## 8.4 Sonstiges/Alarm-Timer

### 8.4.1 Zeitschaltuhr

Mit der Funktion `alarm()` kann man bei Linux und verwandten Betriebssystemen eine Zeitschaltuhr einrichten, die nach Ablauf der angegebenen Zahl von Sekunden ein Signal auslöst. Dieses Signal kann nun von einem geeigneten Signalhandler verarbeitet werden und das Programm beeinflussen.

Danach kann man die Zeitschaltuhr wieder neu loslaufen lassen. Möchte man das sofort und immer wieder, kann man gleich in den Signalhandler wieder einen `alarm()`-Aufruf setzen. Ein Beispiel zeigt `src/alarm.c`:

```c
#include <unistd.h> /* fuer alarm()                 */
#include <signal.h> /* fuer signal() und SIGALRM */
#include <stdio.h>
int anz=0;
/************************************************/
void alarmfunktion(int signr)
{
    ++anz;
    alarm(3); /* Uhr wieder einschalten */
}
/************************************************/
int main(void)
{
    if(signal(SIGALRM, alarmfunktion)==SIG_ERR)
    {
        printf("signal(): Fehler!\n");
        return 1;
    }
    alarm(3); /* Uhr erstmalig einschalten */
    while(anz<5)
    {
        printf("Bisher %d Alarme.\n", anz);
        sleep(1);
    }
    printf("Schalte Zeitschaltuhr aus...\n");
    return 0;
}
```

Für einen solchen immer wiederkehrenden Alarm kann auch die Funktion `setitimer()` verwendet werden, die einige Möglichkeiten mehr bietet.

### 8.4.2 Unterbrechung eines Systemaufrufs

Wenn man mit der Funktion `alarm()` einen Systemaufruf, etwa das Lesen von der Tastatur oder von einer Schnittstelle, unterbricht, dann wird dieser Systemaufruf in der Regel sofort wieder neu gestartet, d.h., das Programm läuft trotzdem nicht weiter. An dieser Stelle hilft die `longjmp()`-Funktion: Der Signalhandler läuft nicht mehr bis zum Ende, sondern springt wieder zurück zu einer Stelle, die vor dem Systemaufruf festgelegt wurde. Das Beispiel in `src/alarm+longjmp.c` zeigt die Technik:

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

```
5  jmp_buf status;
6  void alarmfunktion(int signalnr)
7  {
8     longjmp(status, 1);
9  }
10 /*********************************/
11 int main(void)
12 {
13    int rc, ch, lauf=0;
14    signal(SIGALRM, alarmfunktion);
15 /*---------------------------------*/
16    printf("Setze Sprungziel..\n");
17    rc=setjmp(status);
18    ++lauf;
19    printf("Durchlauf: %d\n", lauf);
20    if(rc==0)
21    {
22       int ch;
23       printf("Sprungzustand gespeichert.\n");
24       printf("Zeichen eingeben+Return druecken...\n");
25       alarm(4);
26       ch=getchar();
27       printf("Zeichen >>%c<< eingegeben. Danke!\n",ch);
28    }
29    else
30    {
31       printf("Zu spaet. Kein Zeichen eingelesen.\n");
32    }
33 /*---------------------------------*/
34    printf("Programmende.\n");
35    return 0;
36 }
```

- In Zeile 14 wird der Signalhandler eingerichtet.
- In Zeile 17 wird das Sprungziel gesetzt.
- Im ersten Durchlauf wird in den `if`-Zweig (Zeilen 22–27) gegangen.
- In Zeile 25 wird der Timer gesetzt auf 4 Sekunden
- Danach folgt der Systemaufruf, der nach 4 Sekunden beendet werden soll.
- Falls vorher jemand ein Zeichen eingibt und ↵ drückt, geht es weiter in Zeile 34.
- Nach 4 Sekunden ohne Eingabe dagegen schlägt das Signal zu, es geht dann zum Signalhandler ab Zeile 6.
- Dort wird gesprungen zu Zeile 17; Rückgabewert von `setjmp()` ist jetzt, beim zweiten Durchlauf, eins (aus dem zweiten Parameter von `longjmp()`.
- Damit wird jetzt in den `else`-Zweig abgebogen und eine Fehlermeldung ausgegeben.