Example program

``**shmctl** system call example'' is a menu-driven program. It allows all possible combinations of using the **shmctl** 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 by the **shmctl**(S) manual page. Note that in this program **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 user identification

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

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

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

shmid
: used to store and pass the shared memory segment 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 for the **IPC_SET** control command is to be changed

shmid_ds
: used to receive the specified shared memory segment 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 **shmid_ds** data structure in this program (line 16) uses the data structure of the same name located in the **sys/shm.h** header file 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 **shmid_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 shared memory segment identifier which is stored in the **shmid** variable (lines 18-20). This is required for every **shmctl** system call.

Then, the code for the desired control command must be entered (lines 21-29); it is 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 39, 40) and the status information returned is printed out (lines 41-71). Note that if the system call is unsuccessful (line 139), 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 (lines 141). If the system call is successful, a message indicates this along with the shared memory segment identifier used (lines 143-147).

If the **IPC_SET** control command is selected (code 2), the first thing done is to get the current status information for the shared memory identifier specified (lines 88-90). 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 91-96). This code is stored in the choice variable (line 97). Now, depending upon the member picked, the program prompts for the new value (lines 98-120). The value is placed in the appropriate member in the user memory area data structure, and the system call is made (lines 121-128). 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 125-128), and the **shmid** 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.

If the **SHM_LOCK** control command (code 4) is selected, the system call is performed (lines 130,131). Depending upon the success or failure, the program returns the same messages as for the other control commands.

If the **SHM_UNLOCK** control command (code 5) is selected, the system call is performed (lines 133-135). Depending upon the success or failure, the program returns the same messages as for the other control commands.

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

```
  1    /*This is a program to illustrate
  2     *the shared memory control, shmctl(),
  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/shm.h>


 10    /*Start of main C language program*/
 11    main()
 12    {
```

```
 13        extern int errno;
 14        int uid, gid, mode;
 15        int rtrn, shmid, command, choice;
 16        struct shmid_ds shmid_ds, *buf;
 17        buf = & shmid_ds;


 18        /*Get the shmid, and command.*/
 19        printf("Enter the shmid = ");
 20        scanf("%d", &shmid);
 21        printf("\nEnter the number for\n");
 22        printf("the desired command:\n");


 23        printf("IPC_STAT    =  1\n");
 24        printf("IPC_SET     =  2\n");
 25        printf("IPC_RMID    =  3\n");
 26        printf("SHM_LOCK    =  4\n");
 27        printf("SHM_UNLOCK  =  5\n");
 28        printf("Entry       =  ");
 29        scanf("%d", &command);


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


 33        switch (command)
 34        {
 35        case 1:    /*Use shmctl() to get
 36                   the data structure for
 37                   shmid in the shmid_ds area pointed
 38                   to by buf and then print it out.*/
 39            rtrn = shmctl(shmid, IPC_STAT,
 40                buf);
 41            printf ("\nThe USER ID = %d\n",
 42                buf->shm_perm.uid);
 43            printf ("The GROUP ID = %d\n",
 44                buf->shm_perm.gid);
 45            printf ("The creator's ID = %d\n",
 46                buf->shm_perm.cuid);
 47            printf ("The creator's group ID = %d\n",
 48                buf->shm_perm.cgid);
 49            printf ("The operation permissions = 0%o\n",
 50                buf->shm_perm.mode);
 51            printf ("The slot usage sequence\n");
 52            printf ("number = 0%x\n",
 53                buf->shm_perm.seq);
 54            printf ("The key= 0%x\n",
 55                buf->shm_perm.key);
 56            printf ("The segment size = %d\n",
 57                buf->shm_segsz);
 58            printf ("The pid of last shmop = %d\n",
 59                buf->shm_lpid);
 60            printf ("The pid of creator = %d\n",
 61                buf->shm_cpid);
 62            printf ("The current # attached = %d\n",
 63                buf->shm_nattch);
 64            printf("The last shmat time = %ld\n",
 65                buf->shm_atime);
 66            printf("The last shmdt time = %ld\n",
 67                buf->shm_dtime);
 68            printf("The last change time = %ld\n",
 69                buf->shm_ctime);
 70            break;


           /* Lines 71 - 85 deleted */
```

```
        case 2:    /*Select and change the desired
                      member(s) of the data structure.*/


            /*Get the original data for this shmid
                  data structure first.*/
            rtrn = shmctl(shmid, IPC_STAT, buf);


            printf("\nEnter the number for the\n");
            printf("member to be changed:\n");
            printf("shm_perm.uid   = 1\n");
            printf("shm_perm.gid   = 2\n");
            printf("shm_perm.mode  = 3\n");
            printf("Entry          = ");
            scanf("%d", &choice);


            switch(choice){
            case 1:
                printf("\nEnter USER ID = ");
                scanf ("%d", &uid);
                buf->shm_perm.uid = uid;
                printf("\nUSER ID = %d\n",
                    buf->shm_perm.uid);
                break;


            case 2:
                printf("\nEnter GROUP ID = ");
                scanf("%d", &gid);
                buf->shm_perm.gid = gid;
                printf("\nGROUP ID = %d\n",
                    buf->shm_perm.gid);
                break;


            case 3:
                printf("\nEnter MODE in octal = ");
                scanf("%o", &mode);
                buf->shm_perm.mode = mode;
                printf("\nMODE = 0%o\n",
                    buf->shm_perm.mode);
                break;
            }
            /*Do the change.*/
            rtrn = shmctl(shmid, IPC_SET,
                buf);
            break;


        case 3:    /*Remove the shmid along with its
                      associated
                      data structure.*/
            rtrn = shmctl(shmid, IPC_RMID, (struct shmid_ds *) NULL);
            break;


        case 4: /*Lock the shared memory segment*/
            rtrn = shmctl(shmid, SHM_LOCK, (struct shmid_ds *) NULL);
            break;
        case 5: /*Unlock the shared memory
                      segment.*/
            rtrn = shmctl(shmid, SHM_UNLOCK, (struct shmid_ds *) NULL);
            break;
        }
        /*Perform the following if the call is unsuccessful.*/
        if(rtrn == -1)
        {
```

```
       printf ("\nThe shmctl call failed, error number = %d\n", errno);
    }
    /*Return the shmid upon successful completion.*/
    else
        printf ("\nShmctl was successful for shmid = %d\n",
            shmid);
    exit (0);
}
```

shmctl system call example

Next topic: Operations for shared memory
Previous topic: Using shmctl

SCO OpenServer Release 6.0.0 -- 02 June 2005