The Department of Electrical and Computer Engineering

The University of Alabama in Huntsville

CPE 435 Lab-05

Message Queue

Introduction

There are many applications in which processes need to cooperate with each other. This is always the case for a variety of forms of inter-process communication that can be used under LINUX. These support resource sharing, synchronization, connectionless and connection-oriented data exchange or combinations of these.

Message queues are a refined method of inter process communication. As an application, we can implement a printer spooler using message queues. Some form of message passing between processes is now part of many modern operating systems. All messages are stored in the kernel, and have an associated message queue identifier. It is this identifier, which we call an msqid that identifies a particular queue of messages, processes read and write messages to arbitrary queues. Every message on a queue has the following attributes:

Long Integer Type
Length of data portion of message (can be 0)
Data (if length is greater than 0)

For every message queue in the system, the kernel maintains the following structure of information:

```
# include <sys/types.h>
# include <sys/ipc.h> /*defines the ipc_perm structure */
stuct msqid_ds
{
stuct ipc_perm msg_perm; /* operation permission struct */
struct msg *msg_first; /* ptr to first message on q */
struct msg *msg _last; /* ptr to last message on q */
ushort msg_cbytes; /* current # bytes on q */
ushort msg_qnum; /* current # of messages on q */
ushort msg_qbytes /* max # of bytes allowed on q */
ushort msg_lspid; /* pid of last msgsnd */
ushort msg_lrpid; /* pid of last msgrcv */
time_t msg_stime; /* time of last msgsnd */
time_t msg_rtime; /* time of last msgrcv */
time_t msg_ctime; /* time of last msgct1 (that changed the above) */ }
```

Assignment

Create a Linux Chat Server

You will write two applications (named Process A and Process B). Process A should create a message queue and write a message to the queue (read from user through stdin). Process A should wait for data to be received from Process B. Process B should send some message to Process A using the message queue. The message sent by Process B should be user typed string through stdin. Process A should respond to Process B with another message sent through the message queue. This is also the message typed by the user in Process A's stdin. Keep the process of sending messages back and forth until one types "Exit". The received messages should be displayed to the terminal by both process A and B.

Please make sure to delete the message queue after you are done. You can check the current shared message queues with ***ipcs -q*** to see if your message queue was deallocated.

Topics for Theory

int msgget (key_t key, int msgflag); // Functions for Creating and Handling Message Queues
int msgsnd (int msqid, struct msgbuf *ptr, int length, int flag); //function to send a message, id from above
int msgrcv (int msqid, struct msgbuf * ptr, int length ,long msgtype, int flag); //function to Receive Message
int msgctl(int msqid, int cmd, struct msqid_ds *buff); // provides a variety of control operations on a message queue.

Coding Hint

Please review the presentation slide on Interprocess Communication IPC Shared Memory. Especially, please go through pages 12 and onwards for examples.
You can use cpp constructs line cin and cout.
You can create a shared header that contains the Key information, type information and structure similar to the structure used in lab 4.

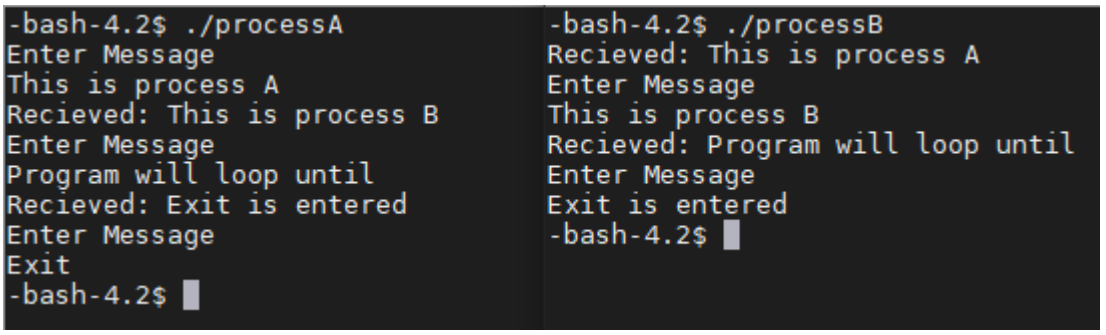
Process A:

1. Create a message queue using *msgget()* . Use IPC_CREAT flag with 0666 as you did for shared memory
2. read user input using scanf/cin function from stdin. Copy this message to the data part of the structure object.
3. send the message using *msgsnd()* function
    a. if message was **Exit**, break
4. receive message from Process B using *msgrcv()* function
    a. compare the received message with **Exit,** if **Exit** break
    b. display message received
5. Continue to Step 2
6. Remove the message queue using *msgctl()* with IPC_RMID flag.

Process B:

1. Access the message queue created by Process A using *msgget()* function. You may need to create the same key.
2. wait until you receive message from Process A
    a. if the message was **Exit**, break
    b. display message received
3. read from user input
4. send the message to Process A using message queue
    a. if the message was **Exit**, break
5. Continue to Step 2
6. Remove the message queue using *msgctl()* function.

Sample Output



Research Online

Please include this in your report also

How do you make a process wait to receive a message and not return immediately?

Message Queue vs Shared Memory (discuss use and differences).

Research use of function **ftok()**, what is its use?

What does IPC_NOWAIT do?

Note: Please see the next page for lab report format.

Deliverables

Lab Report

The following material in each section is expected:

1. Cover page with your name, lab number, course name, and dates
2. Theory/Background (Material or methods relevant to the lab, a few sentences on each)
    a. msgget()
    b. msgsnd()
    c. msgrcv()
    d. msgctl()
3. Observations (Show output demonstrating the two processes can communicate using the message queue, including looping behavior and the use of an exit message)
4. Lab Questions (Make sure to answer the following with at least a few sentences)
    a. How do you make a process wait to receive a message and not return immediately?
    b. Message Queue vs Shared Memory (discuss use and differences).
    c. Research use of function **ftok()**, what is its use?
    d. What does IPC_NOWAIT do?
5. Conclusion (Did your program work as expected, what can you take away from the lab?)
6. Appendix (for source code, submit the text in a table)

The report should be submitted as a single pdf document with the source code for your program within it.

Demonstration

The following material in each section is expected:

1. Introduce yourself and give the name of the lab
2. Compile the program
3. Show that the two processes interact with message queue as required, including the exit condition and looping behavior of the processes

The demonstration may be in person or recorded and submitted as an mp4 file alongside the report.