutine (signal-handler function):

Abgesehen von SIGSTOP und SIGKILL kann man das Standardverhalten jedes Signals durch Installation einer Signal-Bearbeitungsroutine anpassen.

Eine **Signal-Bearbeitungsroutine** ist eine Funktion, die vom Programmierer implementiert wird und jedes Mal aufgerufen wird, wenn der Prozess ein entsprechendes Signal empfängt.

Eine Funktion, die als Signal-Bearbeitungsroutine fungieren soll, muss einen einzigen **Parameter vom Typ int und einen void-Ruckgabetyp** definieren. Wenn ein Prozess ein Signal empfängt, wird die Signal-Bearbeitungsroutine mit der Kennnummer des Signals als Argument aufgerufen.

```
void (*func)(int signal_number);
```

Signale finden Sie in der Header-Datei **/usr/include/bits/signum.h**

Name	Wert	Funktion
SIGHUP	1	Logoff
SIGINT	2	Benutzer-Interrupt (ausgelöst durch [Strg]+[C])
SIGQUIT	3	Benutzeraufforderung zum Beenden (ausgelöst durch [Strg]+[\])
SIGFPE	8	Fließkommafehler, beispielsweise Null-Division
SIGKILL	9	Prozess killen
SIGUSR1	10	Benutzerdefiniertes Signal
SIGSEGV	11	Prozess hat versucht, auf Speicher zuzugreifen, der ihm nicht zugewiesen war
SIGUSR2	12	Weiteres benutzerdefiniertes Signal
SIGALRM	14	Timer (Zeitgeber), der mit der Funktion `alarm()` gesetzt wurde, ist abgelaufen
SIGTERM	15	Aufforderung zum Beenden
SIGCHLD	17	Kindprozess wird aufgefordert, sich zu beenden
SIGCONT	18	Nach einem `SIGSTOP`- oder `SIGTSTP`-Signal fortfahren
SIGSTOP	19	Den Prozess anhalten
SIGTSTP	20	Prozess suspendiert, ausgelöst durch [Strg)+[Z]

1.3.2. Aufgabe: id-code.c (SIGALRM)

Wir wollen nun, wenn das Signal SIGALRM erzeugt wird, das Programm nicht abbrechen lassen, sondern eine eigene Signal-Bearbeitungs-Funktion aufrufen lassen.

Aufgabe: Bringen Sie das folgende Programm zum Laufen und erklären Sie seine Funktion.

```
/* Datei: id-code.c Hofmann Anton
 *
 * Demo: Zeiger auf Funktionen und Interprocess Communication
 * Read with timeout
 */
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

int toolong;
int count;
```

```
void wakeup(int signum){
    toolong=1;
    count++;

    #ifdef DEBUG
    printf("wakeup was called\n"); fflush(stdout);
    #endif

    alarm(10);
}

int main(){
    int idcode;

    //Pointer,um die Adresse der 'alten' Alarmfunktion zu speichern.
    void (*alrm_func)();

    while(1) {
        printf("Wie lautet Ihr ID-CODE ? ");
        toolong= 0;

          // setze das Alarmsignal auf wakeup() und
          //speichere den alten Zeiger alrm_func.
        alrm_func= signal(?????????, ???????????);

        /* setze den Alarm-Timer */
        alarm(10);

        /* lies ID-CODE */
        scanf("%d", &idcode);

        /* ID-CODE wurde ohne timeout eingeben->verlasse Schleife */
        if (toolong == 0) break;

        if (count == 1) /* Antwort auf ersten timeout */ {
            printf("\nIhren ID-CODE finden Sie auf Ihrer ID-KARTE.\n");
            fflush(stdout);
        }
        else {
            printf("\nFragen Sie im Sekretariat nach einer neuen ");
            printf("ID-KARTE, \nfalls sie verloren gegangen ist.\n");
            exit(1); /* PROGRAMMABBRUCH */
        }
    } /* end_while */


    /* setze das Alarmsignal wieder zurueck*/
    signal(???????????, ?????????????);
    alarm(0);

    printf("\nID-CODE: %d\n", idcode);

    return 0;

}
```

☑ man signal

```
#include <signal.h>

       void (*signal(int sig, void (*func)(int)))(int);
```

☑ Hinweis:
gcc -DDEBUG .... bewirkt, dass auch die Anweisungen zwischen #define DEBUG .... #endif übersetzt werden

1.3.3. Aufgabe: wecker.c - Wecker mit Text (SIGALRM)

Wir wollen einen "Wecker mit Text" erstellen. Man startet diesen und übergibt ihm eine Zeit in Sekunden und einen Text. Wenn diese Zeit abgelaufen ist, wird vom Wecker ein Signal (SIGALRM) abgesetzt. Standardmäßig wird dann das laufende Programm beendet. Wir wollen aber, dass das Programm vor dem Ende noch den Text ausgibt.

Aufruf: ./wecker.exe 10 "Kaffeepause mit Bob" &

1.3.4. Aufgabe: control-c.c (SIGINT)

Schreiben Sie ein Programm, das beim erstmaligen Drücken von Ctrl-C den Text "Control-c" am Bildschirm ausgibt. Beim zweiten Drücken von Ctrl-C soll das Programm beendet werden. (exit(1)).
Ansonsten gibt das Programm in einer Endlos-Schleife „Hello World!“ aus.

1.3.5. Aufgabe: kill-usr1-usr2.c (SIGUSR1, SIGUSR2)

Mit dem Shell-Kommando kill (man kill) kann man von der Shell aus Programmen Signale schicken. Oft werden hier die Signale USR1 bzw. USR2 verwendet. Hier ein Beispiel:

Aufruf:

```
$> ./kill-usr1-usr2.exe &
[1]     4720

$> kill -USR1  4720
received SIGUSR1


$> kill -USR2  4720
received SIGUSR2


$> kill 4720
```

Bringen Sie das folgende Programm kill-usr1-usr2.c zum Laufen:

```
/* Datei: kill-usr1-usr2.c
 * demo signal()
 * gcc kill-usr1-usr2.c -o kill-usr1-usr2.exe
 * aufruf: ./kill-usr1-usr2.exe &
 * [1]     4720
 * $ kill -USR1  4720
 * $ kill -USR2  4720
 * $ kill  4720
```

```
 */

#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <errno.h>


int main(){

      // ENTER CODE

      for (;;)
            pause();
            // pause() forces a process to pause until a signal is received

      return 0;
}
```

☑ Hinweis*:*
siehe auch man 2 kill

1.3.6. Aufgabe: minishell (SIGUSR1)

Ordner: minishell

Schreiben Sie das Programm *t_minish.c (o_strlist.o, o_strlist.h)* derart um, dass

1. bei Auftreten des Signals USR1 die zur Zeit aktive Liste von Befehlen verworfen wird und
2. (falls vorhanden) die Datei default.msh eingelesen wird.