Schmid Kevin
1710418023

Interprozesskommunikation

Signale und Pipes

1 Signale als Kommunikationsmittel

1.1 Signalarten

a. Geben Sie `kill -l` in der Konsole ein und listen Sie alle Signale auf

```
kevin@Kevin-XPS15:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

b. Untersuchen Sie mit man die Syntax der Systemfunktion `signal`

```
SIGNAL(2)                  Linux Programmer's Manual                 SIGNAL(2)

NAME
       signal - ANSI C signal handling

SYNOPSIS
       #include <signal.h>

       typedef void (*sighandler_t)(int);

       sighandler_t signal(int signum, sighandler_t handler);
```

c. Listen Sie einige Signale als Beispiel für folgende Anwendungen

i. Welche Signale treten bei Programmfehlern auf?

SIGILL, SIGTRAP, SIGABRT, SIGBUS, SIGFPE, SIGSEGV, SIGSYS, SIGEMT, SIGIOT, SIGPIPE, SIGLOST, SIGXCPU, SIGXFSZ

ii. Mit welchen Signalen können Prozesse beendet werden?

SIGHUP, SIGINT, SIGQUIT, SIGKILL, SIGTERM

iii. Welche Signale unterstützen die Prozesskontrolle?

SIGCHLD, SIGCONT, SIGSTOP, SIGTSTP, SIGTTIN, SIGTTOU, SIGCLD

1.2 Prozesskommunikation

Mit der Funktion `kill` kann ein Prozess einem anderen Prozess ein Signal schicken und diesen beenden. Führen Sie das Programm `alarm.c` aus

a. Untersuchen und dokumentieren Sie die Bedeutung der Systemfunktion `alarm`

alarm() sorgt dafür, dass ein SIGALRM-Signal innerhalb von `seconds` Sekunden an den Aufrufer-Prozess übermittelt wird.
Wenn `seconds` gleich Null ist, wird jeder anstehende Alarm abgebrochen.

b. Listen Sie mit `ps x` alle Prozesse im System

```
10427 pts/2    Ss     0:00 /bin/bash
10461 pts/2    R+     0:15 ./a.out
10469 pts/0    R+     0:00 ps x
kevin@Kevin-XPS15:~/FH/S4/sysprog/ue4$ _
```

```
Terminal
kevin@Kevin-XPS15:~/FH/S4/sysprog/ue4$ ./a.out
_
```

c. Beenden Sie mit `kill` den alarm.c Prozess. Welches Signal wird dabei gesendet?

SIGKILL

```
10427 pts/2    Ss     0:00 /bin/bash
10461 pts/2    R+     0:15 ./a.out
10469 pts/0    R+     0:00 ps x
kevin@Kevin-XPS15:~/FH/S4/sysprog/ue4$ kill -9 10461
kevin@Kevin-XPS15:~/FH/S4/sysprog/ue4$ _
```

```
Terminal
kevin@Kevin-XPS15:~/FH/S4/sysprog/ue4$ ./a.out
Killed
kevin@Kevin-XPS15:~/FH/S4/sysprog/ue4$ _
```

1.3 Abnormales Prozessende signalisieren

In `abort.c` signalisiert das Signal SIGABRT mit der Funktion `abort()`, dass ein Programm abnormal beendet wurde:

a. Untersuchen und dokumentieren Sie die Systemfunktion `abort()`

Die abort() entsperrt zuerst das SIGABRT-Signal und löst dann dieses Signal für den Aufruf-Prozess aus. Dies führt zu einem vorzeitigen Abbruch des Prozesses, es sei denn, das SIGABRT-Signal wird abgefangen und der Signalhandler kehrt nicht zurück.

b. Ergänzen Sie das Programm um eine Systemfunktion, damit das Programm trotz `abort()` funktionieren.

```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <setjmp.h>

jmp_buf env;

void sigfunc(int sig){
  if(sig==SIGABRT)
    printf("Demonstration von SIGABRT ausgeloest mit abort()\n");
  longjmp(env, 1);
}

int main(){
  signal(SIGABRT,sigfunc);
  if(setjmp(env) == 0) {
    abort();
  }
}
```

```
1 / 1
kevin@Kevin-XPS15:~/FH/S4/sysprog/ue4$ ./a.out
Demonstration von SIGABRT ausgeloest mit abort()
kevin@Kevin-XPS15:~/FH/S4/sysprog/ue4$ _
```

1.4 Normales Prozessende signalisieren

Ergänzen Sie den notwendigen Code in signal.c, damit Strg C funktionieren kann:

a. Mit einem möglichen Abbruch im if-Teil mit SIGINT. Welche Bedeutung hat SIGINT und durch was könnte es ersetzt werden?

SIGINT = Strg + C
SIGQUIT = Strg + \
SIGTSTP = Strg + Z

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

void sigfunc(int sig){
 char c;

if(sig != SIGINT)
   return;
 else{
    printf("\nWollen sie das Programm beenden (j/n) : ");
    while((c=getchar()) == 'j')
    exit (0);
   }
}

void main(void){
int i;

signal(SIGINT, sigfunc);

 while(1){
   printf("Die Endlosschleife koennen sie mit STRG-C beenden");
   for(i=0;i<=48;i++)
      printf("\b");
   }
}
```

```
1 / 1

Terminal

kevin@Kevin-XPS15:~/FH/S4/sysprog/ue4$ ./a.out
Die Endlosschleife koDie Endlosschleife koennen sie mit STRG-C beenden
Wollen sie das Programm beenden (j/n) : j
kevin@Kevin-XPS15:~/FH/S4/sysprog/ue4$ _
```

b. Wie ist prinzipiell die Funktionsweise von signal(SIGINT,sigfunc)in signal.h?

Das SIGINT-Signal wird mit der Tastatureingabe Strg + C vom Terminal an den Prozess gesendet, wenn ein Benutzer den Prozess unterbrechen möchte.

1.5 Rechenfehler signalisieren

Das einfache Programm division.c dividiert zwei Zahlen. Erweitern Sie es, dass

Schmid Kevin
1710418023

d. Division durch Null signalisiert und eine Nachricht ausgegeben wird

e. STRG+C ignoriert wird

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

void sigfunc(int sig){
 if(sig != SIGFPE)
  return;
 else {
  printf("durch 0 tun wir nicht \n");
  abort();
 }
}

int main(){
 int wert1,wert2;

 signal(SIGFPE, sigfunc); // TODO: Floating Point Error konfigurieren
 signal(SIGINT, sigfunc); // TODO: STRG - C Signal ignorieren

 printf("Programm zum Dividieren von Zahlen!\n");
 printf("Zahl eingeben : ");
 scanf("%d",&wert1);

 while(1){
  printf("geteilt durch > ");
  scanf("%d",&wert2);
  printf("Gesamt = %f\n",wert1/=wert2);
 }
}
```

```
1 / 1

Terminal

kevin@Kevin-XPS15:~/FH/S4/sysprog/ue4$ ./a.out
Programm zum Dividieren von Zahlen!
Zahl eingeben : 12
geteilt durch > 0
durch 0 tun wir nicht
Aborted (core dumped)
kevin@Kevin-XPS15:~/FH/S4/sysprog/ue4$ _
```

abort() wurde eingebaut, da das SIGFPE in einer Endlosschleife gesendet wird.

Schmid Kevin
1710418023

1.6 Kontrollfragen Signale

1. Wieviel verschiedene Signale gibt es unter Linux? Wie ist die Unterteilung mit Echtzeitsystemen (Realtime Time, RT)

Insgesamt gibt es 64 unterscheidbare Signale, wobei die Signale 32 bis 64 für Echtzeitsysteme gedacht sind.

2. Welche drei Möglichkeiten gibt es, um auf ein Signal zu reagieren?

- Eintragen einer selbst geschriebenen Funktion
- Ignorieren des Signals
- die voreingestellte Standardaktion verwenden

3. Wenn ein Signal auftritt, wo wird dieses hinterlegt bzw. gespeichert?

Es wird als entsprechender Prozesstabelleneintrag gespeichert.

4. Was macht ein Signalhandler prinzipiell?

Er steuer wie mit einem eingehenden Signal umgegangen werden soll. (Siehe Frage 2)

5. In welche drei Klassen werden die Signale unterteilt?

Systemsignale, Gerätesignale und benutzerdefinierte Signale

6. Signale mit fork und exec: Der neue Prozess, der mit fork erzeugt wird, erbt auch alle Signalhandler des aufrufenden Prozesses. Was passiert aber durch eine Überlagerung eines Prozesses mit der exec-Funktion?

Durch Überlagerung eines Prozesses mit den exec-Funktion, verlieren auch die eingerichteten Signalhandler Ihre Gültigkeit. Alle Signale werden wieder auf Ihren Ursprünglichen Zustand (default-Zustand) gesetzt und reagieren auch so.