**Name:Rohan Nivas Chavan**

**PRN:2019BTEIT00084**

**UOS LAB**

**IPC: Message Queues: msgget, msgsnd, msgrcv**

**7.3 Write a server program and two client programs so that the server can communicate privately to each client individually via a single message queue.**

Message queues are one of the interprocess communication mechanisms available under

Linux. Message queues, shared memory and semaphores are normally listed as the three

interprocess communication mechanisms under Linux. Semaphores, though, are really for process

synchronization. In practice, shared memory, aided by semaphores, makes an interprocess

communication mechanism. Message queues is the other interprocess communication mechanis.

1.msgget():

In order to create a new message queue, or access an existing queue, the msgget()

system call is used.

SYSTEM CALL: msgget();

PROTOTYPE: int msgget ( key\_t key, int msgflg );

RETURNS: message queue identifier on success

-1 on error: errno = EACCESS (permission denied)

EEXIST (Queue exists, cannot create)

EIDRM (Queue is marked for deletion)

ENOENT (Queue does not exist)

176

ENOMEM (Not enough memory to create queue)

ENOSPC (Maximum queue limit exceeded)

2.msgsnd():

The msgsnd() function is used to send a message to the queue associated with the message

queue identifier specified by msgid.

The argument msgp points to a user-defined buffer that must contain first a field of type long int

that will specify the type of the message, and then a data portion that will hold the data bytes of the message. The structure below is an example of what this user-defined buffer should look like:

struct message

{

long int mtype; Message type

int mtext[n]; Message text

}

3.msgrcv():

The msgrcv() function reads a message from the queue associated with the message queue

identifier that msqid specifies and places it in the user-defined structure that msgp points to. When

successfully completed, the following actions are taken with respect to the data structure associated with msqid:

• msg\_lrpid is set to the process ID of the calling process.

• msg\_rtime is set to the current time.

Program:

Write a server program and two client programs so that the server can communicate privately to

each client individually via a single message queue

• (Process 1) Sends the message "Are you hearing me?"

• (Process 2) Receives the message and replies "Loud and Clear"

• (Process 1) Receives the reply and then says "I can hear you too"

177

Server.c

#include <stdio.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

extern void exit();

extern void perror();

main()

{

key\_t key; /\* key to be passed to msgget() \*/

int msgflg, /\* msgflg to be passed to msgget() \*/

msqid; /\* return value from msgget() \*/

(void) fprintf(stderr,

"All numeric input is expected to follow C conventions:\n");

(void) fprintf(stderr,

"\t0x... is interpreted as hexadecimal,\n");

(void) fprintf(stderr, "\t0... is interpreted as octal,\n");

(void) fprintf(stderr, "\totherwise, decimal.\n");

(void) fprintf(stderr, "IPC\_PRIVATE == %#lx\n", IPC\_PRIVATE);

(void) fprintf(stderr, "Enter key: ");

(void) scanf("%li", &key);

(void) fprintf(stderr, "\nExpected flags for msgflg argument

are:\n");

(void) fprintf(stderr, "\tIPC\_EXCL =\t%#8.8o\n", IPC\_EXCL);

(void) fprintf(stderr, "\tIPC\_CREAT =\t%#8.8o\n", IPC\_CREAT);

(void) fprintf(stderr, "\towner read =\t%#8.8o\n", 0400);

(void) fprintf(stderr, "\towner write =\t%#8.8o\n", 0200);

(void) fprintf(stderr, "\tgroup read =\t%#8.8o\n", 040);

178

(void) fprintf(stderr, "\tgroup write =\t%#8.8o\n", 020);

(void) fprintf(stderr, "\tother read =\t%#8.8o\n", 04);

(void) fprintf(stderr, "\tother write =\t%#8.8o\n", 02);

(void) fprintf(stderr, "Enter msgflg value: ");

(void) scanf("%i", &msgflg);

(void) fprintf(stderr, "\nmsgget: Calling msgget(%#lx,

%#o)\n",

key, msgflg);if ((msqid = msgget(key, msgflg)) == -1)

{

perror("msgget: msgget failed");

exit(1);

} else {

(void) fprintf(stderr,

"msgget: msgget succeeded: msqid = %d\n", msqid);

exit(0);

}

}

Client1.c

#include <stdio.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <time.h>

static void do\_msgctl();

extern void exit();

extern void perror();

static char warning\_message[] = "If you remove read permission

for \

179

yourself, this program will fail frequently!";

main()

{

struct msqid\_ds buf; /\* queue descriptor buffer for IPC\_STAT

and IP\_SET commands \*/

int cmd, /\* command to be given to msgctl() \*/

msqid; /\* queue ID to be given to msgctl() \*/

(void fprintf(stderr,

"All numeric input is expected to follow C conventions:\n");

(void) fprintf(stderr,

"\t0x... is interpreted as hexadecimal,\n");

(void) fprintf(stderr, "\t0... is interpreted as octal,\n");

(void) fprintf(stderr, "\totherwise, decimal.\n");

/\* Get the msqid and cmd arguments for the msgctl() call. \*/

(void) fprintf(stderr,

"Please enter arguments for msgctls() as requested.");

(void) fprintf(stderr, "\nEnter the msqid: ");

(void) scanf("%i", &msqid);

(void) fprintf(stderr, "\tIPC\_RMID = %d\n", IPC\_RMID);

(void) fprintf(stderr, "\tIPC\_SET = %d\n", IPC\_SET);

(void) fprintf(stderr, "\tIPC\_STAT = %d\n", IPC\_STAT);

(void) fprintf(stderr, "\nEnter the value for the command: ");

(void) scanf("%i", &cmd);

switch (cmd) {

case IPC\_SET:/\* Modify settings in the message queue control structure.

\*/

(void) fprintf(stderr, "Before IPC\_SET, get current

values:");

180

/\* fall through to IPC\_STAT processing \*/

case IPC\_STAT:

/\* Get a copy of the current message queue control

\* structure and show it to the user. \*/

do\_msgctl(msqid, IPC\_STAT, &buf);

(void) fprintf(stderr, ]

"msg\_perm.uid = %d\n", buf.msg\_perm.uid);

(void) fprintf(stderr,

"msg\_perm.gid = %d\n", buf.msg\_perm.gid);

(void) fprintf(stderr,

"msg\_perm.cuid = %d\n", buf.msg\_perm.cuid);

(void) fprintf(stderr, "msg\_perm.cgid = %d\n", buf.msg\_perm.cgid);

(void) fprintf(stderr, "msg\_perm.mode = %#o, ", buf.msg\_perm.mode);

(void) fprintf(stderr, "access permissions = %#o\n", buf.msg\_perm.mode & 0777);

(void) fprintf(stderr, "msg\_cbytes = %d\n", buf.msg\_cbytes);

(void) fprintf(stderr, "msg\_qbytes = %d\n", buf.msg\_qbytes);

(void) fprintf(stderr, "msg\_qnum = %d\n", buf.msg\_qnum);

(void) fprintf(stderr, "msg\_lspid = %d\n", buf.msg\_lspid);

(void) fprintf(stderr, "msg\_lrpid = %d\n", buf.msg\_lrpid);

(void) fprintf(stderr, "msg\_stime = %s", buf.msg\_stime ?

ctime(&buf.msg\_stime) : "Not Set\n");

(void) fprintf(stderr, "msg\_rtime = %s", buf.msg\_rtime ?

ctime(&buf.msg\_rtime) : "Not Set\n");

(void) fprintf(stderr, "msg\_ctime = %s", ctime(&buf.msg\_ctime));

if (cmd == IPC\_STAT)

break;

/\* Now continue with IPC\_SET. \*/

(void) fprintf(stderr, "Enter msg\_perm.uid: ");

181

(void) scanf ("%hi", &buf.msg\_perm.uid);

(void) fprintf(stderr, "Enter msg\_perm.gid: ");

(void) scanf("%hi", &buf.msg\_perm.gid);

(void) fprintf(stderr, "%s\n", warning\_message);

(void) fprintf(stderr, "Enter msg\_perm.mode: ");

(void) scanf("%hi", &buf.msg\_perm.mode);

(void) fprintf(stderr, "Enter msg\_qbytes: ");

(void) scanf("%hi", &buf.msg\_qbytes);

do\_msgctl(msqid, IPC\_SET, &buf);

break;

case IPC\_RMID:

default:/\* Remove the message queue or try an unknown command. \*/

do\_msgctl(msqid, cmd, (struct msqid\_ds \*)NULL);

break;

}

exit(0);

}

/\*

\* Print indication of arguments being passed to msgctl(), call

\* msgctl(), and report the results. If msgctl() fails, do not

\* return; this example doesn't deal with errors, it just reports

\* them.

\*/

static void

do\_msgctl(msqid, cmd, buf)

struct msqid\_ds \*buf; /\* pointer to queue descriptor buffer \*/

int cmd, /\* command code \*/

msqid; /\* queue ID \*/

182

{

register int rtrn; /\* hold area for return value from msgctl()

\*/

(void) fprintf(stderr, "\nmsgctl: Calling msgctl(%d, %d,

%s)\n",

msqid, cmd, buf ? "&buf" : "(struct msqid\_ds \*)NULL");

rtrn = msgctl(msqid, cmd, buf);

if (rtrn == -1) {

perror("msgctl: msgctl failed");

exit(1);

} else {

(void) fprintf(stderr, "msgctl: msgctl returned %d\n",

rtrn);

}

}Client2.c

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <stdio.h>

#include <string.h>

#define MSGSZ

128

/\*

\* Declare the message structure.

\*/

typedef struct msgbuf {

long mtype;

char mtext[MSGSZ];

183

} message\_buf;

main()

{

int msqid;

int msgflg = IPC\_CREAT | 0666;

key\_t key;

message\_buf sbuf;

size\_t buf\_length;

/\*

\* Get the message queue id for the

\* "name" 1234, which was created by

\* the server.

\*/

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);

/\* We'll send message type 1\*/

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 ;

184

/\* Send a message.\*/

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); } Output:

Conclusion:

Use of message queue functions like msgget, msgsend, and msgrecv to implement message passing mechanism between server and client studied. Various clients communicated independently with server. References:Dave’s Programming in C Tutorials