```
1  /*****************************************************************************
2  *
3  * "Semaphoren und Shared-Memory"                Filename: "shm.cc"
4  *
5  * Programmbeschreibung:
6  *
7  * Das vorliegende Programm demonstriert eine Moeglichkeit um von mehreren
8  * Prozessen auf einen gemeinsamen Speicherbereich zuzugreiffen.
9  *
10 * Zu Beginn des Programms wird eine Semaphore und ein gemeinsamer Memory-
11 * Bereich definiert und initialisiert. Ein Reader-Prozess (Kind) liest in
12 * regelmaessigen Abstaenden die Daten aus dem Shared-Memory und zeigt
13 * diese auf dem Bildschirm an waehrend ein Writer-Prozess (Vater) in
14 * zufaelligen Abstaenden ins Shared-Memory schreibt. Der Zugriff auf
15 * das Shared-Memory wird durch eine Semaphore geschuetzt.
16 *
17 *****************************************************************************
18 *
19 * Projekt      : Linux IPC Praktikum
20 *
21 * Datum/Name   : 25-Mai-98  durch M. Rueesch und D. Eisenegger
22 *
23 * Aenderungen  : 31.5.2007, M. Thaler: semaphore.cc und shm.cc
24 *
25 *****************************************************************************/
26
27 #include <stdio.h>                              // Standard IO Funktionen
28 #include <errno.h>                              // Fehlerbehandlung
29 #include <stdlib.h>                             // Standard Funktionen
30 #include <signal.h>                             // Signalbehandlung
31 #include <termio.h>                            // Terminal IO Funktionen
32 #include <unistd.h>                          // Fuer "sleep"
33
34 #include <sys/wait.h>                          // Fuer "wait"
35 #include <sys/types.h>                          // UNIX Typendefinitionen
36 #include <sys/ipc.h>                            // SVR4 IPC Mechanismen
37 #include <sys/sem.h>                            // SVR4 Semaphoren
38 #include <sys/shm.h>                            // SVR4 Shared Memory
39
40 #include "semaphore.h"
41 #include "shmem.h"
42
43 //****************************************************************************
44 //  Lokale Symbole und Typendefinitionen
45 //****************************************************************************
46
47 struct T_Shared_Memory{                         // *** Shared Memory Typ ***
48    int field[10];
49    char msg[20+1];                              // Wird in diesem Beispiel
50 };                                              // nicht benutzt
51
52 typedef T_Shared_Memory *shared_memory_ptr;
53
54 #define SHM_LEN sizeof (T_Shared_Memory)
55 #define keyFnameShm "/tmp/sem_key_file.shm.uebung"
56
57 #define keyFnameSem "/tmp/sem_key_file.sem.uebung"
58
59 #define PROJECT_ID 199
60
61
62 //****************************************************************************
```

```
63  // Lokale Funktionen
64  //*************************************************************************
65  void writer_prozess (pid_t kind_pid);
66  void reader_prozess ();
67  void handler        (int sig);
68  void cleanup        (int sig);
69
70  //*************************************************************************
71  // Lokale Variablen
72  //*************************************************************************
73  int reader_flag = 1;                            // Wenn =0: Kind terminiert
74  int shm_id;                                     // Shared Memory Identifikation
75  int sem_id;                                     // Semaphore Identifikation
76
77  //*************************************************************************
78  // FUNKTION: main ()
79  //*************************************************************************
80  int main (void)
81  {
82     pid_t pid_c;
83     int i;
84
85     struct sigaction sig;
86     sigemptyset(&sig.sa_mask);
87     sig.sa_handler = handler;
88     sig.sa_flags = 0;
89     sigaction(SIGUSR1, &sig, NULL);
90
91     shared_memory_ptr shared;        // Pointer auf eine Shared Memory Region
92
93     // ***** Definieren eines Shared Memory Bereichs *****
94
95     SharedMemory shm = SharedMemory(SHM_LEN,  keyFnameShm, PROJECT_ID);
96     shared =  (shared_memory_ptr)(shm.getSharedMemory());
97
98     Semaphore sem = Semaphore(1, keyFnameSem, PROJECT_ID);
99     sem.up(0);
100
101
102    for (i=0; i<10; i++) shared->field[i]=i;     // Memorysegment initialisieren
103
104    pid_c = fork ();                             // Neuer Prozess erzeugen
105    switch (pid_c)
106    {
107       case -1:                                  // Fehlerfall
108       {
109          perror ("Fork :");
110          exit (-1);
111       }
112       case 0:                                   // Kindprozess
113       {
114          reader_prozess ();
115          break;
116       }
117       default:                                  // Vaterprozess
118       {
119          writer_prozess (pid_c);
120          kill(getpid(),SIGINT);     // Aufruf von cleanup
121          break;
122       }
123    }
124    return 0;
```

```
125 }
126
127 //*****************************************************************************
128 // FUNKTION: writer_prozess ()         "VATER"
129 //*****************************************************************************
130 //void writer_prozess (pid_t pid_c, int sem_id, shared_memory_ptr shared)
131 void writer_prozess (pid_t pid_c)
132 {
133
134    SharedMemory shm = SharedMemory(0,  keyFnameShm, PROJECT_ID);
135    shared_memory_ptr shared =  (shared_memory_ptr)(shm.getSharedMemory());
136
137    Semaphore sem = Semaphore(0, keyFnameSem, PROJECT_ID);
138    signal (SIGINT, cleanup);
139
140    for (int i = 9 ; i >= 0; i--)
141    {
142       printf ("WRITER: Entering Critical Section\n");
143       sem.down(0);
144       printf ("WRITER: Inside Critical Section\n");
145       shared->field[i] = 10 - i;
146       sleep (1);
147       printf ("WRITER: Leave Critical Section\n");
148       sem.up(0);
149       sleep (rand() % 10);                  // Schreiber 0..10 Sekunden warten
150    }
151    kill (pid_c, SIGUSR1);                   // Signal zum Beenden an Kind senden
152    wait (NULL);                             // Warten bis Kind terminiert hat
153 }
154
155 //*****************************************************************************
156 // FUNKTION: reader_prozess ()        "KIND"
157 //*****************************************************************************
158 void reader_prozess ()
159 {
160    SharedMemory shm = SharedMemory(0, keyFnameShm, PROJECT_ID);
161    shared_memory_ptr shared =  (shared_memory_ptr)(shm.getSharedMemory());
162    Semaphore sem = Semaphore(0, keyFnameSem, PROJECT_ID);
163
164
165    while (reader_flag)
166    {
167       printf ("READER: Entering Critical Section\n");
168       sem.down(0);
169       printf ("READER: Inside Critical Section\n");
170       printf ("READER: %d, %d, %d, %d, %d, %d, %d, %d, %d, %d\n",
171                shared->field[0], shared->field[1], shared->field[2], shared->field[3],
172                shared->field[4], shared->field[5], shared->field[6], shared->field[7],
173                shared->field[8], shared->field[9]);
174       fflush (stdout);                          // Buffer an stdout ausgeben
175       sleep (2);
176       printf ("READER: Leave Critical Section\n");
177       sem.up(0);
178       sleep (1);
179    }
180    exit (0);
181 }
182
183 //*****************************************************************************
184 // FUNKTION: handler () (Signalhandler)
185 //*****************************************************************************
186 void handler (int sig)
```

```
187 {
188    reader_flag = 0;
189 }
190 
191 //*****************************************************************************
192 // FUNKTION: cleanup () (Signalhandler)
193 //*****************************************************************************
194 void cleanup (int sig)
195 {
196    semun sem_union;
197 
198    printf("CLEANUP: Ressourcen werden freigegeben \n");
199    SharedMemory shm = SharedMemory(0, keyFnameShm, PROJECT_ID);
200    shm.removeSharedMemory();
201    Semaphore sem = Semaphore(0, keyFnameSem, PROJECT_ID);
202    sem.removeSemaphore();
203    exit (0);
204 }
205 
206 //*****************************************************************************
```