```
1  /*****************************************************************************
2  *
3  * Message Queues                                Filename: "msg_bsp.cc"
4  *
5  * Programmbeschreibung:
6  *
7  * Der Vaterprozess (Client) sendet dem Kindprozess (Server) Rechnungsaufgaben.
8  * Der Meldungstyp bestimmt, ob die Meldung fuer den Server oder den Client ist.
9  * Je nach Operation (op) fuehrt der Server die Rechnung aus und sendet das
10 * Resultat dem Client zurueck.
11 *
12 *****************************************************************************
13 *
14 * Projekt      : Linux IPC Praktikum
15 *
16 * Datum/Name   : 25-Mai-98  durch M. Rueesch und D. Eisenegger
17 *
18 * Aenderungen  :
19 *
20 *****************************************************************************/
21
22 //---------------------------------------------------------------------------
23 //  Include-Dateien:
24 //---------------------------------------------------------------------------
25 #include <stdlib.h>
26 #include <stdio.h>                          // Standard IO Funktionen
27 #include <sys/wait.h>                       // feur wait, waitpid,...
28 #include <unistd.h>                         // Linux standards
29 #include <sys/types.h>                      // Linux Typendefinitionen
30 #include <sys/ipc.h>                        // SVR4 IPC Mechanismen
31 #include <sys/msg.h>                        // SVR4 Message Queues
32
33 //---------------------------------------------------------------------------
34 //  Lokale Symbole, Makros und Typendefinitionen:
35 //---------------------------------------------------------------------------
36 #define PLUS  '+'                           // Definitionen der Operationen.
37 #define MINUS '-'
38 #define MULT  '*'
39 #define DIV   '/'
40 #define ENDE  0x7F
41
42 #define REQUEST 100L                        // Meldungstypen
43 #define ANSWER  101L
44
45 typedef struct {
46   long  mtype;
47   char  op;
48   int arg1;
49   int arg2;
50 } msg_request_type;
51
52 typedef struct {
53   long  mtype;
54   long  result;
55 } msg_answer_type;
56
57 #define LEN_REQ sizeof (msg_request_type) - sizeof (long)
58 #define LEN_ANS sizeof (msg_answer_type) - sizeof (long)
59                                             // Laenge der Message definieren
60                                             // wobei die Laenge des Grundtyps ab-
61                                             // gezogen wird.
62 //---------------------------------------------------------------------------
```

```
63 //  Lokale Funktionen:
64 //------------------------------------------------------------------------------
65
66 void vater_prozess (pid_t kind_pid, int qid);
67 void calculate (int qid, int arg1, int arg2, char op, int lauf);
68 void kind_prozess (int qid);
69
70 //******************************************************************************
71 // FUNKTION: main ()
72 //******************************************************************************
73
74 int main (void){
75
76    pid_t pid_c;
77    int qid;
78
79    qid = msgget (IPC_PRIVATE, 0777 | IPC_CREAT | IPC_EXCL);
80                                                  // Queue erzeugen ohne Key
81    pid_c = fork ();                              // Kindprozess erzeugen
82    switch (pid_c)
83    {
84       case -1:{                                  // Fehlerfall
85          perror ("Fork :");
86          exit (-1);
87       }
88       case 0:{                                   // Kindprozess starten
89          kind_prozess (qid);
90          break;
91       }
92       default:{
93          vater_prozess (pid_c, qid);             // Vaterprozess starten
94          msgctl (qid, IPC_RMID, NULL);           // Queue wieder zerstoeren
95          break;
96       }
97    }
98    return 0;
99 }
100
101 //******************************************************************************
102 // FUNKTION: vater_prozess ()
103 //******************************************************************************
104
105 void vater_prozess (pid_t kind_pid, int qid){
106
107    msg_request_type sendbuf;                    // Buffer um Meldungen zu senden
108
109    printf ("Start der Verarbeitung\n");
110
111    calculate (qid, 12, 3, PLUS, 1);             // Rechnungsaufgaben senden
112    sleep(1);
113
114    calculate (qid, 12, 3, MINUS, 2);
115    sleep(1);
116
117    calculate (qid, 12, 3, MULT, 3);
118    sleep(1);
119
120    calculate (qid, 12, 3, DIV, 4);
121    sleep(1);
122
123    printf ("CLIENT: Server abschalten\n");
124
```

```
125    sendbuf.mtype = REQUEST;                    // Dem Client das Ende der Be-
126    sendbuf.op = ENDE;                          // rechnungen signalisieren.
127    msgsnd (qid, &sendbuf, LEN_REQ, 0);
128    wait (NULL);                                // Warten bis der Kindprozess
129 }                                              // terminiert hat.
130
131 //*****************************************************************************
132 // FUNKTION: calculate ()
133 //*****************************************************************************
134
135 void calculate (int qid, int arg1, int arg2, char op, int lauf){
136
137    msg_request_type sendbuf;                   // Buffer um Meldungen zu senden.
138    msg_answer_type  recvbuf;                   // Buffer fuer emfangene Meldungen.
139
140    sendbuf.mtype = REQUEST;                    // Sendebuffer einfuellen.
141    sendbuf.op = op;
142    sendbuf.arg1 = arg1;
143    sendbuf.arg2 = arg2;
144
145    printf("CLIENT: gebe %d. Rechnung (%d %c %d) in Auftrag.\n",lauf,arg1,op,arg2);
146
147    msgsnd (qid, &sendbuf, LEN_REQ, 0);         // Rechnung als Meldung senden
148
149    printf ("CLIENT: warte auf Antwort...\n");
150
151    msgrcv (qid, &recvbuf, LEN_ANS, ANSWER, 0); // Auf Resultat als Meldung warten
152
153    printf ("CLIENT: Das Ergebnis lautet: %ld\n", recvbuf.result);
154 }                                              // Resultat ausgeben.
155
156 //*****************************************************************************
157 // FUNKTION: kind_prozess ()
158 //*****************************************************************************
159
160 void kind_prozess (int qid){
161
162    msg_request_type recvbuf;                   // Variablen fuer Sende- und
163    msg_answer_type  sendbuf;                   // Empfangsmeldungen.
164
165    printf ("SERVER: Warte auf Anfrage...\n");
166
167    msgrcv (qid, &recvbuf, LEN_REQ, REQUEST, 0); // 1. Meldung lesen
168    while (recvbuf.op != ENDE)   {
169       printf ("SERVER: Verarbeite Anfrage...\n");
170       sendbuf.result = 0;
171       switch (recvbuf.op){                     // Je nach op Resultat berechnen
172          case PLUS:{
173             sendbuf.result = recvbuf.arg1 + recvbuf.arg2;
174             break;
175          }
176          case MINUS:{
177             sendbuf.result = recvbuf.arg1 - recvbuf.arg2;
178             break;
179          }
180          case MULT:{
181             sendbuf.result = recvbuf.arg1 * recvbuf.arg2;
182             break;
183          }
184          case DIV:{
185             sendbuf.result = recvbuf.arg1 / recvbuf.arg2;
186             break;
```

```
187          }
188          default:{
189             break;
190          }
191       }
192       printf ("SERVER: Sende Antwort...\n");
193       sendbuf.mtype = ANSWER;
194
195       if (msgsnd(qid, &sendbuf, LEN_ANS, 0) < 0){
196                                                  // Meldung (Resultat) senden.
197          perror ("error beim Queue beschreiben");
198          exit (1);
199       }
200       sleep (5);
201       if (msgrcv (qid, &recvbuf, LEN_REQ, REQUEST, 0) < 0){
202                                                  // naechste Meldung lesen.
203          perror ("Queue lesen");
204          exit (1);
205       }
206    }
207    printf ("SERVER: *** Ende ***\n");
208 }
```