BAHIR DAR UNIVERSTY

INSTITUTE OF TECHNOLOGY

FACULTY OF COMPUTING

DEPARTMENT OF SOFTWARE ENGINEERING

Course: operating system and system programming

Individual assignment two

Topic: message send system call

Studentname: Ephrem Habtamu

Id: 1308250

Date of submission:-17/11/2014E.C

Instructor:wendimubaye

INTRODUCTION

In this text I will try to wite in detail about message send system call hohow it works the parmeters and the flags are also discussed in detail and also there is acode implementation of the flags. this system call four parametrs and anumber of flags that is used to implement its operation.

MESSAGE SEND SYSTEM CALL

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

int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

The `msgsnd()` function is used to send a message to the queue associated with the message queue identifier specified by `msqid`.

The `msgp` argument 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 might look like:

```
struct  mymsg {
        long  mtype;     /* message type */
        char  mtext[1];  /* message text */
}
```

The `mtype` member is a non-zero positive type `long int` that can be used by the receiving process for message selection.

The `mtext` member is any text of length `msgsz` bytes. The `msgsz` argument can range from 0 to a system-imposed maximum.

The `msgflg` argument specifies the action to be taken if one or more of the following are true:

- The number of bytes already on the queue is equal to `msg_qbytes`. See
- The total number of messages on the queue would exceed the maximum allowed by the system. See NOTES.

These actions are as follows:

- If (`msgflg&IPC_NOWAIT`) is non-zero, the message will not be sent and the calling process will return immediately.
- If (`msgflg&IPC_NOWAIT`) is 0, the calling process will suspend execution until one of the following occurs:
    - The condition responsible for the suspension no longer exists, in which case the message is sent.
    - The message queue identifier `msqid` is removed from the system (see `msgctl`(2)); when this occurs, `errno` is set equal to EIDRM and `-1` is returned.

- The calling process receives a signal that is to be caught; in this case the message is not sent and the calling process resumes execution in the manner prescribed in `sigaction`(2).

Upon successful completion, the following actions are taken with respect to the data structure associated with `msqid` (see `Intro`(2)):

- `msg_qnum` is incremented by 1.
- `msg_lspid` is set equal to the process ID of the calling process.
- `msg_stime` is set equal to the current time.

Return Values

Upon successful completion, `0` is returned. Otherwise, `-1` is returned, no message is sent, and `errno` is set to indicate the error.

Errors

The `msgsnd()` function will fail if:

EACCES
Operation permission is denied to the calling process. See `Intro`(2).

EAGAIN
The message cannot be sent for one of the reasons cited above and (`msgflg&IPC_NOWAIT`) is non-zero.

EIDRM
The message queue identifier `msgid` is removed from the system.

EINTR
The `msgsnd()` function was interrupted by a signal.

EINVAL
The value of `msqid` is not a valid message queue identifier, or the value of `mtype` is less than 1.

The value of `msgsz` is less than 0 or greater than the system-imposed limit.

The `msgsnd()` function may fail if:

EFAULT
The `msgp` argument points to an illegal address.

parameters

msqid

(Input) Message queue identifier, a positive integer. It is returned by the msgget() function and used to identify the message queue to send the message to.

msgp

(Input) Pointer to a buffer with the message to be sent. See above for the details on the format of the buffer.

msgsz

(Input) Length of the data part of the message to be sent.

msgflg

(Input) Operations flags. The value of *msgflg* is either zero or is obtained by performing an OR operation on one or more of the following constants:

Specifies the action to be taken if one or more of the following are true:

- The number of bytes already on the queue is equal to `msg_qbytes` from the `msqid_ds` data structure.
- The total number of messages on all queues system-wide is equal to the system-imposed limit.

If (*msgflg* `&` `IPC_NOWAIT`) is non-zero, the message is not sent and the calling process returns immediately. If (*msgflg* `&` `IPC_NOWAIT`) is zero, the calling process suspends execution until one of the following occurs:

- The condition responsible for the suspension no longer exists, in which case the message is sent.
- *msqid* is removed from the system. When this occurs, `errno` is set to `EIDRM`, and `-1` is returned.
- The calling process receives a signal that is to be caught. In this case, the message is not sent and the calling process resumes execution as appropriate for the signal.

Flags in message send system call.

MSG_OOB

Sends out-of-band data on the socket. Only SOCK_STREAM sockets support out-of-band data. The out-of-band data is a single byte.

Before out-of-band data can be sent between two programs, there must be some coordination of effort. If the data is intended to not be read inline, the recipient of the out-of-band data must specify the recipient of the SIGURG signal that is generated when the out-of-band data is sent. If no recipient is set, no signal is sent. The recipient is set up by using F_SETOWN operand of the fcntl() command, specifying either a pid or gid. For more information on this operand, refer to the fcntl() command.

The recipient of the data determines whether to receive out-of-band data inline or not inline by the setting of the SO_OOBINLINE option of setsockopt(). For more information on receiving out-of-band data, refer to the setsockopt(), recv(), recvfrom() and recvmsg() commands.

MSG_DONTROUTE

The SO_DONTROUTE option is turned on for the duration of the operation. This is usually used only by diagnostic or routing programs.

MSG_NOSIGNAL

Don't generate a **SIGPIPE** signal if the peer on a stream-oriented socket has closed the connection. The **EPIPE** error is still returned. This provides similar behavior to using sigaction(2) to ignore **SIGPIPE**, but, whereas **MSG_NOSIGNAL** is a per-call feature, ignoring **SIGPIPE** sets a process attribute that affects all threads in the process.

MSG_EOR

Terminates a record (when this notion is supported, as for sockets of type **SOCK_SEQPACKET**).

MSG_MORE

The caller has more data to send. This flag is used with TCP sockets to obtain the same effect as the **TCP_CORK** socket option , with the difference that this flag can be set on a per-call basis.

Since Linux 2.6, this flag is also supported for UDP sockets, and informs the kernel to package all of the data sent in calls with this flag set into a single datagram which is transmitted only when a call is performed that does not specify this flag.

MSG_DONTWAIT

Enables nonblocking operation; if the operation would block, **EAGAIN** or **EWOULDBLOCK** is returned. This provides similar behavior to setting the **O_NONBLOCK** flag (via the fcntl(2) **F_SETFL** operation), but differs in that **MSG_DONTWAIT** is a per-call option, whereas **O_NONBLOCK** is a setting on the open file description (see open(2)), which will affect all threads in the calling process and as well as other processes that hold file descriptors referring to the same open file description.

MSG_CONFIRM

Tell the link layer that forward progress happened: you got a successful reply from the other side. If the link layer doesn't get this it will regularly reprobe the neighbor (e.g., via a unicast ARP). Valid only on **SOCK_DGRAM** and **SOCK_RAW** sockets and currently implemented only for IPv4 and IPv6.

MSG_EOF

Requests that the sender side of asocket be shut down,and that an appropriate indication be sent at the end of the specified data; this flag is only implemented for SOCK_STREAM socket in the PF_INET protocol family.

IPC_NOWAIT

If the message queue is full, then the message is not written to the queue, and control is returned to the calling process. If not specified, then the calling process will suspend (block) until the message can be written.

Code implementation

Code for flag IPC_WAIT

```
#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];

} message_buf;

int 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
   /*
    * 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

msgget:calling msgget(ox4d2,01666)

msgget:msgget succeded :msqid=0

msgget:msgget succeded :msqid=0

Message:”Did you get this sent” sent

Code for flag MSG_OOB

```
#include <string.h>
#include <sys/socket.h>
#include <stdlib.h>
```

```
#include <stdio.h>
#include <netdb.h>
#include <unistd.h>
int my_connect(const char ip[], const char port[])
{


}
int main(int argc, char **argv)
{
   int sockfd;/* w  w   w.   d e  m   o2   s  .c  o  m  */
   if(argc != 3)
   {
      perror("usage: %s <host> <port>");
      exit(1);
   }
   sockfd = my_connect(argv[1], argv[2]);
   /*???????*/
   write(sockfd, "123", 3);
   printf("?????3????????:123\n");   /*sleep?????????????write?send?????????????TCP?????????*/
   sleep(1);
   /*???????????*/
send(sockfd, "4", 1, MSG_OOB);
   printf("?????1????????:4\n");
   sleep(1);
   write(sockfd, "56", 2);
```

```
   printf("?????2????????:56\n");
   sleep(1);
    send(sockfd,”7”,1,MSG_OOB);
   printf("?????1????????:7");
   sleep(1);
   write(sockfd, "89", 2);
   printf("?????2????????:89\n");
   sleep(1);
   exit(0);
}
```

OUT PUT

usage %s <host><port>

Is displayed.

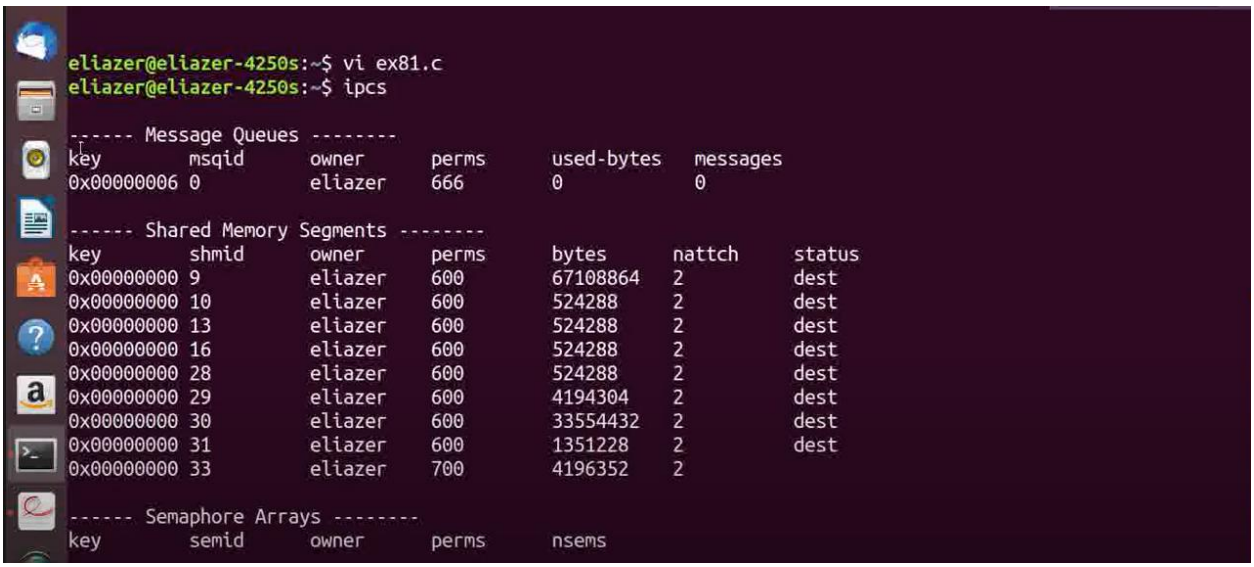
Threre is an other msgsnd system call I implemented by using Ubuntu.

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


int main(int argc ,char *argv[])
{
   int mid, len,i=1;
   struct buffer{
       long mtype;
       char buf[50];
   }x;
   /*
    * Create a message queue with the given key
    */
    mid=msget((key_t),IPC_CREAT|0666);
    X.mtype=atio(argv[1]);   //message type number
```

```
    Strcpy(x,buf,argv[2]);  //message text
    len=strlen(x.buf);
    msgsnd(mid,&x,len,0);  // four parametrs here
     printf("message of size %d sent successfully\n",len);

    return 0;
}
```

The output is shown in Ubuntu terminal.

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/msg.h>
int main(int argc, char *argv[])
{
  int len,mid,i=1;

  struct buffer
  {
    long mtype;
    char buf[50];
  } x;

  mid=msgget((key_t)6,IPC_CREAT|0666);
  x.mtype=atoi(argv[1]);              //message type number
  strcpy(x.buf,argv[2]);             // message text
  len=strlen(x.buf);
  msgsnd(mid,&x,len,0);
  printf("Message of size %d sent successfully \n",len);
  return 0;
}
```



```
eliazer@eliazer-4250s:~$ vi ex81.c
eliazer@eliazer-4250s:~$ ipcs

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x00000006 0          eliazer    666        0            0

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 9          eliazer    600        67108864   2          dest
0x00000000 10         eliazer    600        524288     2          dest
0x00000000 13         eliazer    600        524288     2          dest
0x00000000 16         eliazer    600        524288     2          dest
0x00000000 28         eliazer    600        524288     2          dest
0x00000000 29         eliazer    600        4194304    2          dest
0x00000000 30         eliazer    600        33554432   2          dest
0x00000000 31         eliazer    600        1351228    2          dest
0x00000000 33         eliazer    700        4196352    2

------ Semaphore Arrays --------
key        semid      owner      perms      nsems
```

Generally I write two code implementation one for the flag MSG_OOB and the other code is two show parameters in message send system call.

References

https://www.ibm.com/docs/en/zos/2.3.0?topic=functions-sendto-send-data-socket

https://man7.org/linux/man-pages/man2/send.2.html

https://www.tutorialspoint.com/unix_system_calls/msgop.htm

https://users.cs.cf.ac.uk/dave/C/node25.html

THANK YOU!