Steven Tang
CSE 460
2/7/14

Lab 5

1. Message queues

   Msgctl – What this message queue function does is it performs control operation specified by cmd on the system's message queue
   Msgget – This message queue function GETS a system message queue, returns identifier associated with the value of the key
   Msgrcv – This message queue function RECEIVES and READS a message from the queue associated with the queue identifer
   Msgsnd – This message queue function SENDS a message to a system message queue

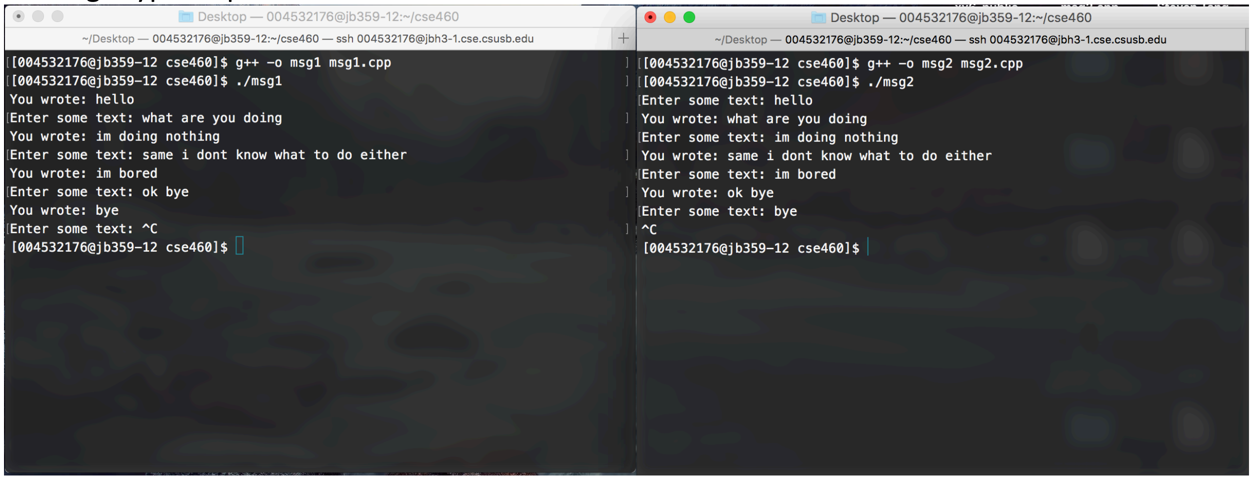
   Msg1.cpp

```
//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>


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;

/* 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

```
//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. */

#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[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);
}
```

message typescript


Desktop — 004532176@jb359-12:~/cse460
~/Desktop — 004532176@jb359-12:~/cse460 — ssh 004532176@jbh3-1.cse.csusb.edu
Desktop — 004532176@jb359-12:~/cse460
~/Desktop — 004532176@jb359-12:~/cse460 — ssh 004532176@jbh3-1.cse.csusb.edu

```
[004532176@jb359-12 cse460]$ g++ -o msg1 msg1.cpp
[004532176@jb359-12 cse460]$ ./msg1
You wrote: hello
Enter some text: what are you doing
You wrote: im doing nothing
Enter some text: same i dont know what to do either
You wrote: im bored
Enter some text: ok bye
You wrote: bye
Enter some text: ^C
[004532176@jb359-12 cse460]$
```

```
[004532176@jb359-12 cse460]$ g++ -o msg2 msg2.cpp
[004532176@jb359-12 cse460]$ ./msg2
Enter some text: hello
You wrote: what are you doing
Enter some text: im doing nothing
You wrote: same i dont know what to do either
Enter some text: im bored
You wrote: ok bye
Enter some text: bye
^C
[004532176@jb359-12 cse460]$
```

2. IPCS Status Commands

<u>IPCS –s</u>

```
Script started on Tue Feb  7 17:48:35 2017
[?1034hbash-3.2$ ipcs -s
IPC status from <running system> as of Tue Feb  7 17:48:39 PST 2017
T     ID     KEY        MODE       OWNER    GROUP
Semaphores:

bash-3.2$ exit
exit

Script done on Tue Feb  7 17:48:44 2017
```

IPCS –s displays information about any ACTIVE semaphores on the system

<u>Ipcrm sem</u> – It asked me to enter a specific id to remove the semaphore, but because there was none running I could not do so.

IPCRM SEM removes the semaphore set from the specific ID that the user has entered.

Ipcs –m

```
Script started on Tue Feb  7 17:49:46 2017
[?1034hbash-3.2$ ipcs m[K-m
IPC status from <running system> as of Tue Feb  7 17:49:52 PST 2017
T     ID     KEY        MODE       OWNER    GROUP
Shared Memory:

bash-3.2$ exit
exit

Script done on Tue Feb  7 17:49:57 2017
```

IPCS –M displays information about the active shared memory on the system

Ipcs –q

```
Script started on Tue Feb  7 17:50:14 2017
[?1034hbash-3.2$ ipcs -[K-q
IPC status from <running system> as of Tue Feb  7 17:50:16 PST 2017
T     ID     KEY        MODE       OWNER    GROUP
Message Queues:
q  65536 0x000004d2 --rw-rw-rw- steventang    staff

bash-3.2$ ei[Kxit
exit

Script done on Tue Feb  7 17:50:21 2017
```

IPCS –Q Displays information about the message queues

3. Study of XV6

Debugging Process:

```
[004532176@jb359-16 xv6]$ 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/004532176/Desktop/xv6/.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/004532176/Desktop/xv6/.gdbinit
line to your configuration file "/u/cse/004532176/.gdbinit".
To completely disable this security protection add
    set auto-load safe-path /
line to your configuration file "/u/cse/004532176/.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 :27907
Remote debugging using :27907
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/004532176/Desktop/xv6/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) break
Note: breakpoint 2 also set at pc 0x801009b0.
Breakpoint 3 at 0x801009b0: file exec.c, line 12.
(gdb) clear
Deleted breakpoints 2 3
(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=0x8dffeeb0) 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  0x8dffefb4 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) list
413       break;
414     }
```

```
      415     if(fetchstr(uarg, &argv[i]) < 0)
      416       return -1;
      417   }
      418   return exec(path, argv);
      419 }
      420
      421 int
      422 sys_pipe(void)
      (gdb)
```

Scheduler function

```
[004532176@jb359-16 xv6]$ 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/004532176/Desktop/xv6/.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/004532176/Desktop/xv6/.gdbinit
line to your configuration file "/u/cse/004532176/.gdbinit".
To completely disable this security protection add
    set auto-load safe-path /
line to your configuration file "/u/cse/004532176/.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:27907
Remote debugging using :27907
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/004532176/Desktop/xv6/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) list
276 //   - swtch to start running that process
277 //   - eventually that process transfers control
278 //       via swtch back to the scheduler.
279 void
280 scheduler(void)
281 {
282   struct proc *p;
283
284  for(;;){
285   // Enable interrupts on this processor.
(gdb) list
286   sti();
287
288   // Loop over process table looking for process to run.
289   acquire(&ptable.lock);
290   for(p = ptable.proc; p < &ptable.proc[NPROC]; p++){
291    if(p->state != RUNNABLE)
292     continue;
293
294    // Switch to chosen process.  It is the process's job
295    // to release ptable.lock and then reacquire it
(gdb) list
296    // before jumping back to us.
297    proc = p;
298    switchuvm(p);
299    p->state = RUNNING;
300    swtch(&cpu->scheduler, p->context);
301    switchkvm();
302
303    // Process is done running for now.
304    // It should have changed its p->state before coming back.
305    proc = 0;
(gdb) list
306   }
307   release(&ptable.lock);
308
309  }
310 }
311
312 // Enter scheduler.  Must hold only ptable.lock
313 // and have changed proc->state. Saves and restores
314 // intena because intena is a property of this
315 // kernel thread, not this CPU. It should
(gdb) list
316 // be proc->intena and proc->ncli, but that would
317 // break in the few places where a lock is held but
318 // there's no process.
319 void
320 sched(void)
321 {
322  int intena;
323
324  if(!holding(&ptable.lock))
325   panic("sched ptable.lock");
(gdb) list
326  if(cpu->ncli != 1)
327   panic("sched locks");
328  if(proc->state == RUNNING)
329   panic("sched running");
330  if(readeflags()&FL_IF)
331   panic("sched interruptible");
332  intena = cpu->intena;
333  swtch(&proc->context, cpu->scheduler);
334  cpu->intena = intena;
335 }
(gdb) list
336
337 // Give up the CPU for one scheduling round.
```

```
338 void
339 yield(void)
340 {
341   acquire(&ptable.lock);  //DOC: yieldlock
342   proc->state = RUNNABLE;
343   sched();
344   release(&ptable.lock);
345 }
(gdb) list
346
347 // A fork child's very first scheduling by scheduler()
348 // will swtch here.  "Return" to user space.
349 void
350 forkret(void)
351 {
352   static int first = 1;
353   // Still holding ptable.lock from scheduler.
354   release(&ptable.lock);
355
(gdb) q
```

Evaluation: I did all of the parts that was asked for in the lab. In the first part I explored sending messages through different terminals and then edited my programs so that they would switch back and forth every time they send a message. I have found out that working on OSX will not make the program switch terminals, only on a linux based system. Next I learned how to use the IPCS (Inter-process communication status) and its commands. I also got to XV6 to work and debugged it just like how I was asked to. I followed the steps and did my own debugging of scheduler. At the end I would give myself a 20/20

Score: 20/20