Controlling message queues

This section describes how to use the **msgctl** system call. The accompanying program illustrates its use.

Using msgctl

The synopsis found on the **msgctl**(S) manual page is as follows:

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



int msgctl (msqid, cmd, buf)
int msqid, cmd;
struct msqid_ds *buf;
```

The **msgctl** system call requires three arguments to be passed to it; it returns an integer-type value.

When successful, it returns a zero value; when unsuccessful, it returns a **-1**.

The **msqid** variable must be a valid, non-negative, integer value. In other words, it must have already been created by using the **msgget** system call.

The **cmd** argument can be any one of the following values:

IPC_STAT
: return the status information contained in the associated data structure for the specified message queue identifier, and place it in the data structure pointed to by the **buf** pointer in the user memory area.

IPC_SET
: for the specified message queue identifier, set the effective user and group identification, operation permissions, and the number of bytes for the message queue to the values contained in the data structure pointed to by the **buf** pointer in the user memory area.

IPC_RMID
: remove the specified message queue identifier along with its associated message queue and data structure.

To perform an **IPC_SET** or **IPC_RMID** control command, a process must have:

- an effective user id of OWNER/CREATOR, or
- an effective user id of *root* (if the system is running with the SUM privilege module), or
- the **P_OWNER** privilege.

Read permission is required to perform the **IPC_STAT** control command.

The details of this system call are discussed in the following example program. If you need more information on the logic manipulations in this program, read the **msgget**(S) manual page.

Example program

``**msgctl** system call example'' is a menu-driven program. It allows all possible combinations of using the **msgctl** system call 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 **msgctl**(S) manual page. Note in this program that **errno** is declared as an external variable, and 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 for the system call. 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:

uid
used to store the **IPC_SET** value for the effective user identification

gid
used to store the **IPC_SET** value for the effective group identification

mode
used to store the **IPC_SET** value for the operation permissions

bytes
used to store the **IPC_SET** value for the number of bytes in the message queue (**msg_qbytes**)

rtrn
used to store the return integer value from the system call

msqid
used to store and pass the message queue identifier to the system call

command
used to store the code for the desired control command so that subsequent processing can be performed on it

choice
used to determine which member is to be changed for the **IPC_SET** control command

msqid_ds
used to receive the specified message queue identifier's data structure when an **IPC_STAT** control command is performed

buf
a pointer passed to the system call which locates the data structure in the user memory area where the **IPC_STAT** control command is to place its return values or where the **IPC_SET** command gets the values to set

Note that the **msqid_ds** data structure in this program (line 16) uses the data structure, located in the *sys/msg.h* header file of the same name, as a template for its declaration.

The next important thing to observe is that although the **buf** pointer is declared to be a pointer to a data structure of the **msqid_ds** type, it must also be initialized to contain the address of the user memory area data structure (line 17). Now that all of the required declarations have been explained for this program, this is how it works.

First, the program prompts for a valid message queue identifier which is stored in the **msqid** variable (lines 19, 20). This is required for every **msgctl** system call.

Then the code for the desired control command must be entered (lines 21-27) and stored in the command variable. The code is tested to determine the control command for subsequent processing.

If the **IPC_STAT** control command is selected (code 1), the system call is performed (lines 37, 38) and the status information returned is printed out (lines 39-46); only the members that can be set are printed out in this program. Note that if the system call is unsuccessful (line 106), the status information of the last successful call is printed out. In addition, an error message is displayed and the **errno** variable is printed out (line 108). If the system call is successful, a message indicates this along with the message queue identifier used (lines 110-113).

If the **IPC_SET** control command is selected (code 2), the first thing is to get the current status information for the message queue identifier specified (lines 50-52). This is necessary because this example program provides for changing only one member at a time, and the system call changes all of them. Also, if an invalid value happened to be stored in the user memory area for one of these members, it would cause repetitive failures for this control command until corrected. The next thing the program does is to prompt for a code corresponding to the member to be changed (lines 53-59). This code is stored in the choice variable (line 60). Now, depending upon the member picked, the program prompts for the new value (lines 66-95). The value is placed into the appropriate member in the user memory area data structure, and the system call is made (lines 96-98). Depending upon success or failure, the program returns the same messages as for **IPC_STAT** above.

If the **IPC_RMID** control command (code 3) is selected, the system call is performed (lines 100-103), and the **msqid** along with its associated message queue and data structure are removed from the SCO OpenServer operating system. Note that the **buf** pointer is ignored in performing this control command, and its value can be zero or NULL. Depending upon the success or failure, the program returns the same messages as for the other control commands.

The example program for the **msgctl** system call follows. We suggest that you name the source program file *msgctl.c* and the executable file **msgctl**.

```
 1    /*This is a program to illustrate
 2     *the message control, msgctl(),
 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    /*Start of main C language program*/
11    main()
12    {
13        extern int errno;
14        int mode, bytes;
15        uid_t uid;
16        gid_t gid;
17        int rtrn, msqid, command, choice;
18        struct msqid_ds msqid_ds, *buf;
19        buf = & msqid_ds;


20        /*Get the msqid, and command.*/
21        printf("Enter the msqid = ");
22        scanf("%d", &msqid);
23        printf("\nEnter the number for\n");
24        printf("the desired command:\n");
25        printf("IPC_STAT    =  1\n");
26        printf("IPC_SET     =  2\n");
```

```
27        printf("IPC_RMID    =  3\n");
28        printf("Entry       =  ");
29        scanf("%d", &command);


30        /*Check the values.*/
31        printf ("\nmsqid =%d, command = %d\n",
32            msqid, command);


33        switch (command)
34        {
35        case 1:    /*Use msgctl() to duplicate
36              the data structure for
37                      msqid in the msqid_ds area pointed
38                      to by buf and then print it out.*/
39            rtrn = msgctl(msqid, IPC_STAT,
40                buf);
41            printf ("\nThe USER ID = %d\n",
42                buf->msg_perm.uid);
43            printf ("The GROUP ID = %d\n",
44                buf->msg_perm.gid);
45            printf ("The operation permissions = 0%o\n",
46                buf->msg_perm.mode);
47            printf ("The msg_qbytes = %d\n",
48                buf->msg_qbytes);
49            break;
50        case 2:    /*Select and change the desired
51                      member(s) of the data structure.*/
52            /*Get the original data for this msqid
53                  data structure first.*/
54            rtrn = msgctl(msqid, IPC_STAT, buf);
55            printf("\nEnter the number for the\n");
56            printf("member to be changed:\n");
57            printf("msg_perm.uid   = 1\n");
58            printf("msg_perm.gid   = 2\n");
59            printf("msg_perm.mode  = 3\n");
60            printf("msg_qbytes     = 4\n");
61            printf("Entry          = ");


62            scanf("%d", &choice);
63            /*Only one choice is allowed per
64              pass as an invalid entry will
65                  cause repetitive failures until
66              msqid_ds is updated with
67                  IPC_STAT.*/


68            switch(choice){
69            case 1:
70                printf("\nEnter USER ID = ");
71                scanf ("%ld", &uid);
72                buf->msg_perm.uid =(uid_t)uid;
73                printf("\nUSER ID = %d\n",
74                    buf->msg_perm.uid);
75                break;
76            case 2:
77                printf("\nEnter GROUP ID = ");
78                scanf("%d", &gid);
79                buf->msg_perm.gid = gid;
80                printf("\nGROUP ID = %d\n",
81                    buf->msg_perm.gid);
82                break;
83            case 3:
84                printf("\nEnter MODE = ");
85                scanf("%o", &mode);
86                buf->msg_perm.mode = mode;
87                printf("\nMODE = 0%o\n",
88                    buf->msg_perm.mode);
```

```
 89              break;
 90          case 4:
 91              printf("\nEnter msq_bytes = ");
 92              scanf("%d", &bytes);
 93              buf->msg_qbytes = bytes;
 94              printf("\nmsg_qbytes = %d\n",
 95                  buf->msg_qbytes);
 96              break;
 97          default: /* Invalid Input */
 98              exit(-1);
 99          }


100          /*Do the change.*/
101          rtrn = msgctl(msqid, IPC_SET,
102              buf);
103          break;


104      case 3:    /*Remove the msqid along with its
105                     associated message queue
106                     and data structure.*/
107          rtrn = msgctl(msqid, IPC_RMID, (struct msqid_ds *) NULL);
108      break;
109      default:     /* Invalid Input */
110          exit(-1);
111      }
112      /*Perform the following if the call is unsuccessful.*/
113      if(rtrn == -1)
114      {
115          printf ("\nThe msgctl call failed, error number = %d\n", errno);
116      }
117      /*Return the msqid upon successful completion.*/
118      else
119          printf ("\nMsgctl was successful for msqid = %d\n",
120              msqid);
121      exit (0);
122  }
```

msgctl system call example

Next topic: Operations for messages
*Previous topic: **msgget** system call example*

SCO OpenServer Release 6.0.0 -- 02 June 2005