Message Queues

7. Juni 2005

1 Übersicht

Neben Pipes gibt es unter Unix noch weitere Mechanismen zur lokalen Interprozess-Kommunikation (IPC). Lokale IPC bedeutet, dass nur Prozesse auf dem selben Rechner miteinander kommunizieren können bzw. sollen.

Diese Mechanismen sind FIFO-Dateien, Message Queues und Semaphoren. Auf letztere wird nicht näher eingegangen (siehe `man semget, semop, semctl`). FIFO-Dateien können als Spezialfall einer Message Queue angesehen werden.

Der wesentliche Unterschied zwischen Pipes und Message Queues ist, dass bei Message Queues die kommunizierenden Prozesse nicht verwandt sein müssen. Bei Pipes kann nur ein parent-prozess mit einem seiner child-Prozesse miteinander kommunizieren können. Über Message Queues können beliebige Prozesse, also auch solche, die nicht durch ein fork() miteinander verwandt sind, miteinander kommunizieren. Des weiteren können mehrere Prozesse gemeinsam eine einzige Message Queue zur Kommunikation nutzen, es muss nicht für jeden Kommunikationsweg eine eigene Pipe verwendet werden.

2 Arbeiten mit Message Queues

2.1 Erzeugung einer Message Queue

Zum Erzeugen einer Message Queue dient die Funktion `msgget()`:

```
#include <sys/msg.h>
#include <sys/types.h>
#include <sys/ipc.h>

int msgget(key_t key, int msgflg);
```

Der Rückgabewert von msgget ist der Handler der durch einen Key indentifierten Message Queue. Wenn die Message Queue nur von verwandten

Prozessen benutzt werden soll, kann als Key der Wert `IPC_PRIVATE` verwendet werden. Sollen nicht verwandte Prozesse die Queue nutzen, empfiehlt sich die Verwendung der Funktion `ftok()` zur Erzeugung des Keys:

```
\key_t ftok(const char *pathname, int proj_id);
```

Der Parameter `msgflg` von `msgget()` steuert die Zugriffsrechte und die Art der Erzeugung der Message Queue, ähnlich wie bei `open()`. Die einzelnen Optionen werden über ein bitweises OR kombiniert. Die Option `IPC_CREAT` sorgt dafür, dass die Queue erzeugt wird, wenn sie noch nicht existiert, die Option `IPC_EXCL` sorgt dafür, dass msgget() fehlschlägt, wenn die Queue bereits existiert.

2.2 Senden und empfangen von Nachrichten

Die Kommunikation über Message Queues erfolgt nicht wie bei Pipes direkt, sondern über Nachrichten (Messages). Eine Nachricht besteht aus einem Nachrichten-Kopf (header) und einem Nachrichten-Text (body). Der header besteht aus einer echt positiven Integer-Zahl, die den Typ der Nachricht beschreibt. Der body besteht aus einem Character-Array (nahezu) beliebiger, aber fest vorgegebener Größe. Eine Nachricht wird durch einen Struct beschrieben:

```
struct msgbuf{
        int mtype;
        char mtext[MSG_SIZE];
}
```

Zum Senden und Empfangen von Nachrichten dienen die Funktionen `msgsnd()` und `msgrcv()`:

```
int msgsnd(int msqid, struct msgbuf *msgp, size_t msgsz, int msgflg);
ssize_t  msgrcv(int msqid, struct msgbuf *msgp, size_t msgsz,
                long msg-typ, int msgflg);
```

Der erste Parameter beschreibt die Queue, die verwendet werden soll, der zweite die zu sendende bzw. zu empfangende Nachricht. Der Parameter `msgsz` gibt die maximale Größe des Elements `mtext` an, der Parameter `msgflg` beschreibt das Verhalten der Funktionen. Ist `msgflg` gleich 0, so blockieren die Funktionen den ausführenden Prozeß wenn nicht gesendet oder empfangen werden kann. Dieses Verhalten kann durch das setzen von `msgflg` auf `IPC_NOWAIT` geändert werden.

Der Parameter `msg-typ` von `msgrcv()` hat folgende Bedeutung:

0	Die erste Nachricht in der Queue wird empfangen
> 0	Die erste Nachricht mit mtype = msg-typ wird empfangen
< 0	Die erste Nachricht mit dem kleinsten mtype kleiner oder gleich dem Absolutbetrag von msg-typ wird empfangen.

2.3 Beenden einer Message Queue

Eine Message Queue kann mit der Funktion `msgctl()` entfernt werden:

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

Wird `cmd` auf `IPC_RMID` gesetzt entfernt der Aufruf der Funktion die mit `msqid` referenzierte Message Queue.

2.4 Ein einfaches Beispiel

Im folgenden Beispiel wird eine Message Queue erzeugt, die nur von verwandten Prozessen benutzt werden kann. Als Nachricht wird die aktuelle Zeit vom selben Prozess gesendet und empfangen.

```
#include<stdio.h>
#include<string.h>
#include<time.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<sys/wait.h>
#include<sys/errno.h>

extern int errno;
/*permissions for the message queue*/
#define MSGPERM 0600

int msgqid, rc;
int done;
time_t _ltime;

/*data structure for the messages*/
struct q_type {
  long mtype;
  char mtext[81];
}q_entry;

int main(int argc,char **argv)
{
  /* create message queue, abort if message queue with the same key exists */
  msgqid=msgget(IPC_PRIVATE,MSGPERM|IPC_CREAT|IPC_EXCL);
  if (msgqid < 0) {
    perror( strerror(errno) );
    printf("msgget failed, msgqid = %d\n", msgqid);
    exit(1);
  }
```

```
  printf("message queue %d created\n",msgqid);
  q_entry.mtype=1; /* set the type of message */
  _ltime = time(&_ltime); /* retrieving current time by calling time() */
  /* setting the right time format by means of ctime() */
  sprintf (q_entry.mtext,"%s\n",ctime(&_ltime));

  /*send the message*/
  rc = msgsnd(msgqid,&q_entry,80,0);
  /* take IPC_NOWAIT instead of 0, if you want the process not to suspend,
     if message delivery is not possible at the moment */

  if (rc < 0) {
    perror( strerror(errno) );
    printf("msgsnd failed, rc = %d\n", rc);
    exit(1);
  }

  /*receive the message*/
  rc=msgrcv(msgqid,&q_entry,80,0,0);
  if (rc < 0) {
    perror( strerror(errno) );
    printf("msgrcv failed, rc=%d\n", rc);
    exit(1);
  }
  printf("received msg: %s\n", q_entry.mtext);

  /*remove the message queue*/
  rc=msgctl(msgqid,IPC_RMID,NULL);
  if (rc < 0) {
    perror( strerror(errno) );
    printf("msgctl (return queue) failed, rc=%d\n", rc);
    exit(1);
  }
  printf("message queue %d is gone\n",msgqid);

}
```