POLITÉCNICO DE PORTALEGRE

Escola Superior de Tecnologia, Gestão e Design

OPERATING SYSTEMS

Exercise 3 – Message Queues

1. Create a message queue and send a message.

The procedures for message queues are partly similar to those for semaphores. To create a message queue, use the **msgget** function, as exemplified in the following program, which also sends a message to the created queue.

To send a message, use the **msgsnd** function. The first argument is, of course, the id obtained from **msgget**. The second argument is the message itself, the third argument is the size of the message and the last argument is usually 0 (but we will see other hypotheses later).

Regarding the message, the rule is that there must be an **int** field (designated as the message "type") before the message itself. Within this rule, it is the program that defines the structure of the messages. In this case, a structure **msg_struct** was defined which provides for messages containing a maximum of 250 characters.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define exit_on_error(s,m) if ( s < 0 ) { perror(m); exit(1); }

typedef struct {
    long type;
    char text[250];
} msg_struct;

int main(int argc, char *argv[])
{
    int msg_id;
    int status;
    msg_struct msg;

    // connect to the message queue
    msg_id = msgget(1000, 0600 | IPC_CREAT);
    exit_on_error(msg_id, " Creation/Connection");

    // send a message
    msg.type = 1;
    strcpy(msg.text, "Any message...");
    status = msgsnd(msg_id, &msg, sizeof(msg.text), 0);
    exit_on_error(status, "Send");
    printf("Message sent!\n");
}
```

2. Commands

ipcs -q command (or just **ipcs**) allows you to see the message queues created. You can also, from this list, see the number of messages currently waiting in the queue.

3. Receive a message

The following program receives (receives, in this context, means removing) a message from the queue. The only sensible difference is that, instead of **msgsnd** to send a message, we have the **msgrcv** function to receive.

In this call, the first, second and third arguments have identical meanings to send. The second indicates the structure that will receive the message and the third indicates the size of this structure.

The fourth argument indicates the type of message you want to receive. The program will only receive messages of the indicated type, in the case of type 1 (that is, if there are messages of another type in the queue, for this purpose, it is as if they did not exist). Fortunately, the previous program sent the message as being type 1!

Finally, the last argument remains at 0. This means that, if there is no message available in the queue (of type 1), the program will be blocked in waiting; it will only unlock when someone sends a new message (type 1) to the queue.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define exit_on_error(s,m) if ( s < 0 ) { perror(m); exit(1); }

typedef struct {
   long type;
   char text[250];
} msg_struct;

int main(int argc, char *argv[])
{
   int msg_id;
   int status;
   msg_struct msg;

    // connect to the message queue
    msg_id = msgget(1000, 0600 | IPC_CREAT);
    exit_on_error(msg_id, "Creation/Connection");

    // receive a message (blocks if none)
    status = msgrcv(msg_id, &msg, sizeof(msg.text), 1, 0);
    exit_on_error(status, "Reception");
    printf("MESSAGE <%s>\n", msg.text);
}
```

4. No wait

Try the previous programs with different sequences; for example: with the first one you put several messages in the queue; with the second one you remove each one of them. Obviously, the second program only blocks if, at that moment, there are no messages available.

Alternatively, the last argument to **msgrcv** can be **IPC_NOWAIT**. In this case the **msgrcv** does not block. It fetches a message and if there is one, it is read; otherwise, the program continues, returning an error code to the function.

5. Example: client/server

The following example implements a model in which one or more clients send messages to a server, which responds to each of them.

The message structure is now more elaborate. In addition to the mandatory part (i.e. the "type" of the message) the message now has two fields: an identifier that will contain the number of the process sending the message and the text of the message itself.
The scheme is as follows:

- the server only searches the queue for type 1 messages;
- the client sends a type 1 message to the server, indicating its PID in the body of the message;
- the server responds with a message with a type equal to the pid indicated by the client;
- accordingly, the client searches the message queue for a response with a type equal to its own pid;

SERVER

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define exit_on_error(s,m) if ( s < 0 ) { perror(m); exit(1); }

typedef struct {
    long type;
    struct _msg_struct {
        int pid;
        char text[250];
    } msg;
} msg_struct;

int main(int argc, char *argv[])
{
    int msg_id;
    int status;
    msg_struct msg;
    int prize = 0;

    // connect to the message queue
    msg_id = msgget(1000, 0600 | IPC_CREAT);
    exit_on_error(msg_id, "Creation/Connection");

   // receive a message (blocks if none)
   while (1) {
      status = msgrcv(msg_id, &msg, sizeof(msg.msg), 1, 0);
      exit_on_error(status, "Reception");

      printf("From PID %d: <%s>\n", msg.msg.pid, msg.msg.text);
      msg.type = msg.msg.pid;
      if (++prize %5 == 0){
          strcpy(msg.msg.text, "PRIZE!");
          printf("Prize for PID =%d\n", msg.msg.pid);
      }
      else
          strcpy (msg.msg.text, "Try again.");

      status = msgsnd(msg_id, &msg, sizeof(msg.msg), 0);
      exit_on_error(status, "Answer");
   }
}
```

CLIENT

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define exit_on_error(s,m) if ( s < 0 ) { perror(m); exit(1); }

typedef struct {
   long type;
   struct _msg_struct {
      int pid;
      char text[250];
   } msg;
} msg_struct;

int main(int argc, char *argv[])
{
   int msg_id;
   int status;
   msg_struct msg;
   char s[250];

   // connect to the message queue
   msg_id = msgget(1000, 0600 | IPC_CREAT);
   exit_on_error(msg_id, "Creation/Connection");

   printf("PID=%d\n", getpid());
   while (1) {
      msg.type = 1;
      msg.msg.pid = getpid();
      printf("Message: ");
      fgets(s, 250, stdin);
      strcpy(msg.msg.text, s);
      status = msgsnd(msg_id, &msg, sizeof(msg.msg), 0);
      exit_on_error(status, "Send");

      status = msgrcv(msg_id, &msg, sizeof(msg.msg), getpid(), 0);
      exit_on_error(status, "Answer");
      printf("Answer: %s\n", msg.msg.text);
  }
}
```