IPC: Message Queue

Subject:- Unix Operating System

System Lab Class :- TYIT

Name PRN

Aditi Sudhir Ghate 2020BTEIT00044

Assignment No 7a

Title-Write a program to perform IPC using message and send did u get this

Objective:

1. To learn about IPC through message queue.
2. Use of system call and IPC mechanism to write effective application programs

Theory:

Two (or more) processes can exchange information via access to a common system message queue. The sending process places via some (OS) message-passing module a message onto a queue which can be read by another process

Each message is given an identification or type so that processes can select the appropriate message. Process must share a common key in order to gain access to the queue in the first place.

Basic Message Passing IPC messaging lets processes send and receive messages, and queue messages for processing in an arbitrary order. Unlike the file byte-stream data flow of pipes, each IPC message has an explicit length. Messages can be assigned a specific type. Because of this, a server process can direct message traffic between clients on its queue by using the client process PID as the message type. For single-message transactions, multiple server processes can work in parallel on transactions sent to a shared message queue.

When a message is sent, its text is copied to the message queue. The msgsnd() and msgrcv() functions can be performed as either blocking or non-blocking operations. Non-blocking operations allow for asynchronous message transfer -- the process is not suspended as a result of sending or receiving a message. In blocking or synchronous message passing the sending process cannot continue until the message has been transferred or has even been acknowledged by a receiver. IPC signal and other mechanisms can be employed to implement such transfer. A blocked message operation remains suspended until one of the following three conditions occurs:

1.The call succeeds.
2.The process receives a signal.
3.The queue is removed

1. Initialising the Message Queue
   - The msgget() function initializes a new message queue
   - int msgget(key_t key, int msgflg)
   - It can also return the message queue ID (msqid) of the queue corresponding to the key argument. The value passed as the msgflg argument must be an octal integer with settings for the queue's permissions and control flags.

2.Controlling message queues
   - The msgctl() function alters the permissions and other characteristics of a message queue. The owner or creator of a queue can change its ownership or permissions using msgctl() Also, any process with permission to do so can use msgctl() for control operations.
   - int msgctl(int msqid, int cmd, struct msqid_ds *buf)

3.Sending and Receiving Messages
   - The msgsnd() and msgrcv() functions send and receive messages, respectively:
   - int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
   - int msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg);
   - The msqid argument must be the ID of an existing message queue. The msgp argument is a pointer to a structure that contains the type of the message and its text.

Data Dictionary:

SR.NO	Variable/function	DataType	Use
1.	msqid	int	For Socket Tuple
2.	msgflg	int	For Semaphore
3.	key	key_t	Semaphore id

Program-

Messagesend.c

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <string.h>
#include<stdlib.h>
#define MSGSZ
typedef struct msgbuf {
long mtype;
char mtext[MSGSZ];
} message_buf;
main()
```

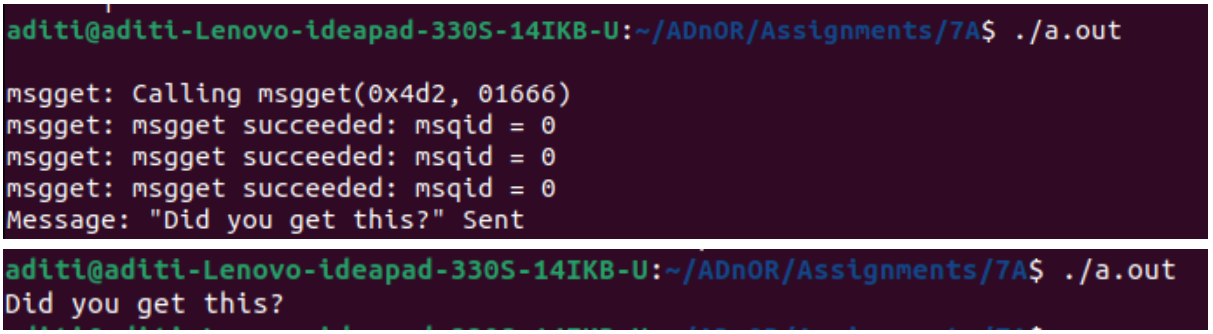
```
{
int msqid;
int msgflg = IPC_CREAT |0666;
key_t key;
message_buf sbuf;
size_t buf_length;
key = 1234;
(void) fprintf(stderr, "\nmsgget: Calling msgget(%#lx,\
%#o)\n",key, msgflg);
if ((msqid = msgget(key, msgflg )) < 0)
{ perror("msgget");
exit(1);
}
else
(void) fprintf(stderr,"msgget: msgget succeeded: msqid =
%d\n", msqid);
sbuf.mtype = 1;
(void) fprintf(stderr,"msgget: msgget succeeded: msqid =
%d\n", msqid);
(void) strcpy(sbuf.mtext, "Did you get this?");
(void) fprintf(stderr,"msgget: msgget succeeded: msqid =
%d\n",
msqid); buf_length = strlen(sbuf.mtext) + 1 ;

if (msgsnd(msqid, &sbuf, buf_length, IPC_NOWAIT) < 0) {
printf ("%d, %d, %s, %d\n", msqid, sbuf.mtype, sbuf.mtext,
buf_length);
perror("msgsnd");
exit(1);
}
else
printf("Message: \"%s\" Sent\n", sbuf.mtext);
exit(0);
}
```

Messagreceive.c

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#define MSGSZ 128
typedef struct msgbuf {
long mtype;
char mtext[MSGSZ];
} message_buf;
main()
```

```
{
int msqid;
key_t key;
message_buf rbuf; /*
* Get the message queue id for the
* "name" 1234, which was created by
* the server.
*/
key = 1234;
if ((msqid = msgget(key, 0666)) < 0)
{ perror("msgget");
exit(1);
}
/*
* Receive an answer of message type
1. */
if (msgrcv(msqid, &rbuf, MSGSZ, 1, 0) < 0) {
perror("msgrcv");
exit(1);
}
/* Print the answer.*/
printf("%s\n", rbuf.mtext);
exit(0);
}
```

Output-



Conclusion-

Use of message queue functions like msgget, msgsend, and msgrecv to implement message passing mechanism between server and client studied and implemented it to intoduce concept of chatting.

Reference-

Dave's Programming in C Tutorials