Getting message queues

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

Using msgget

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

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


int  msgget (key_t key, int msgflg);
```

All of these **include** files are located in the */usr/include/sys* directory of the SCO OpenServer operating system.

The following line in the synopsis:

```
int msgget (key_t key, int msgflg);
```

informs you that **msgget** is a function that returns an integer-type value. It also declares the types of the two formal arguments: **key** is of type **key_t**, and **msgflg** is of type **int**. **key_t** is defined by a **typedef** in the *sys/types.h* header file to be an integral type.

The integer returned from this function upon successful completion is the message queue identifier that was discussed earlier. Upon failure, the external variable **errno** is set to indicate the reason for failure, and the value **-1** (which is not a valid **msqid**) is returned.

As declared, the process calling the **msgget** system call must supply two arguments to be passed to the formal **key** and **msgflg** arguments.

A new **msqid** with an associated message queue and data structure is provided if either

- **key** is equal to **IPC_PRIVATE**,

or

- **key** is a unique integer and the control command **IPC_CREAT** is specified in the **msgflg** argument.

The value passed to the **msgflg** argument must be an integer-type value that will specify the following:

- operations permissions
- control fields (commands)

Operation permissions determine the operations that processes are permitted to perform on the associated message queue. ``Read'' permission is necessary for receiving messages or for determining queue status by means of a **msgctl IPC_STAT** operation. ``Write'' permission is necessary for sending messages.

The following table reflects the numeric values (expressed in octal notation) for the valid operation permissions codes.

Operation permissions codes

Operation permissions	Octal value
Read by user	00400
Write by user	00200
Read by group	00040
Write by group	00020
Read by others	00004
Write by others	00002

A specific value is derived by adding or bitwise ORing the octal values for the operation permissions wanted. That is, if read by user and read/write by others is desired, the code value would be 00406 (00400 plus 00006). There are constants located in the *sys/msg.h* header file which can be used for the user operations permissions. They are as follows:

```
MSG_W   0200    /* write permissions by owner */


MSG_R   0400    /* read permissions by owner */
```

Control flags are predefined constants (represented by all upper-case letters). The flags which apply to the **msgget** system call are **IPC_CREAT** and **IPC_EXCL** and are defined in the *sys/ipc.h* header file.

The value for **msgflg** is therefore a combination of operation permissions and control commands. After determining the value for the operation permissions as previously described, the desired flag(s) can be specified. This is accomplished by adding or bitwise ORing (|) them with the operation permissions; the bit positions and values for the control commands in relation to those of the operation permissions make this possible.

The **msgflg** value can easily be set by using the flag names in conjunction with the octal operation permissions value:

```
msqid = msgget (key, (IPC_CREAT | 0400));


msqid = msgget (key, (IPC_CREAT | IPC_EXCL | 0400));
```

As specified by the **msgget**(S) manual page, success or failure of this system call depends upon the argument values for **key** and **msgflg** or system-tunable parameters. The system call will attempt to return a new message queue identifier if one of the following conditions is true:

- **key** is equal to **IPC_PRIVATE**
- **key** does not already have a message queue identifier associated with it and (**msgflg** and **IPC_CREAT**) is ``true" (not zero).

The **key** argument can be set to **IPC_PRIVATE** like this:

```
msqid = msgget (IPC_PRIVATE, msgflg);
```

The system call will always be attempted. Exceeding the **MSGMNI** system-tunable parameter always causes a failure. The **MSGMNI** system-tunable parameter determines the systemwide number of unique message queues that may be in use at any given time.

IPC_EXCL is another control command used in conjunction with **IPC_CREAT**. It will cause the system call to return an error if a message queue identifier already exists for the specified **key.** This is necessary to prevent the process from thinking that it has received a new identifier when it has not. In other words, when both **IPC_CREAT** and **IPC_EXCL** are specified, a new message queue identifier is returned if the system call is successful.

Refer to the **msgget**(S) manual page for specific, associated data structure initialization for successful completion. The specific failure conditions and their error names are contained there also.

Example program

``**msgget** system call example'' is a menu-driven program. It allows all possible combinations of using the **msgget** 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 4-8) by including the required header files as specified by **msgget**(S). Note that the *sys/errno.h* header file is included as opposed to declaring **errno** as an external variable; either method will work.

Variable 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 programs more readable 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:

key
: used to pass the value for the desired **key**

opperm
: used to store the desired operation permissions

flags
: used to store the desired control commands (flags)

opperm_flags
: used to store the combination from the logical ORing of the **opperm** and **flags** variables; it is then used in the system call to pass the **msgflg** argument

msqid
: used for returning the message queue identification number for a successful system call or the error code (**-1**) for an unsuccessful one.

The program begins by prompting for a hexadecimal **key**, an octal operation permissions code, and finally for the control command combinations (flags) which are selected from a menu (lines 15-32). All possible combinations are allowed even though they might not be viable. This allows errors to be observed for invalid combinations.

Next, the menu selection for the flags is combined with the operation permissions, and the result is stored in the **opperm_flags** variable (lines 36-51).

The system call is made next, and the result is stored in the **msqid** variable (line 53).

Since the **msqid** variable now contains a valid message queue identifier or the error code (**-1**), it is tested to see if an error occurred (line 55). If **msqid** equals **-1**, a message indicates that an error resulted, and the external **errno** variable is displayed (line 57).

If no error occurred, the returned message queue identifier is displayed (line 61).

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

```
 1    /*This is a program to illustrate
 2     *the message get, msgget(),
 3     *system call capabilities*/


 4    #include    <stdio.h>
 5    #include    <sys/types.h>
 6    #include    <sys/ipc.h>
 7    #include    <sys/msg.h>
 8    #include    <errno.h>


 9    /*Start of main C language program */
10    main()
11    {
12        key_t key;
13        int opperm, flags;
14        int msqid, opperm_flags;
15        /*Enter the desired key*/
16        printf("Enter the desired key in hex = ");
17        scanf("%x", &key);


18        /*Enter the desired octal operation
19          permissions.*/
20        printf("\nEnter the operation\n");
21        printf("permissions in octal = ");
22        scanf("%o", &opperm);


23        /*Set the desired flags.*/
24        printf("\nEnter corresponding number to\n");
25        printf("set the desired flags:\n");
26        printf("No flags                  = 0\n");
27        printf("IPC_CREAT                 = 1\n");
28        printf("IPC_EXCL                  = 2\n");
29        printf("IPC_CREAT and IPC_EXCL    = 3\n");
30        printf("            Flags         = ");


31        /*Get the flag(s) to be set.*/
32        scanf("%d", &flags);


33        /*Check the values.*/
34        printf ("\nkey =0x%x, opperm = 0%o, flags = 0%o\n",
35            key, opperm, flags);


36        /*Incorporate the control fields (flags) with
37          the operation permissions*/
38        switch (flags)
39        {
40        case 0:    /*No flags are to be set.*/
41            opperm_flags = (opperm | 0);
42            break;
43        case 1:    /*Set the IPC_CREAT flag.*/
44            opperm_flags = (opperm | IPC_CREAT);
45            break;
46        case 2:    /*Set the IPC_EXCL flag.*/
47            opperm_flags = (opperm | IPC_EXCL);
48            break;
49        case 3:    /*Set the IPC_CREAT and IPC_EXCL flags.*/
50            opperm_flags = (opperm | IPC_CREAT | IPC_EXCL);
51            break;
52        default:       /* Invalid Input */
53            exit(-1);
54        }
```

```
55        /*Call the msgget system call.*/
56        msqid = msgget (key, opperm_flags);


57        /*Perform the following if the call is unsuccessful.*/
58        if(msqid == -1)
59        {
60            printf ("\nThe msgget call failed, error number = %d\n", errno);
61        }
62        /*Return the msqid upon successful completion.*/
63        else
64            printf ("\nThe msqid = %d\n", msqid);
65        exit(0);
66    }
```

msgget system call example

Next topic: Controlling message queues
*Previous topic: **ipc_perm** data structure*

SCO OpenServer Release 6.0.0 -- 02 June 2005