```
1  /*****************************************************************************
2  *
3  * Signale                                        Filename: "signal_p.cc"
4  *
5  * Programmbeschreibung:
6  *
7  * Es wird ein Vater und ein Kindprozess erzeugt. Der Vaterprozess steuert
8  * mit verschiedenen Signalen den Ablauf des Kindprozesses.
9  *
10 *****************************************************************************
11 *
12 * Projekt      : Linux IPC Praktikum
13 *
14 * Datum/Name   : 25-Mai-98  durch M. Rueesch und D. Eisenegger
15 *
16 * Aenderungen  : 6-11-00 Markus Thaler: Signale mit sigemtyset()
17 *
18 *****************************************************************************/
19
20 //---------------------------------------------------------------------------
21 // Include Files
22 //---------------------------------------------------------------------------
23 #include <stdio.h>                              // Standard IO Funktionen
24 #include <stdlib.h>                             // Standardbibliothek
25 #include <unistd.h>                             // Linux Standard
26 #include <errno.h>                              // Fehlerbehandlung
27 #include <sys/types.h>                          // Linux Typendefinitionen
28 #include <sys/wait.h>                           // fuer wait(), waitpid()
29 #include <signal.h>                             // Signalhandling Funktionen
30
31 //---------------------------------------------------------------------------
32 // Lokale Funktionen
33 //---------------------------------------------------------------------------
34 void vater_prozess (pid_t kind_pid);    // Prototyp Vaterprozess
35 void signal_handler (int sig);
36
37 //****************************************************************************
38 // FUNKTION: main ()
39 //****************************************************************************
40
41 int main (void){
42
43    pid_t pid_c;                                 // Variable fuer Process ID
44
45    printf("Hauptprogramm startet mit (PID: %d, PPID: %d)\n",getpid(),getppid());
46    pid_c = fork();                              // Kindprozess erzeugen
47    switch (pid_c){
48       case -1: {                                // Fehlerfall
49          perror ("Fork :");
50          exit (-1);
51       }
52       case 0: {                                 // wir sind im Kindprozess
53          execl ("./signal_c.e", "signal_c.e", NULL);
54          perror ("execl error:  ");             // externes Programm starten
55          exit (-1);
56          break;
57       }
58       default: {
59          vater_prozess (pid_c);                 // Vaterprozess starten
60          break;
61       }
62    }
```

```
63    return 0;
64 }
65
66 //*************************************************************************
67 // FUNKTION: vater_prozess ()                    Vaterprozess
68 //*************************************************************************
69
70 void vater_prozess (pid_t kind_pid){
71
72    int    i;
73    int    status, stat;
74    struct sigaction neu;
75
76
77    printf ("Vater nimmt Arbeit auf (PID: %d, PPID: %d)\n", getpid(), getppid());
78    sleep(2);
79
80    neu.sa_handler = signal_handler;             // Signalhandler fuer das Signal
81    neu.sa_flags = SA_RESTART;                   // SIGCLD einrichten
82    sigemptyset(&neu.sa_mask);     // Zuslassen anderer Signale
83    sigaction (SIGCLD, &neu, NULL);              // Das Flag SA_SIGINFO existiert
84                                                 // nicht unter Linux!
85
86    signal (SIGINT, SIG_IGN);                    // Control C ignorieren !!
87
88    for (i = 0 ; i <= 4 ; i++){
89       printf ("VATER: Schlaufe %u \n", i);
90       sleep (2);
91    }
92
93    printf ("VATER: Kind gestoppt\n");           // Kind stoppen
94    kill (kind_pid, SIGSTOP);
95
96    for (i = 5 ; i <= 8 ; i++){
97       printf ("VATER: Schlaufe %u \n", i);
98       sleep (2);
99    }
100
101   printf ("VATER: Kind laeuft weiter\n");
102   kill (kind_pid, SIGCONT);                    // Kind wieder starten
103   sleep (2);
104
105   printf ("VATER: Sende SIGUSR1 an Kind...\n");
106   kill (kind_pid, SIGUSR1);
107   for (i = 8 ; i <= 10 ; i++){
108      printf ("VATER: Schlaufe %u \n", i);
109      sleep (2);
110   }
111   printf ("VATER: Jetzt bringen wir den Kindprozess um...\n");
112   kill (kind_pid, SIGINT);
113
114   printf ("VATER: Warten, bis Kind abgeschlossen hat...\n");
115   waitpid (kind_pid, &status, 0);
116   stat = WEXITSTATUS(status);                  // Status auswerten und in
117                                                // int wandeln
118   printf ("VATER: Exitstatus des Kindes: %d\n", stat);
119   printf ("*** Vater terminiert ***\n");
120 }
121
122 //*************************************************************************
123 // HANDLER: signal_handler ()
124 //*************************************************************************
```

```
125
126 void signal_handler (int sig){
127
128    printf ("-------------------------------------------------------\n");
129    printf ("Signalhandler Vater hat Signal Nr: %d empfangen\n", sig);
130    printf ("-------------------------------------------------------\n");
131 }
132
```