# Example program

``**msgop** system call example'' is a menu-driven program. It allows all possible combinations of using the **msgsnd** and **msgrcv** system calls to be exercised.

From studying this program, you can observe the method of passing arguments and receiving return values. The user-written program requirements are pointed out.

This program begins (lines 5-9) by including the required header files as specified on the **msgop**(S). Note that in this program **errno** is declared as an external variable; therefore, the *sys/errno.h* header file does not have to be included.

Variable and structure names have been chosen to be as close as possible to those in the synopsis. Their declarations are self explanatory. These names make the program more readable and are perfectly valid since they are local to the program.

The variables declared for this program and what they are used for are as follows:

**sndbuf**
: used as a buffer to contain a message to be sent (line 13); it uses the **msgbuf1** data structure as a template (lines 10-13). The **msgbuf1** structure (lines 10-13) is a duplicate of the **msgbuf** structure contained in the *sys/msg.h* header file, except that the size of the character array for **mtext** is tailored to fit this application. The **msgbuf** structure should not be used directly because **mtext** has only one element that would limit the size of each message to one character. Instead, declare your own structure. It should be identical to **msgbuf** except that the size of the **mtext** array should fit your application.

**rcvbuf**
: used as a buffer to receive a message (line 13); it uses the **msgbuf1** data structure as a template (lines 10-13)

**msgp**
: used as a pointer (line 13) to both the **sndbuf** and **rcvbuf** buffers

**i**
: used as a counter for inputing characters from the keyboard, storing them in the array, and keeping track of the message length for the **msgsnd** system call; it is also used as a counter to output the received message for the **msgrcv** system call

**c**
: used to receive the input character from the **getchar** function (line 50)

**flag**
: used to store the code of **IPC_NOWAIT** for the **msgsnd** system call (line 61)

**flags**
: used to store the code of the **IPC_NOWAIT** or **MSG_NOERROR** flags for the **msgrcv** system call (line 117)

**choice**
: used to store the code for sending or receiving (line 30)

**rtrn**

used to store the return values from all system calls

**msqid**
used to store and pass the desired message queue identifier for both system calls

**msgsz**
used to store and pass the **size** of the message to be sent or received

**msgflg**
used to pass the value of flag for sending or the value of flags for receiving

**msgtyp**
used for specifying the message type for sending or for picking a message type for receiving.

Note that a **msqid_ds** data structure is set up in the program (line 21) with a pointer initialized to point to it (line 22); this will allow the data structure members affected by message operations to be observed. They are observed by using the **msgctl** (**IPC_STAT**) system call to get them for the program to print them out (lines 80-92 and lines 160-167).

The first thing the program prompts for is whether to send or receive a message. A corresponding code must be entered for the desired operation; it is stored in the choice variable (lines 23-30). Depending upon the code, the program proceeds as in the following **msgsnd** or **msgrcv** sections.

# msgsnd

When the code is to send a message, the **msgp** pointer is initialized (line 33) to the address of the send data structure, **sndbuf**. Next, a message type must be entered for the message; it is stored in the variable **msgtyp** (line 42), and then (line 43) it is put into the **mtype** member of the data structure pointed to by **msgp**.

The program now prompts for a message to be entered from the keyboard and enters a loop of getting and storing into the **mtext** array of the data structure (lines 48-51). This will continue until an end-of-file is recognized which, for the **getchar** function, is a CTRL-d immediately following a carriage return (<Return>).

The message is immediately echoed from the **mtext** array of the **sndbuf** data structure to provide feedback (lines 54-56).

The next and final thing that must be decided is whether to set the **IPC_NOWAIT** flag. The program does this by requesting that a code of a 1 be entered for yes or anything else for no (lines 57-65). It is stored in the flag variable. If a 1 is entered, **IPC_NOWAIT** is logically ORed with **msgflg**; otherwise, **msgflg** is set to zero.

The **msgsnd** system call is performed (line 69). If it is unsuccessful, a failure message is displayed along with the error number (lines 70-72). If it is successful, the returned value is printed and should be zero (lines 73-76).

Every time a message is successfully sent, three members of the associated data structure are updated. They are:

**msg_qnum**
represents the total number of messages on the message queue; it is incremented by one.

**msg_lspid**
contains the process identification (PID) number of the last process sending a message; it is set accordingly.

**msg_stime**

contains the time in seconds since January 1, 1970, Greenwich Mean Time (GMT) of the last message sent; it is set accordingly.

These members are displayed after every successful message send operation (lines 79-92).

## msgrcv

When the code is to receive a message, the program continues execution as in the following paragraphs.

The **msgp** pointer is initialized to the **rcvbuf** data structure (line 99).

Next, the message queue identifier of the message queue from which to receive the message is requested; it is stored in **msqid** (lines 100-103).

The message type is requested; it is stored in **msgtyp** (lines 104-107).

The code for the desired combination of control flags is requested next; it is stored in flags (lines 108-117). Depending upon the selected combination, **msgflg** is set accordingly (lines 118-131).

Finally, the number of bytes to be received is requested; it is stored in **msgsz** (lines 132-135).

The **msgrcv** system call is performed (line 142). If it is unsuccessful, a message and error number is displayed (lines 143-145). If successful, a message indicates so, and the number of bytes returned and the **msg** type returned (because the
value returned may be different from the value requested) is displayed followed by the received message (lines 150-156).

When a message is successfully received, three members of the associated data structure are updated. They are:

`msg_qnum`
contains the number of messages on the message queue; it is decremented by one.

`msg_lrpid`
contains the PID of the last process receiving a message; it is set accordingly.

`msg_rtime`
contains the time in seconds since January 1, 1970, Greenwich Mean Time (GMT) that the last process received a message; it is set accordingly.

[``**msgop** system call example"] shows the **msgop** system calls. We suggest that you put the program into a source file called *msgop.c* and then compile it into an executable file called **msgop**.

```
 1    /*This is a program to illustrate
 2     *the message operations, msgop(),
 3     *system call capabilities.
 4     */


 5    /*Include necessary header files.*/
 6    #include    <stdio.h>
 7    #include    <sys/types.h>
 8    #include    <sys/ipc.h>
 9    #include    <sys/msg.h>


10    struct msgbuf1 {
11        long    mtype;
12        char    mtext[8192];
13    } sndbuf, rcvbuf, *msgp;
```

```
14    /*Start of main C language program*/
15    main()
16    {
17        extern int errno;
18        int i, c, flag, flags, choice;
19        int rtrn, msqid, msgsz, msgflg;
20        long mtype, msgtyp;
21        struct msqid_ds msqid_ds, *buf;
22        buf = & msqid_ds;


23        /*Select the desired operation.*/
24        printf("Enter the corresponding\n");
25        printf("code to send or\n");
26        printf("receive a message:\n");
27        printf("Send            =  1\n");
28        printf("Receive         =  2\n");
29        printf("Entry           =  ");
30        scanf("%d", &choice);


31        if(choice == 1) /*Send a message.*/
32        {
33            msgp = & sndbuf; /*Point to user send structure.*/


34            printf("\nEnter the msqid of\n");
35            printf("the message queue to\n");
36            printf("handle the message = ");
37            scanf("%d", &msqid);


38            /*Set the message type.*/
39            printf("\nEnter a positive integer\n");
40            printf("message type (long) for the\n");
41            printf("message = ");
42            scanf("%ld", &msgtyp);
43            msgp->mtype = msgtyp;


44            /*Enter the message to send.*/
45            printf("\nEnter a message: \n");


46            /*A control-d (^d) terminates as
47              EOF.*/


48            /*Get each character of the message
49              and put it in the mtext array.*/
50            for(i = 0; ((c = getchar()) != EOF); i++)
51                sndbuf.mtext[i] = c;


52            /*Determine the message size.*/
53            msgsz = i;


54            /*Echo the message to send.*/
55            for(i = 0; i < msgsz; i++)
56                putchar(sndbuf.mtext[i]);


57            /*Set the IPC_NOWAIT flag if
58              desired.*/
59            printf("\nEnter a 1 if you want \n");
60            printf("the IPC_NOWAIT flag set:  ");
61            scanf("%d", &flag);
62            if(flag == 1)
63                msgflg = IPC_NOWAIT;
```

```
64            else
65                msgflg = 0;


66            /*Check the msgflg.*/
67            printf("\nmsgflg = 0%o\n", msgflg);


68            /*Send the message.*/
69            rtrn = msgsnd(msqid, (const void*) msgp, msgsz, msgflg);
70            if(rtrn == -1)
71            printf("\nMsgsnd failed.  Error = %d\n",
72                   errno);
73            else {
74                /*Print the value of test which
75                      should be zero for successful.*/
76                printf("\nValue returned = %d\n", rtrn);


77                /*Print the size of the message
78                  sent.*/
79                printf("\nMsgsz = %d\n", msgsz);


80                /*Check the data structure update.*/
81                msgctl(msqid, IPC_STAT, buf);


82                /*Print out the affected members.*/


83                /*Print the incremented number of
84                  messages on the queue.*/
85                printf("\nThe msg_qnum = %d\n",
86                    buf->msg_qnum);
87                /*Print the process id of the last sender.*/
88                printf("The msg_lspid = %d\n",
89                    buf->msg_lspid);
90                /*Print the last send time.*/
91                printf("The msg_stime = %d\n",
92                    buf->msg_stime);
93            }
94        }


95        if(choice == 2)  /*Receive a message.*/
96        {
97            /*Initialize the message pointer
98              to the receive buffer.*/
99            msgp = & rcvbuf;


100           /*Specify the message queue which contains
101                  the desired message.*/
102           printf("\nEnter the msqid = ");
103           scanf("%d", &msqid);


104           /*Specify the specific message on the queue
105                 by using its type.*/
106           printf("\nEnter the msgtyp = ");
107           scanf("%ld", &msgtyp);


108           /*Configure the control flags for the
109                 desired actions.*/
110           printf("\nEnter the corresponding code\n");
111           printf("to select the desired flags: \n");
112           printf("No flags                    =  0\n");
113           printf("MSG_NOERROR                 =  1\n");
114           printf("IPC_NOWAIT                  =  2\n");
```

```
115             printf("MSG_NOERROR and IPC_NOWAIT  =  3\n");
116             printf("                Flags       =  ");
117             scanf("%d", &flags);


118             switch(flags) {
119             case 0:
120                 msgflg = 0;
121                 break;
122             case 1:
123                 msgflg = MSG_NOERROR;
124                 break;
125             case 2:
126                 msgflg = IPC_NOWAIT;
127                 break;
128             case 3:
129                 msgflg = MSG_NOERROR | IPC_NOWAIT;
130                 break;
131             }


132             /*Specify the number of bytes to receive.*/
133             printf("\nEnter the number of bytes\n");
134             printf("to receive (msgsz) = ");
135             scanf("%d", &msgsz);


136             /*Check the values for the arguments.*/
137             printf("\nmsqid =%d\n", msqid);
138             printf("\nmsgtyp = %ld\n", msgtyp);
139             printf("\nmsgsz = %d\n", msgsz);
140             printf("\nmsgflg = 0%o\n", msgflg);


141             /*Call msgrcv to receive the message.*/
142             rtrn = msgrcv(msqid, (void*), msgp, msgsz, msgtyp, msgflg);


143             if(rtrn == -1)  {
144                 printf("\nMsgrcv failed., Error = %d\n", errno);
145             }
146             else {
147                 printf ("\nMsgctl was successful\n");
148                 printf("for msqid = %d\n",
149                     msqid);


150                 /*Print the number of bytes received,
151                   it is equal to the return
152                   value.*/
153                 printf("Bytes received = %d\n", rtrn);


154                 /*Print the received message.*/
155                 for(i = 0; i<rtrn; i++)
156                     putchar(rcvbuf.mtext[i]);
157             }
158             /*Check the associated data structure.*/
159             msgctl(msqid, IPC_STAT, buf);
160             /*Print the decremented number of messages.*/
161             printf("\nThe msg_qnum = %d\n", buf->msg_qnum);
162             /*Print the process id of the last receiver.*/
163             printf("The msg_lrpid = %d\n", buf->msg_lrpid);
164             /*Print the last message receive time*/
165             printf("The msg_rtime = %d\n", buf->msg_rtime);
166         }
167     }
```

**msgop system call example**

*Next topic: Semaphores*
*Previous topic: Receiving messages*


*SCO OpenServer Release 6.0.0 -- 02 June 2005*