Lab 5: Study of Interprocess Communications and XV6

By: Sarjil Hasan

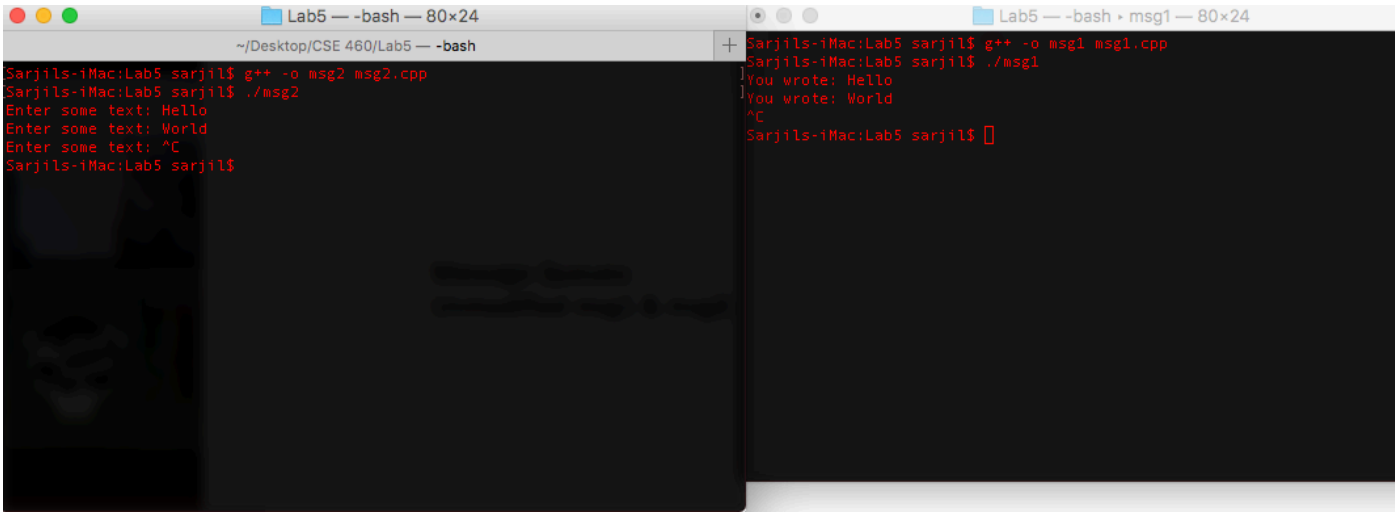
Lab Objectives:

- **Study interprocess communications**
- **Conduct XV6 debugging**

Message Queues:

msgctl: performs the control operation specified by cmd.
msgget: returns the message queue associated with the value of the key argument.
msgrcv: receives and reads a message from the associated queue identifier.
msgsnd: sends a message to queue.

Unmodified msg1 & msg2:



Modified msg1 & msg2:

```
//modified msg1.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];
};

int main()
{
    int running = 1;
```

```
    int msgid, msgid1;
    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);
    msgid1 = 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;
        }
    else{
         printf("Enter some text: ");
        fgets(buffer, BUFSIZ, stdin);
        some_data.my_msg_type = 1;
        strcpy(some_data.some_text, buffer);

        if (msgsnd(msgid1, (void *)&some_data, MAX_TEXT, 0) == -1) {

            exit(EXIT_FAILURE);
        }
        if (strncmp(buffer, "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);
}
```

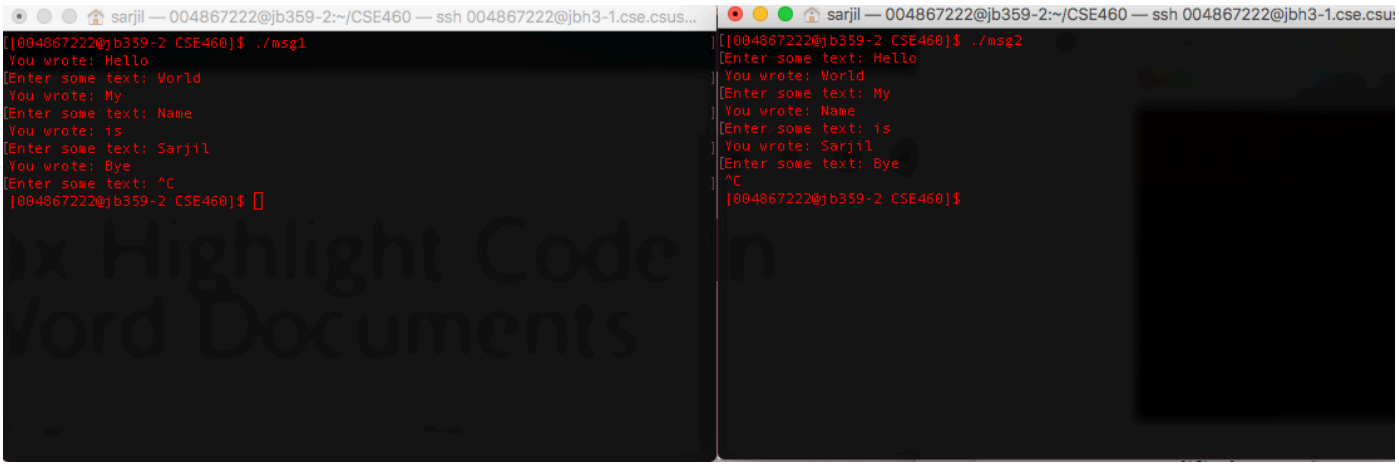
```
//modified msg2.cpp
/* The sender program is very similar to msg1.cpp. In the main set up, delete the
 msg_to_receive declaration and replace it with buffer[BUFSIZ], remove the message
 queue delete and make the following changes to the running loop.
 We now have a call to msgsnd to send the entered text to the queue. */

```

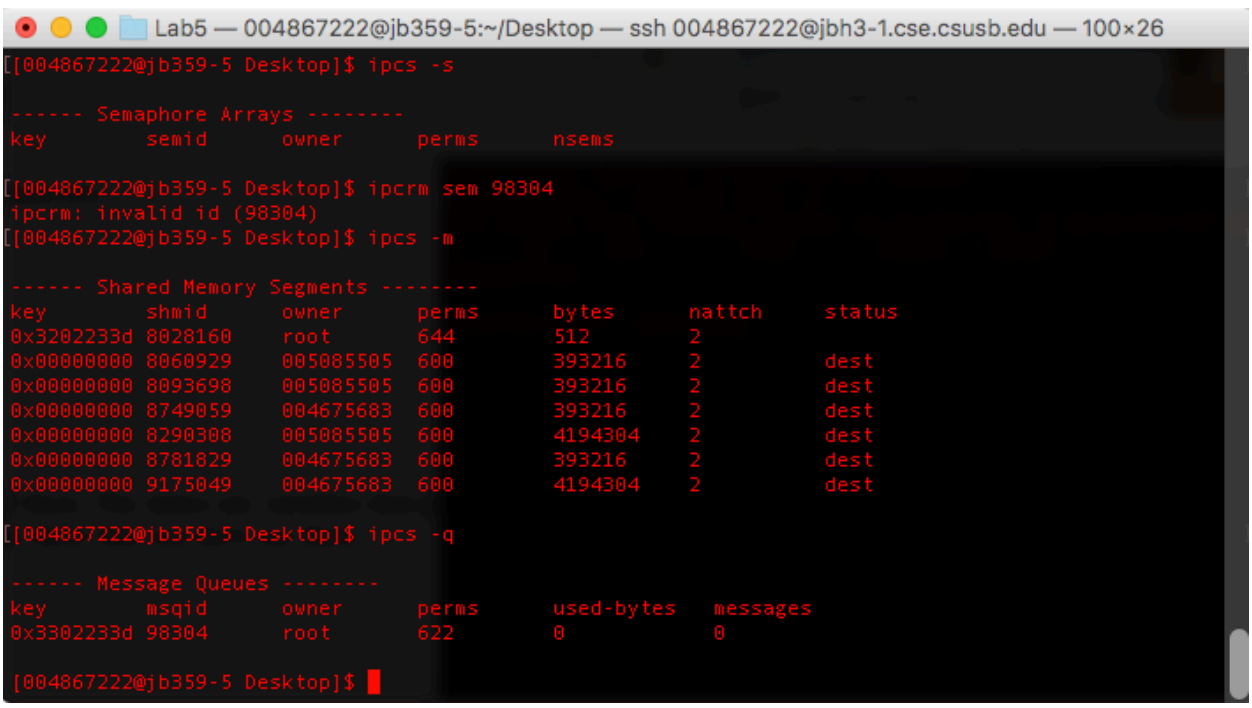
```
7.  #include <stdlib.h>
8.  #include <stdio.h>
9.  #include <string.h>
10. #include <errno.h>
11. #include <unistd.h>
12.
13. #include <sys/types.h>
14. #include <sys/ipc.h>
15. #include <sys/msg.h>
16.
17. #define MAX_TEXT 512
18.
19. struct my_msg_st {
20.     long int my_msg_type;
21.     char some_text[MAX_TEXT];
22. };
23.
24. int main()
25. {
26.     int running = 1;
27.     struct my_msg_st some_data;
28.     int msgid, msgid1;
29.     char buffer[BUFSIZ];
30.     long int msg_to_receive = 0;
31.
32.     msgid = msgget((key_t)1234, 0666 | IPC_CREAT);
33.   msgid1 = msgget((key_t)1234, 0666 | IPC_CREAT);
34.     if (msgid == -1) {
35.         fprintf(stderr, "msgget failed with error: %d\n", errno);
36.         exit(EXIT_FAILURE);
37.     }
38.
39.     while(running) {
40.         printf("Enter some text: ");
41.         fgets(buffer, BUFSIZ, stdin);
42.         some_data.my_msg_type = 1;
43.         strcpy(some_data.some_text, buffer);
44.
45.         if (msgsnd(msgid, (void *)&some_data, MAX_TEXT, 0) == -1) {
46.
47.             exit(EXIT_FAILURE);
48.         }else{
49.              if (msgrcv(msgid1, (void *)&some_data, BUFSIZ,
50.                    msg_to_receive, 0) == -1) {
51.             fprintf(stderr, "msgrcv failed with error: %d\n", errno);
52.             exit(EXIT_FAILURE);
53.         }
54.         printf("You wrote: %s", some_data.some_text);
55.         if (strncmp(some_data.some_text, "end", 3) == 0) {
56.             running = 0;
57.         }
58.         }
59.         if (strncmp(buffer, "end", 3) == 0) {
60.             running = 0;
61.         }
62.
63.     }
64.
65. if (msgctl(msgid, IPC_RMID, 0) == -1) {
66.         fprintf(stderr, "msgctl(IPC_RMID) failed\n");
67.         exit(EXIT_FAILURE);
```

```
    }

    exit(EXIT_SUCCESS);
}
```



<u>IPC Status:</u>



$ipcs –s: identifies which process is using semaphores.
$ipcrm sem *8028160*: although this did not work for me properly, it removes interprocess communication w/ the italicized ID (sem id).
$ipcs –m: identifies which segments of memory is shared.
$ipcs –q: identifies which IPC's semeaphores has messages in its queue.

Study of XV6:

```
[004867222@jb359-5 Desktop]$ gdb
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-94.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
warning: File "/students/csci/004867222/Desktop/.gdbinit" auto-loading has been
declined by your `auto-load safe-path' set to "$debugdir:$datadir/auto-
load:/usr/bin/mono-gdb.py:/usr/lib/golang/src/pkg/runtime/runtime-gdb.py".
To enable execution of this file add
       add-auto-load-safe-path /students/csci/004867222/Desktop/.gdbinit
line to your configuration file "/u/cse/004867222/.gdbinit".
To completely disable this security protection add
       set auto-load safe-path /
line to your configuration file "/u/cse/004867222/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
       info "(gdb)Auto-loading safe path"
(gdb) target remote :27800
Remote debugging using :27800
0x0000fff0 in ?? ()
(gdb) file kernel
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Reading symbols from /students/csci/004867222/Desktop/kernel...done.
(gdb) break swtch
Breakpoint 1 at 0x8010456c: file swtch.S, line 10.
(gdb) continue
Continuing.

Breakpoint 1, swtch () at swtch.S:10
10       movl 4(%esp), %eax
(gdb) step
11       movl 8(%esp), %edx
(gdb) step
14       pushl %ebp
(gdb) step
swtch () at swtch.S:15
15       pushl %ebx
(gdb) step
swtch () at swtch.S:16
16       pushl %esi
(gdb) step
swtch () at swtch.S:17
17       pushl %edi
(gdb) step
swtch () at swtch.S:20
20       movl %esp, (%eax)
(gdb) step
21       movl %edx, %esp
(gdb) step
swtch () at swtch.S:24
24       popl %edi
(gdb) step
swtch () at swtch.S:25
25       popl %esi
(gdb) step
```

```
swtch () at swtch.S:26
26      popl %ebx
(gdb) step
swtch () at swtch.S:27
27      popl %ebp
(gdb) step
swtch () at swtch.S:28
28      ret
(gdb) step
forkret () at proc.c:351
351   {
(gdb) step
forkret () at proc.c:354
354     release(&ptable.lock);
(gdb) step
release (lk=<error reading variable: can't compute CFA for this frame>,
    lk@entry=0x80112da0 <ptable>) at spinlock.c:48
48    {
(gdb) step
49      if(!holding(lk))
(gdb) continue
Continuing.

Breakpoint 1, swtch () at swtch.S:10
10      movl 4(%esp), %eax
(gdb) clear
Deleted breakpoint 1
(gdb) break exec
Breakpoint 2 at 0x801009b0: file exec.c, line 12.
(gdb) continue
Continuing.
[New Thread 2]
[Switching to Thread 2]

Breakpoint 2, exec (
    path=<error reading variable: can't compute CFA for this frame>,
    argv=<error reading variable: can't compute CFA for this frame>,
    argv@entry=0x8dfffeb0) at exec.c:12
12    {
(gdb) continue
Continuing.

Breakpoint 2, exec (
    path=<error reading variable: can't compute CFA for this frame>,
    argv=<error reading variable: can't compute CFA for this frame>,
    argv@entry=0x8dffeeb0) at exec.c:12
12    {
(gdb) continue
Continuing.

Breakpoint 2, exec (
    path=<error reading variable: can't compute CFA for this frame>,
    argv=<error reading variable: can't compute CFA for this frame>,
    argv@entry=0x8dfbeeb0) at exec.c:12
12    {
(gdb) print argv[0]
can't compute CFA for this frame
(gdb) print argv[1]
can't compute CFA for this frame
(gdb) print argv[2]
can't compute CFA for this frame
(gdb) backtrace
#0  exec (path=<error reading variable: can't compute CFA for this frame>,
    argv=<error reading variable: can't compute CFA for this frame>,
```

```
    argv@entry=0x8dfbeeb0) at exec.c:12
#1  0x801051e3 in sys_exec () at sysfile.c:418
#2  0x80104749 in syscall () at syscall.c:133
#3  0x801056d1 in trap (
    tf=<error reading variable: can't compute CFA for this frame>) at trap.c:43
#4  0x801054bd in alltraps () at trapasm.S:23
#5  0x8dfbefb4 in ?? ()
Backtrace stopped: previous frame inner to this frame (corrupt stack?)
(gdb) up
#1  0x801051e3 in sys_exec () at sysfile.c:418
418     return exec(path, argv);
(gdb)
```

What I observed was that we made a breakpoint at swtch and then continued the code by stepping over each line until we had finished/cleared the breakpoint. Then we breaked exec and went through different threads. Although I had some errors trying to obtain the variable names as I debugged it at first, I still understood the concept and got the same results as the professor's notes in the end.

```
[004867222@jb359-2 Desktop]$ gdb
GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-94.el7
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
warning: File "/students/csci/004867222/Desktop/.gdbinit" auto-loading has been
declined by your `auto-load safe-path' set to "$debugdir:$datadir/auto-
load:/usr/bin/mono-gdb.py:/usr/lib/golang/src/pkg/runtime/runtime-gdb.py".
To enable execution of this file add
       add-auto-load-safe-path /students/csci/004867222/Desktop/.gdbinit
line to your configuration file "/u/cse/004867222/.gdbinit".
To completely disable this security protection add
       set auto-load safe-path /
line to your configuration file "/u/cse/004867222/.gdbinit".
For more information about this security protection see the
"Auto-loading safe path" section in the GDB manual.  E.g., run from the shell:
       info "(gdb)Auto-loading safe path"
(gdb) target remote :27800
Remote debugging using :27800
0x0000fff0 in ?? ()
(gdb) file kernel
A program is being debugged already.
Are you sure you want to change the file? (y or n) y
Reading symbols from /students/csci/004867222/Desktop/kernel...done.
(gdb) break scheduler
Breakpoint 1 at 0x80103a20: file proc.c, line 281.
(gdb) continue
Continuing.
[New Thread 2]
[Switching to Thread 2]

Breakpoint 1, scheduler () at proc.c:281
281    {
(gdb) step
286        sti();
(gdb) step
```

```
sti () at x86.h:117
117       asm volatile("sti");
(gdb) step
scheduler () at proc.c:289
289         acquire(&ptable.lock);
(gdb) step
290         for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
(gdb) step
289         acquire(&ptable.lock);
(gdb) step
acquire (lk=<error reading variable: can't compute CFA for this frame>,
    lk@entry=0x80112da0 <ptable>) at spinlock.c:27
27        pushcli(); // disable interrupts to avoid deadlock.
(gdb) step
pushcli () at spinlock.c:27
27        pushcli(); // disable interrupts to avoid deadlock.
(gdb) step
readeflags () at x86.h:98
98        asm volatile("pushfl; popl %0" : "=r" (eflags));
(gdb) step
pushcli () at spinlock.c:106
106       cli();
(gdb) step
cli () at x86.h:111
111       asm volatile("cli");
(gdb) step
pushcli () at spinlock.c:107
107       if(cpu->ncli == 0)
(gdb) step
108         cpu->intena = eflags & FL_IF;
(gdb) step
109       cpu->ncli += 1;
(gdb) step
acquire (lk=<error reading variable: can't compute CFA for this frame>,
    lk@entry=0x80112da0 <ptable>) at spinlock.c:28
28        if(holding(lk))
(gdb) step
holding (lock=0x80112da0 <ptable>) at spinlock.c:92
92        return lock->locked && lock->cpu == cpu;
(gdb) step
acquire (lk=<error reading variable: can't compute CFA for this frame>,
    lk@entry=0x80112da0 <ptable>) at spinlock.c:32
32        while(xchg(&lk->locked, 1) != 0)
(gdb) continue
Continuing.
[Switching to Thread 1]

Breakpoint 1, scheduler () at proc.c:281
281     {
(gdb) clear
Deleted breakpoint 1
```

Evaluation:
I have successfully completed each part of the lab and understood each section. It was very interesting debugging in XV6 because it's very similar to Xcode's breakpoint system and I felt like I understood that the most.

Score: **20/20**