1. Interrupts und Signale

In dieser Übung lernen Sie den Signal-Mechanismus kennen, über den Prozesse Hinweise „von außen“ (von anderen Prozessen oder vom Betriebssystem) erhalten und darauf reagieren können.

Sowohl die Signalbehandlung in Prozessen als auch die Interrupt-Behandlung im Betriebssystem sind Mechanismen, die *asynchrone* Ereignisse handhaben, indem sie spezialisierte Funktionen (die Signal- oder Interrupt-*Handler*) ausführen, die vorab registriert werden.

In der Signalbehandlung in Prozessen geht es darum, dass ein Betriebssystem oder ein anderes Programm ein Signal an einen laufenden Prozess sendet, um diesem Prozess mitzuteilen, dass ein bestimmtes Ereignis eingetreten ist. Beispiele für Signale sind `SIGINT` (Interrupt-Signal, typischerweise durch Drücken von Strg+C ausgelöst) und `SIGTERM` (Beenden-Signal). Der Prozess kann für jedes Signal einen Signal-Handler registrieren, der die spezielle Aufgabe übernimmt, wenn das Signal eintrifft. Dieser Handler wird in der Regel durch die Funktion `signal()` oder `sigaction()` registriert. Wenn das Signal eintrifft, wird die normale Ausführung des Prozesses unterbrochen, der registrierte Handler wird ausgeführt, und nach dessen Abschluss kehrt der Prozess zu seiner normalen Ausführung zurück.

Ähnlich funktioniert die Interrupt-Behandlung im Betriebssystem auf der Ebene der Hardware. Ein Interrupt ist ein Signal von einem Hardwaregerät, das die CPU darüber informiert, dass ein bestimmtes Ereignis, wie z. B. ein Tastendruck oder eine Netzwerkaktivität, eingetreten ist. Das Betriebssystem registriert für jeden Interrupt-Typ einen Interrupt-Handler, der eine spezielle Routine ausführt, wenn der Interrupt auftritt. Diese Handler werden unter Linux direkt vom Kernel oder durch einen als Modul geladenen Treiber registriert. Wenn ein Interrupt eintritt, wird die aktuelle CPU-Ausführung unterbrochen, der entsprechende Interrupt-Handler wird ausgeführt, und anschließend wird die unterbrochene Arbeit fortgesetzt.

Sowohl bei der Signal- als auch bei der Interrupt-Behandlung handelt es sich um Mechanismen zur *asynchronen* Ereignisverarbeitung, wobei spezielle Handler für die Verarbeitung registriert werden. Beide Systeme ermöglichen es, dass bestimmte Aufgaben sofort und direkt bearbeitet werden können, wenn entsprechende Ereignisse eintreten, und nach der Fertigstellung wird der unterbrochene Code fortgesetzt.

Das folgende Programm `strg-c.c` registriert einen Handler `int_handler()`, der beim Drücken von Strg-C aufgerufen wird.

strg-c.c

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

int pid = -1;

void int_handler (int sig) {
  printf ("\n[%d] Strg-C gedrueckt...\n", pid);
}

int main () {
  signal (SIGINT, int_handler);   // Handler eintragen
  pid = getpid();
  for (;;) {
    printf ("[%d] Idle...\n", pid);
    sleep (1);
  }
}
```

a) Starten Sie einen Ubuntu-Container, wechseln Sie in das Arbeitsverzeichnis (`/realworld`), laden Sie das Code-Archiv für diese Übung herunter, entpacken Sie es, und wechseln Sie in den Ordner `ue05`:

```
cd /realworld
wget swf.hgesser.de/bs-b1/prakt/bs1-ue05.zip
unzip bs1-ue05.zip
cd ue05
```

Windows-Anwender: Wenn Sie die PowerShell unter Windows verwenden, geben Sie jetzt das folgende Kommando ein:

```
stty susp ^E
```

(Damit ändern Sie in der Bash-Shell die Tastenkombination zum Suspendieren eines laufenden Prozess von Strg-Z auf Strg-E – das ist nötig, weil die PowerShell Strg-Z nicht korrekt an die in Docker laufende Bash weiter leiten kann.)

b) Kompilieren Sie das Programm `strg-c.c` und starten Sie es:

```
gcc -o strg-c strg-c.c
./strg-c
```

c) Versuchen Sie, das Programm mit Strg-C zu beenden – was passiert? Warum?

d) Drücken Sie Strg-Z (bzw. unter Windows: Strg-E), um den Prozess zu suspendieren, und geben Sie dann

```
killall -SIGTERM strg-c
fg
```

ein, um dem Prozess das Signal `SIGTERM` (in der Regel gilt `SIGTERM = 15`) zu schicken und ihn im Vordergrund (foreground, fg) fortzusetzen (also die Suspendierung aufzuheben).

(Der Befehl `killall` schickt das hinter dem „-“ angegebene Signal an alle Prozesse, die den als letztes Argument verwendeten Namen haben; hier also `SIGTERM` an alle Prozesse, die `strg-c` heißen.)

e) Ändern Sie den Quellcode so ab, dass auch die in **d)** beschriebene Vorgehensweise das Programm nicht beenden kann. Sie werden dazu einen zusätzlichen Handler brauchen, der auch eingetragen werden muss.

f) Statt `SIGTERM` können Sie auch das Signal `SIGKILL = 9` verwenden, das „aggressiver“ ist und Prozesse in jedem Fall unterbricht. Gehen Sie wie in **d)** vor, aber mit `SIGKILL` statt `SIGTERM`, und prüfen Sie, dass Sie den `strg-c`-Prozess erfolgreich beendet haben.

g) Das Signal `SIGKILL` kann nicht (!) auf gleiche Weise mit einem Handler abgefangen werden. Versuchen Sie es: Ergänzen und aktivieren Sie eine dritte Handler-Funktion und überzeugen Sie sich davon, dass Sie trotzdem das Programm über das `SIGKILL`-Signal abbrechen können.

h) Was passiert, wenn Sie ein Signal verschicken, für das kein Signal-Handler eingetragen wurde? Lassen Sie sich, um mögliche Signale zu entdecken, mit

```
kill -l
```

eine Liste der bekannten Signale ausgeben.

2. Division durch 0

Das folgende Programm `div0.c` teilt durch 0, was zunächst einen Prozessorfehler (*Division Error*[1]) auslöst. Der zugehörige Fault-Handler im Betriebssystem reicht das Problem in Form eines Signals an den Prozess weiter.

div0.c

```
int main () {
  int a=1; int b=0;
  int c=a/b;  // Division durch 0
  printf ("1/0 = %d\n", c);
}
```

a) Testen Sie das Programm (es liegt auch im Ordner `ue05`).

b) Fangen Sie das Signal ab. Sie können herausfinden, welches Signal Linux an den Prozess schicken, indem Sie die Ausführung mit `strace` überwachen:

```
strace -e signal ./div0
```

In Ihrer angepassten Programmversion führt die Division nun nicht zum Programmabbruch – das neue Programmverhalten ist aber unerwartet. Überlegen Sie, auch im Austausch mit anderen Studierenden, woran das liegen könnte.

3. Signal-Ping-Pong

Betrachten Sie das Programm `signal-pingpong.c` auf der folgenden Seite. (Auch diese Datei liegt in `ue05`.) Es erzeugt einen Kindprozess, und die beiden Prozesse nutzen dann den Signal-Mechanismus, um sich gegenseitig im Ping-Pong-Stil Signale zuzusenden. Dabei muss einer den Anfang machen: Das übernimmt der Vaterprozess.

Zum Verständnis:

```
signal (Nummer, Handler);
```

trägt (wie in Aufgabe **1** und **2**) im laufenden Prozess einen Signal-Handler für die angegebene Signalnummer ein, und dieser wird automatisch aufgerufen, wenn ein Prozess mit

```
kill (ProzessID, Nummer);
```

dieses Signal an den Prozess schickt.

a) Testen Sie das Programm: Sie können es mit

```
gcc -o signal-pingpong signal-pingpong.c
./signal-pingpong
```

übersetzen und starten.

b) Betrachten Sie die Ausgabe und finden Sie heraus, warum

– die Länge der „+“- und „–“-Zeilen zunächst wächst und dann wieder schrumpft,

– die „+“- und „–“-Zeilen immer abwechselnd erscheinen.

c) Warum wird am Schluss nur einer der beiden Prozesse beendet? Wie sorgen Sie dafür, dass beide Prozesse beendet werden?

1 https://wiki.osdev.org/Exceptions#Division_Error

signal-pingpong.c

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

int meine_pid, partner_pid, pid;
int counter = 1;
char symbol;

void show_symbols (char sym, int count, char *string) {
  int i;
  sleep (1);
  for (i = 0;  i < 3*count;  i++)
    printf ("%c", sym);
  printf ("%s", string);
}

void usr1_handler_b (int sig) {
  counter--;
  if (counter < 1)
    exit (0);
  show_symbols (symbol, counter, "\n");
  kill (partner_pid, SIGUSR1);
}

void usr1_handler_a (int sig) {
  counter++;
  if (counter > 4)
    signal (SIGUSR1, usr1_handler_b);                // neuen Handler eintragen
  show_symbols (symbol, counter, "\n");
  kill (partner_pid, SIGUSR1);
}

int main () {
  char message[20];
  printf ("Signal-Handler-Ping-Pong\n");
  signal (SIGUSR1, usr1_handler_a);                  // Handler eintragen
  pid = fork ();
  meine_pid = getpid ();
  if (pid == 0) {
    partner_pid = getppid ();                        // ID vom Vater
    symbol = '-';
  } else {
    partner_pid = pid;
    symbol = '+';
  }
  printf ("Prozess %d: Mein Partner ist Prozess %d\n", meine_pid, partner_pid);
  if (pid != 0) {
    // starte Ping-Pong
    sleep (5); kill (partner_pid, SIGUSR1);
  }

  sprintf (message, " main() - %d\n", meine_pid);    // baut String zusammen
  for (;;) {
    if (counter <= 1)
      show_symbols ('*', counter, message);
    sleep (5);
  }
}
```