Dean Cosanella
Flavio dos Santos-Ross
CSE 460
Dr. Tong Lai Yu
Lab 6 Report

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

1. Message Queues

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

int  msgctl(int msqid, int cmd, struct msqid_ds *buf);
int  msgget(key_t key, int msgflg);
int  msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg);
int  msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

Use "man" to study each of the message queue functions. In your lab report, briefly describe each of them and its usage.

msgctl:
System V message control operations

msgctl() performs the control operation specified by cmd on the System V message queue with identifier msqid.

msgget:
get a System V message queue identifier

The msgget() system call returns the System V message queue identifier associated with the value of the key argument. A new message queue is created if key has the value IPC_PRIVATE or key isn't IPC_PRIVATE, no message queue with the given key key exists, and IPC_CREAT is specified in msgflg.

If msgflg specifies both IPC_CREAT and IPC_EXCL and a message queue already exists for key, then msgget() fails with errno set to EEXIST. (This is analogous to the effect of the combination O_CREAT | O_EXCL for open(2).)

Upon creation, the least significant bits of the argument msgflg define the permissions of the message queue. These permission bits have the same format and semantics as the permissions specified for the mode argument of open(2). (The execute permissions are not used.)

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

msgrcv:
System V message queue operations

The msgsnd() and msgrcv() system calls are used, respectively, to send messages to, and receive messages from, a System V message queue. The calling process must have write permission on the message queue in order to send a message, and read permission to receive a message.

The msgp argument is a pointer to caller-defined structure of the following general form:

```
struct msgbuf {
   long mtype;       /* message type, must be > 0 */
   char mtext[1];    /* message data */
};
```

The mtext field is an array (or other structure) whose size is specified by msgsz, a nonnegative integer value. Messages of zero length (i.e., no mtext field) are permitted. The mtype field must have a strictly positive integer value. This value can be used by the receiving process for message selection (see the description of msgrcv() below).

msgsnd:
System V message queue operations

The msgsnd() system call appends a copy of the message pointed to by msgp to the message queue whose identifier is specified by msqid.

If sufficient space is available in the queue, msgsnd() succeeds immediately. (The queue capacity is defined by the msg_qbytes field in the associated data structure for the message queue. During queue creation this field is initialized to MSGMNB bytes, but this limit can be modified using msgctl(2).) If insufficient space is available in the queue, then the default behavior of msgsnd() is to block until space becomes available. If IPC_NOWAIT is specified in msgflg, then the call instead fails with the error EAGAIN.

A blocked msgsnd() call may also fail if:

* the queue is removed, in which case the system call fails with errno set to EIDRM; or

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

> * a signal is caught, in which case the system call fails with errno set to EINTR;see signal(7). (msgsnd() is never automatically restarted after being interrupted by a signal handler, regardless of the setting of the SA_RESTART flag when establishing a signal handler.)

Try the following two programs, **msg1.cpp**, which receives messages and **msg2.cpp**, which sends messages. Either of the program is allowed to create the message queue, but the receiver is responsible for deleting it after it receives the last message. You may compile and run them by:

```
$ g++ -o msg1 msg1.cpp
$ g++ -o msg2 msg2.cpp
$ ./msg1
$ ./msg2
```

Note that when you run the two programs, you should run them in two different windows (terminals). You should be able to send messages from one to the other and terminate them by entering "end".

Msg1.cpp code:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>

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


struct my_msg_st {
    long int my_msg_type;
    char some_text[BUFSIZ];
};

int main()
{
    int running = 1;
    int msgid;
    struct my_msg_st some_data;
    long int msg_to_receive = 0;
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

```
/* First, we set up the message queue. */

   msgid = msgget((key_t)1234, 0666 | IPC_CREAT);

   if (msgid == -1) {
      fprintf(stderr, "msgget failed with error: %d\n", errno);
      exit(EXIT_FAILURE);
   }
/* Then the messages are retrieved from the queue, until an end message is encountered.
 Lastly, the message queue is deleted. */

   while(running) {
      if (msgrcv(msgid, (void *)&some_data, BUFSIZ,
                msg_to_receive, 0) == -1) {
         fprintf(stderr, "msgrcv failed with error: %d\n", errno);
         exit(EXIT_FAILURE);
      }
      printf("You wrote: %s", some_data.some_text);
      if (strncmp(some_data.some_text, "end", 3) == 0) {
         running = 0;
      }
   }

   if (msgctl(msgid, IPC_RMID, 0) == -1) {
      fprintf(stderr, "msgctl(IPC_RMID) failed\n");
      exit(EXIT_FAILURE);
   }

   exit(EXIT_SUCCESS);
}
```

Msg2.cpp code:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>

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

#define MAX_TEXT 512
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

```
struct my_msg_st {
    long int my_msg_type;
    char some_text[MAX_TEXT];
};

int main()
{
    int running = 1;
    struct my_msg_st some_data;
    int msgid;
    char buffer[BUFSIZ];

    msgid = msgget((key_t)1234, 0666 | IPC_CREAT);

    if (msgid == -1) {
        fprintf(stderr, "msgget failed with error: %d\n", errno);
        exit(EXIT_FAILURE);
    }

    while(running) {
        printf("Enter some text: ");
        fgets(buffer, BUFSIZ, stdin);
        some_data.my_msg_type = 1;
        strcpy(some_data.some_text, buffer);

        if (msgsnd(msgid, (void *)&some_data, MAX_TEXT, 0) == -1) {
            fprintf(stderr, "msgsnd failed\n");
            exit(EXIT_FAILURE);
        }
        if (strncmp(buffer, "end", 3) == 0) {
            running = 0;
        }
    }

    exit(EXIT_SUCCESS);
}
```

Programs running:

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

Programs terminate by typing “end”

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

Second part:
msg3.cpp:

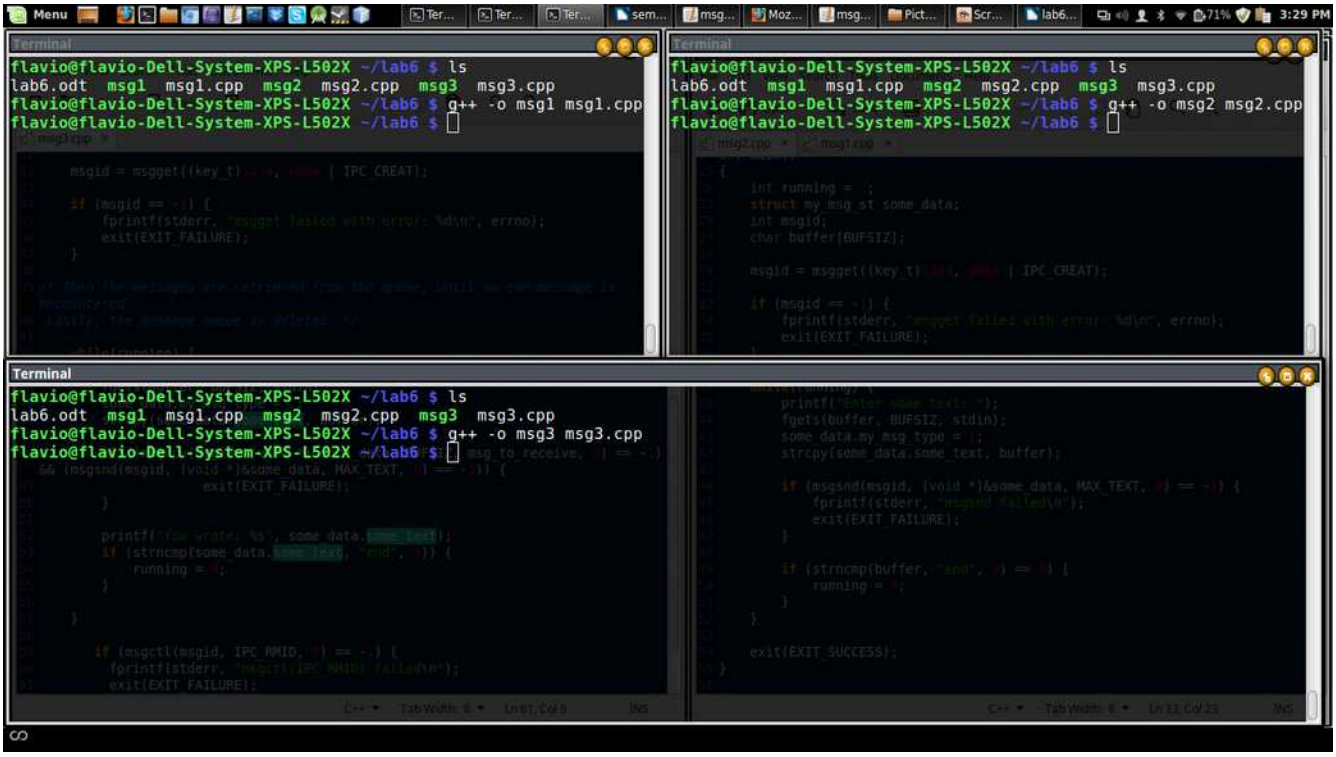
```
//msg3.cpp
/* Here's the receiver program. */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>

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

#define MAX_TEXT 512

struct my_msg_st {
    long int my_msg_type;
    char some_text[BUFSIZ];
   // char some_text2[MAX_TEXT];
 };
```

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

```
int main()
{
   int running = 1;
   int msgid;
   struct my_msg_st some_data;
   long int msg_to_receive = 0;
   char buffer[BUFSIZ];
/* First, we set up the message queue. */

   msgid = msgget((key_t)1234, 0666 | IPC_CREAT);

   if (msgid == -1) {
      fprintf(stderr, "msgget failed with error: %d\n", errno);
      exit(EXIT_FAILURE);
   }

/* Then the messages are retrieved from the queue, until an end message is encountered.
 Lastly, the message queue is deleted. */

   while(running) {
      printf("Enter some text: ");
      fgets(buffer, BUFSIZ, stdin);
      some_data.my_msg_type = 1;
      strcpy(some_data.some_text, buffer);

      if ((msgrcv(msgid, (void *)&some_data, BUFSIZ, msg_to_receive, 0) == -1) && (msgsnd(msgid,
(void *)&some_data, MAX_TEXT, 0) == -1)) {
         exit(EXIT_FAILURE);
      }

     // printf("You wrote: %s", some_data.some_text);
      if (strncmp(some_data.some_text, "end", 3)) {
         running = 0;
      }

   }

     if (msgctl(msgid, IPC_RMID, 0) == -1) {
          fprintf(stderr, "msgctl(IPC_RMID) failed\n");
          exit(EXIT_FAILURE);
  }

   exit(EXIT_SUCCESS);
}
```

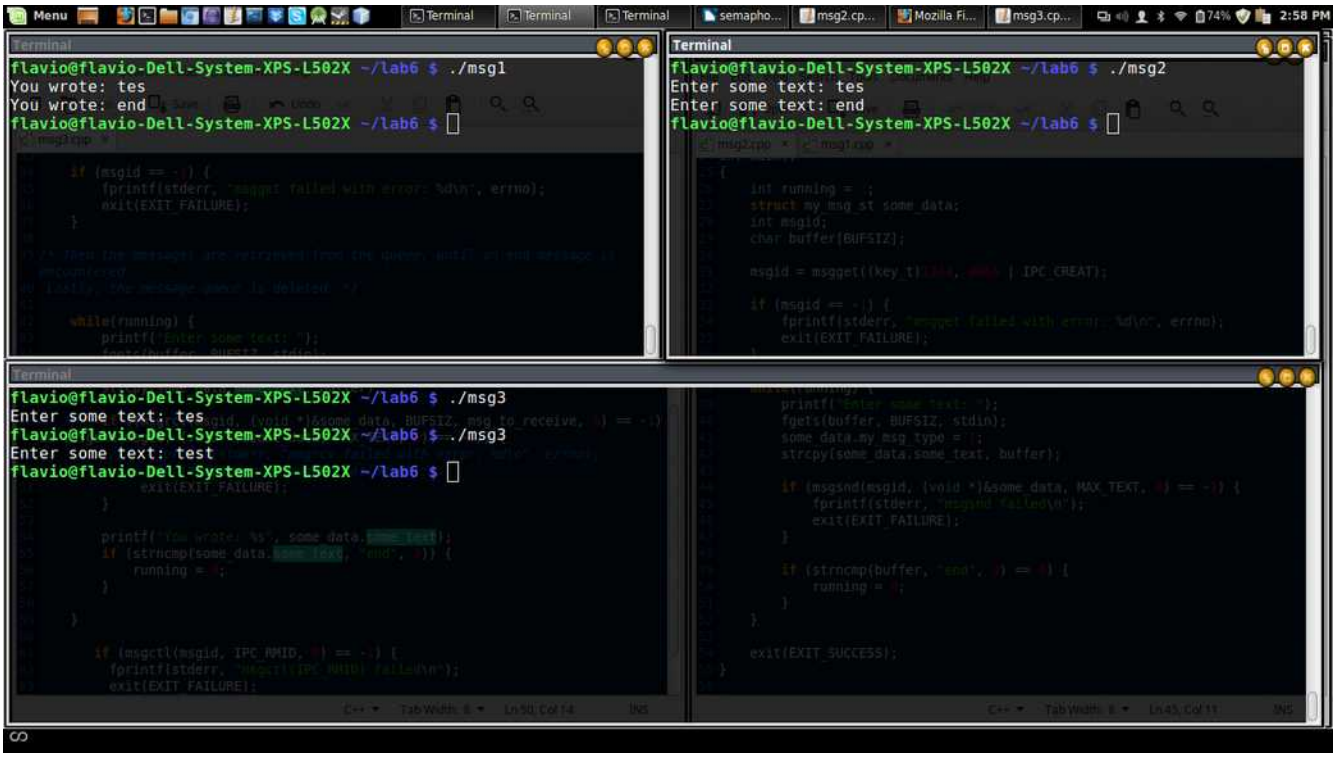
Dean Cosanella
Flavio dos Santos-Ross
CSE 460

All three programs compiled:



All three windows closed:

Dean Cosanella
Flavio dos Santos-Ross
CSE 460



2. IPC Status Commands

Use "man" to study the commands:

```
$ ipcs
$ ipcrm
```

ipcs:
provide information on ipc facilities
ipcs provides information on the ipc facilities for which the calling process has read access.

ipcrm:
remove a message queue, semaphore set or shared memory id
ipcrm removes System V interprocess communication (IPC) objects and associated data structures from the system. In order to delete such objects, you must be superuser, or the creator or owner of

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

the object.

Try and explain briefly what you see when executing each of the following commands. The italics are variables that you have to substitute with appropriate values. You may need to run your programs concerning semaphores, shared memory and message queues in order to see the effect.

```
$ ipcs -s
$ ipcrm sem semid
$ ipcs -m
$ ipcrm shm id
$ ipcs -q
$ ipcrm shm id
```

ipcs -s:
When this command is executed with the sema1.cpp program from last weeks lab, it prints the following:

------ Semaphore Arrays --------
key semid owner perms nsems
0x000004d2 98304 dean 666 1

From what we can tell, it seems like this command prints out the information of the semaphore that is being used in that particular program.

ipcrm sem *semid:*
When this command runs while sema1.cpp is running in the background, it prints the following:

resource(s) deleted

The *semid* is referring to the semid given from the previous command. In our case, the semid is 98304. This command appears to remove the entire semaphore associated with the semid given.

ipcs -m:

When this command is executed, we get the following:

------ Shared Memory Segments --------
key shmid owner perms bytes nattch status
0x00000000 65536 dean 600 33554432 2 dest
0x00000000 294913 dean 600 524288 2 dest
0x00000000 327682 dean 600 4194304 2 dest
0x00000000 360451 dean 777 4196352 2
0x00000000 393220 dean 600 393216 2 dest

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

```
0x00000000 491525     dean      600       393216     2          dest
0x00000000 1048582    dean      600       33554432   2          dest
0x00000000 4882439    dean      600       393216     2          dest
0x00000000 720904     dean      600       2097152    2          dest
0x00000000 4980745    dean      600       524288     2          dest
0x00000000 819210     dean      600       4194304    2          dest
0x00000000 5537803    dean      600       393216     2          dest
0x00000000 2261004    dean      600       4194304    2          dest
0x00000000 5636109    dean      600       2097152    2          dest
0x00000000 5996558    dean      600       4194304    2          dest
0x000004d2 5767183    dean      666       2052       1
0x000003e8 5799952    dean      666       27         0
0x00000000 2064401    dean      600       8388608    2          dest
0x00000000 6029330    dean      600       393216     2          dest
0x00000000 6127635    dean      600       393216     2          dest
0x00000000 6160404    dean      600       524288     2          dest
```

From this output, we believe that this command is showing the information of all of the shared memory segments which includes the id, permissions, size, attachments, and status.

ipcrm shm *id*:
When this command was executed, it printed out the following:

resource(s) deleted

This command is similar to the *ipcrm sem semid* command. This command removes the shared memory id.

ipcs -q:
When this command is executed while running the msg1.cpp and msg2.cpp, we get the following:

```
------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x000004d2 32768      dean       666        0            0
```

This command shows the information of the message queues that are currently running.

ipcrm msg *id*:
This command deletes the message queue associated with the msgid. In our case the msgid was 32768 and we had the following output:

resource(s) deleted

Dean Cosanella
Flavio dos Santos-Ross
CSE 460

For this lab, we learned a lot about inter process communication and we learned a lot about the commands used to view information about these IPCs. We give ourselves 20/20 on this lab.