George Suarez

David Cruz

CSE 460

Lab 7 – Semaphore II and XV6 System Calls

1. Shared Memory

Type in some text at the terminals. What do you see? What text you enter will terminate the programs? Explain what you have seen.

Both terminals print out “Memory attached at…”, however, the shared2 also gets user input. When you type text in the shared2 terminal, it appears in the shared1 terminal because the two programs share a memory space. To terminate both programs, you type “end” into the shared2 terminal.

shared1_mod.cpp:

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

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <iostream>

#define TEXT_SZ 2048
#define SNAME "mysem"
using namespace std;

struct shared_use_st
{
  int written_by_you;
  char some_text[TEXT_SZ];
};

//Checks if semaphore was created successfully
bool semaphore_error(sem_t *sem)
{
  if (sem == SEM_FAILED)
  {
    return true;
  }
  return false;
}

int main()
{
  int running = 1;
  void *shared_memory = (void *)0;
  struct shared_use_st *shared_stuff;
  int shmid;

  char buffer[BUFSIZ];

//Creates semaphore
  sem_t *sem = sem_open(SNAME, O_CREAT, 06344, 1);
//If semaphore not created out put ”Semaphore connection Failure!”
  if (semaphore_error(sem))
  {
    cout << "Semaphore connection failure 1!\n";
     int sem_unlink(const char* mutex); //close semaphore
    exit(-1);
  }

  srand((unsigned int)getpid());

  shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666 | IPC_CREAT);
```

```
if (shmid == -1)
{
  fprintf(stderr, "shmget failed\n");
  exit(EXIT_FAILURE);
}

/* We now make the shared memory accessible to the program. */

shared_memory = shmat(shmid, (void *)0, 0);
if (shared_memory == (void *)-1)
{
  fprintf(stderr, "shmat failed\n");
  exit(EXIT_FAILURE);
}

printf("Memory attached at %X\n", shared_memory);

/* The next portion of the program assigns the shared_memory segment to shared_stuff,
which then prints out any text in written_by_you. The loop continues until end is found
in written_by_you. The call to sleep forces the consumer to sit in its critical section,
which makes the producer wait. */

shared_stuff = (struct shared_use_st *)shared_memory;
shared_stuff->written_by_you = 0;
while (running)
{
  if (shared_stuff->written_by_you)
  {

    sem_wait(sem);
    printf("You wrote: %s", shared_stuff->some_text);
    sleep(rand() % 4); /* make the other process wait for us ! */
    shared_stuff->written_by_you = 0;
    sem_post(sem);
    if (strncmp(shared_stuff->some_text, "end", 3) == 0)
    {
      running = 0;
      }
    }
  }

  /* Lastly, the shared memory is detached and then deleted. */

  if (shmdt(shared_memory) == -1)
  {
    fprintf(stderr, "shmdt failed\n");
    exit(EXIT_FAILURE);
  }

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

  sem_close(sem);
  sem_unlink(SNAME);

  exit(EXIT_SUCCESS);
}
```

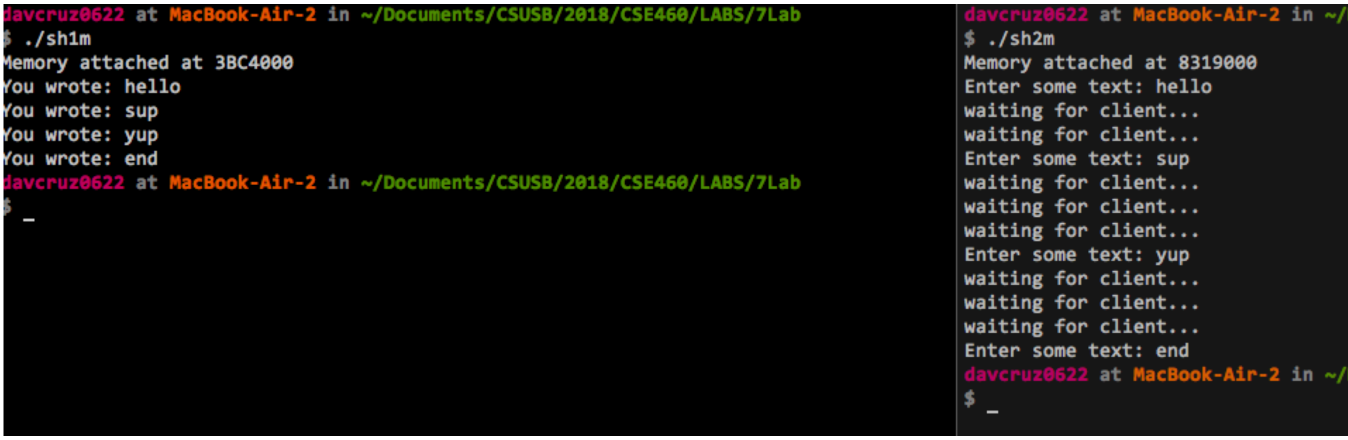
shared2_mod.cpp:

```
/*
  shared2_mod.cpp: Similar to shared1.cpp except that it writes data to
  the shared memory.
*/
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <semaphore.h>
#include <iostream>
#include <sys/types.h>
```

```
#include <sys/ipc.h>
#include <sys/shm.h>

#define SNAME "mysem"
#define TEXT_SZ 2048

using namespace std;

struct shared_use_st
{
  int written_by_you;
  char some_text[TEXT_SZ];
};

bool semaphore_error( sem_t *mutex)
{
  if(mutex == SEM_FAILED)
    return true;
  else  return false;
}

int main()
{
  int running = 1;
  void *shared_memory = (void *)0;
  struct shared_use_st *shared_stuff;
  char buffer[BUFSIZ];
  int shmid;

  sem_t *sem = sem_open(SNAME,O_CREAT, 0644, 1);
  if(semaphore_error(sem))
  {
    cout<<"Semaphore connection failure!"<<endl;
    sem_close(sem);
    exit(-1);
  }
```

```
shmid = shmget((key_t)1234, sizeof(struct shared_use_st), 0666 | IPC_CREAT);

if (shmid == -1)
{
  fprintf(stderr, "shmget failed\n");
  exit(EXIT_FAILURE);
}

shared_memory = shmat(shmid, (void *)0, 0);
if (shared_memory == (void *)-1)
{
  fprintf(stderr, "shmat failed\n");
  exit(EXIT_FAILURE);
}

printf("Memory attached at %X\n", shared_memory);

shared_stuff = (struct shared_use_st *)shared_memory;
while (running)
{
  while (shared_stuff->written_by_you == 1)
  {
    sleep(1);
    printf("waiting for client...\n");
  }
  sem_wait(sem);
  printf("Enter some text: ");
  fgets(buffer, BUFSIZ, stdin);

  strncpy(shared_stuff->some_text, buffer, TEXT_SZ);
  shared_stuff->written_by_you = 1;
  sem_post(sem);

  if (strncmp(buffer, "end", 3) == 0)
  {
    running = 0;
  }
  }

  if (shmdt(shared_memory) == -1)
  {
    fprintf(stderr, "shmdt failed\n");
    exit(EXIT_FAILURE);
  }

  sem_close(sem);
  sem_unlink(SNAME);
  exit(EXIT_SUCCESS);
}
```

Output:

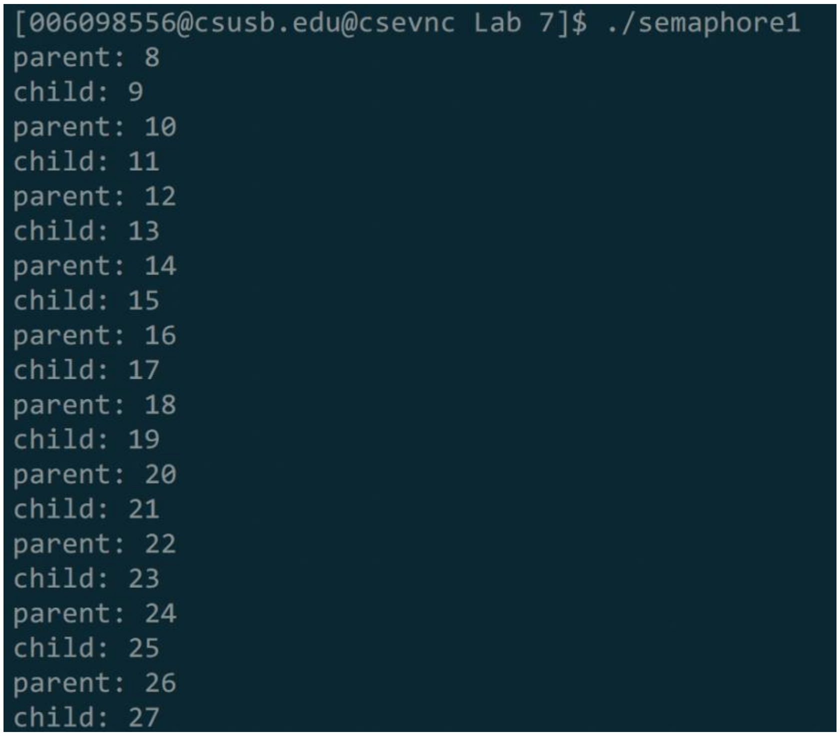


```
davcruz0622 at MacBook-Air-2 in ~/Documents/CSUSB/2018/CSE460/LABS/7Lab
$ ./sh1m
Memory attached at 3BC4000
You wrote: hello
You wrote: sup
You wrote: yup
You wrote: end
davcruz0622 at MacBook-Air-2 in ~/Documents/CSUSB/2018/CSE460/LABS/7Lab
$ _
```

```
davcruz0622 at MacBook-Air-2 in ~/
$ ./sh2m
Memory attached at 8319000
Enter some text: hello
waiting for client...
waiting for client...
Enter some text: sup
waiting for client...
waiting for client...
waiting for client...
Enter some text: yup
waiting for client...
waiting for client...
waiting for client...
Enter some text: end
davcruz0622 at MacBook-Air-2 in ~/
$ _
```

2. POSIX Semaphores:

Output:

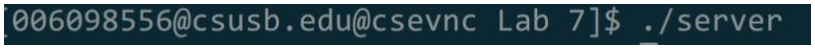


```
[006098556@csusb.edu@csevnc Lab 7]$ ./semaphore1
parent: 8
child: 9
parent: 10
child: 11
parent: 12
child: 13
parent: 14
child: 15
parent: 16
child: 17
parent: 18
child: 19
parent: 20
child: 21
parent: 22
child: 23
parent: 24
child: 25
parent: 26
child: 27
```

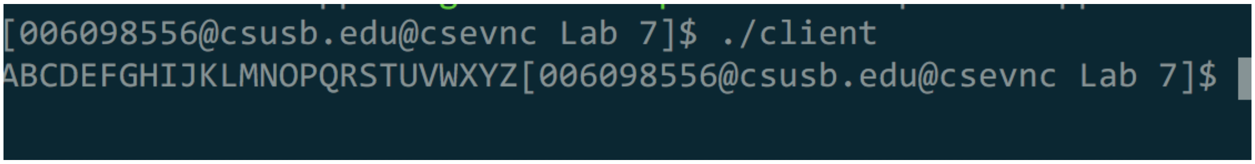
Explanation: A semaphore is created to control if a parent or child gets access to the critical section and there is a counter that starts at 8. Every time a process has been created, the counter gets increased by 1, and the parent process gets access first then followed by the child process. There is a total of 10 parent processes and 10 child processes that were created in this program.

Try the **server-client** example above and explain what you observe. You have to start the server first (why?).

Output:



```
[006098556@csusb.edu@csevnc Lab 7]$ ./server
```



```
[006098556@csusb.edu@csevnc Lab 7]$ ./client
ABCDEFGHIJKLMNOPQRSTUVWXYZ[006098556@csusb.edu@csevnc Lab 7]$
```

Explanation: The server is executed first because it is the one that creates the semaphore, and waits for another process to get access such as a client which then outputs ‘A-Z’ when *client.cpp* is executed.

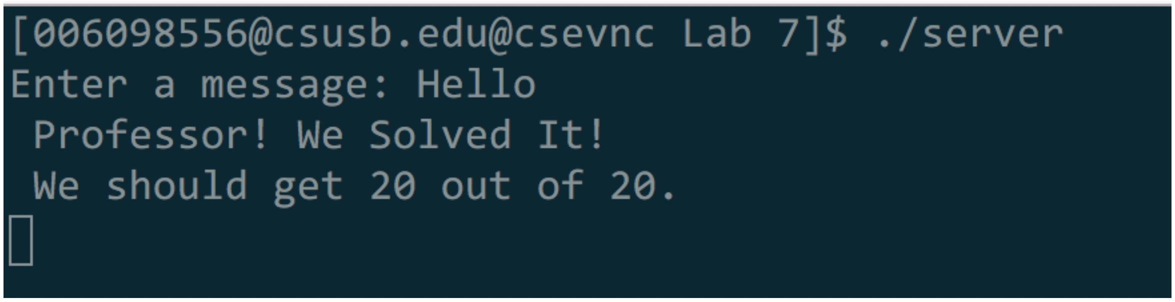
Modify the programs so that the server sits in a loop to accept string inputs from users and send them to the client, which then prints out the string.

server.cpp:

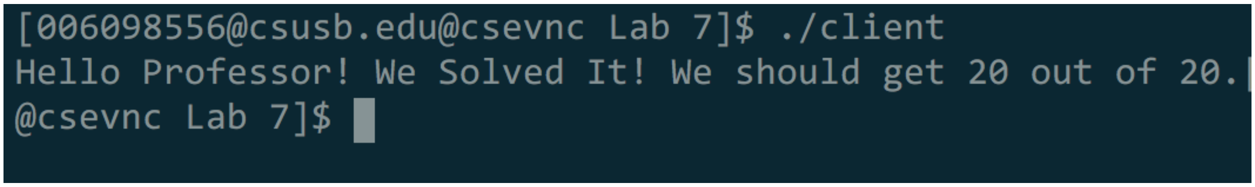
```
// server.cpp
// g++ -o server server.cpp -lpthread -lrt
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <semaphore.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <iostream>

using namespace std;

#define SHMSZ 27
char SEM_NAME[] = "vik";

int main()
{
   char ch;
   int shmid;
   key_t key;
   char *shm, *s;
   sem_t *mutex;

   //name the shared memory segment
   key = 1000;

   //create & initialize semaphore
```

```
 mutex = sem_open(SEM_NAME, O_CREAT, 0644, 1);
 if (mutex == SEM_FAILED)
 {
   perror("unable to create semaphore");
   sem_unlink(SEM_NAME);
   exit(-1);
 }

 //create the shared memory segment with this key
 shmid = shmget(key, SHMSZ, IPC_CREAT | 0666);
 if (shmid < 0)
 {
     perror("failure in shmget");
  exit(-1);
 }

 //attach this segment to virtual memory
 shm = (char*)shmat(shmid, NULL, 0);

 //start writing into memory
 s = shm;

 cout << "Enter a message: ";
 string message = "";
 while (getline(cin, message))
 {
   for (int i = 0; i < message.length(); i++)
   {
     sem_wait(mutex);
     *s++ = message[i];
     sem_post(mutex);
   }
 }

//the below loop could be replaced by binary semaphore
 while (*shm != '*')
 {
        sleep(1);
    }
    sem_close(mutex);
    sem_unlink(SEM_NAME);
    shmctl(shmid, IPC_RMID, 0);
    _exit(0);
}
```

Output:



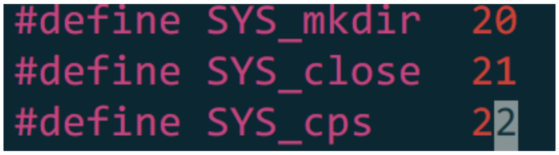
```
[006098556@csusb.edu@csevnc Lab 7]$ ./server
Enter a message: Hello
 Professor! We Solved It!
 We should get 20 out of 20.
```



```
[006098556@csusb.edu@csevnc Lab 7]$ ./client
Hello Professor! We Solved It! We should get 20 out of 20.
@csevnc Lab 7]$
```

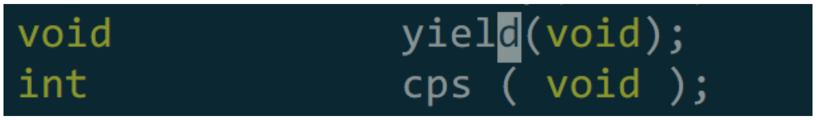
3. **XV6 – System Calls**

   a. Adding the c*ps* name to *syscall.h*



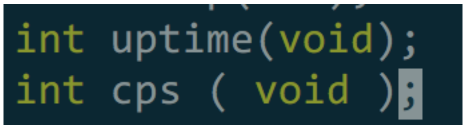
```
#define SYS_mkdir  20
#define SYS_close  21
#define SYS_cps    22
```

   b. Adding the function prototype to *defs.h*



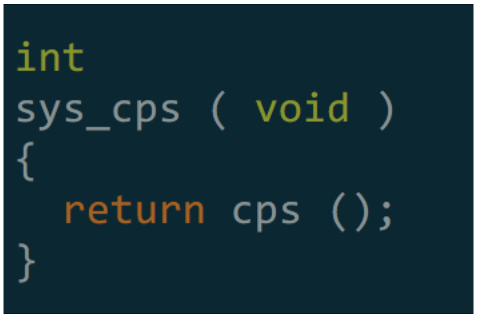
```
void                yield(void);
int                 cps ( void );
```

c. Adding function prototype to *user.h:*



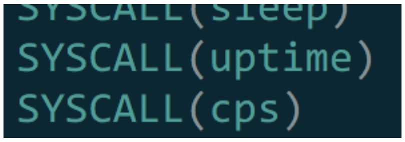
```
int uptime(void);
int cps ( void );
```

d. Adding function call to *sysproc.c:*



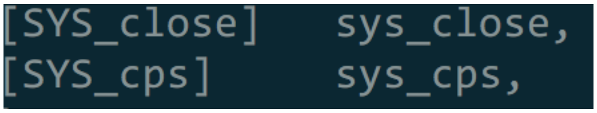
```
int
sys_cps ( void )
{
  return cps ();
}
```

e. Adding call to *usys.S:*



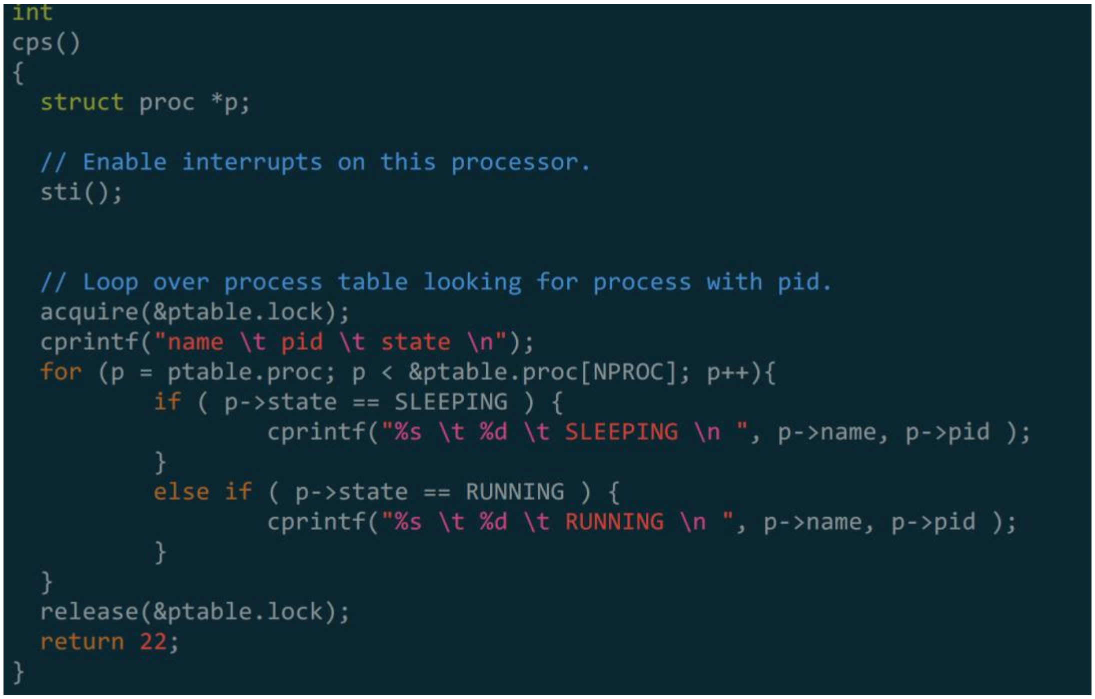
```
SYSCALL(sleep)
SYSCALL(uptime)
SYSCALL(cps)
```

f. Add call to *syscall.c:*



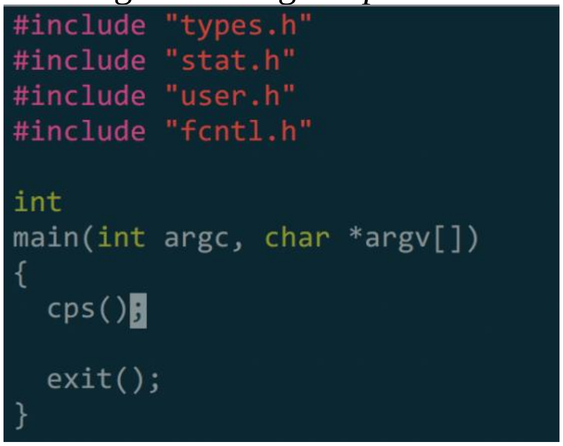
```
[SYS_close]   sys_close,
[SYS_cps]     sys_cps,
```

g. Add the code to *proc.c:*



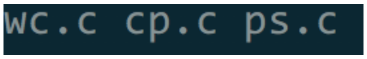
```
int
cps()
{
  struct proc *p;

  // Enable interrupts on this processor.
  sti();


  // Loop over process table looking for process with pid.
  acquire(&ptable.lock);
  cprintf("name \t pid \t state \n");
  for (p = ptable.proc; p < &ptable.proc[NPROC]; p++){
          if ( p->state == SLEEPING ) {
                  cprintf("%s \t %d \t SLEEPING \n ", p->name, p->pid );
          }
          else if ( p->state == RUNNING ) {
                  cprintf("%s \t %d \t RUNNING \n ", p->name, p->pid );
          }
  }
  release(&ptable.lock);
  return 22;
}
```

h. Creating the testing file *ps.c:*

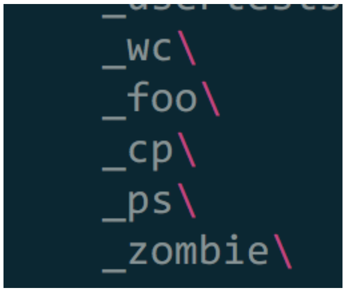


```
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"

int
main(int argc, char *argv[])
{
  cps();

  exit();
}
```

i. Modifying *Makefile:*

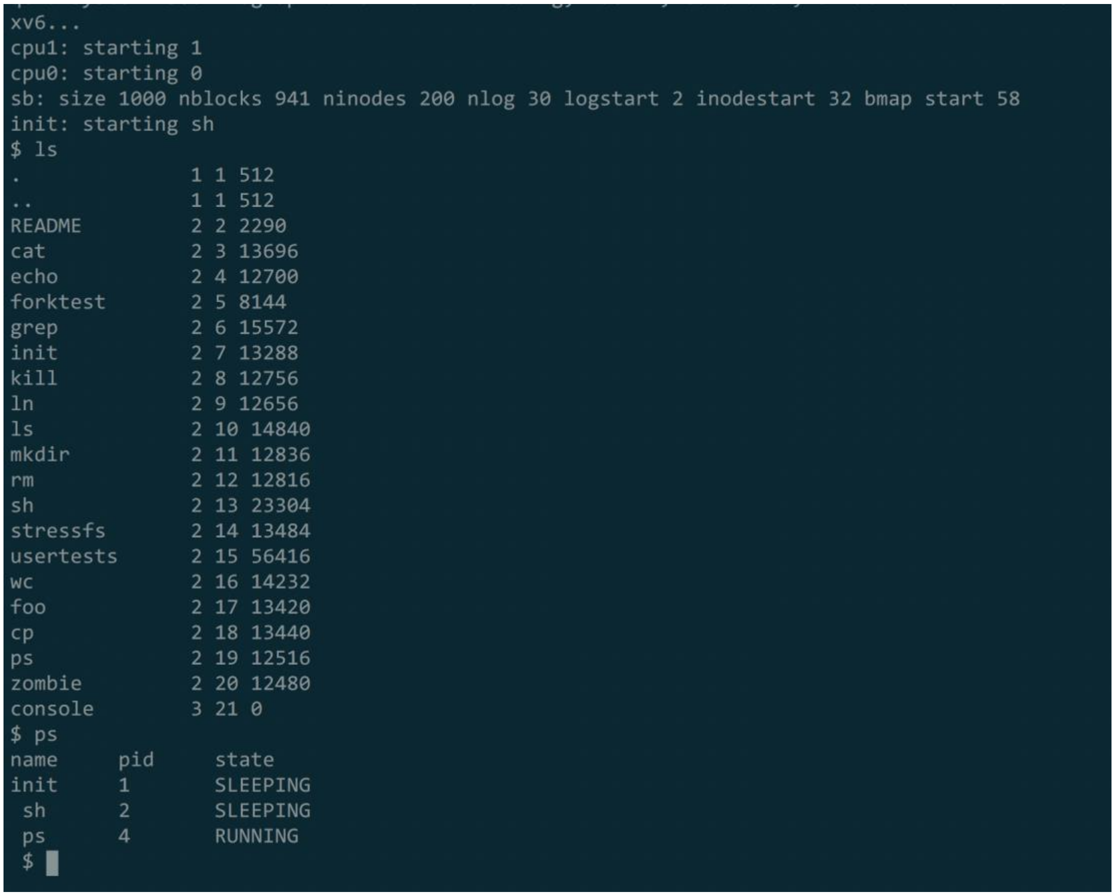


```
wc.c cp.c ps.c
```



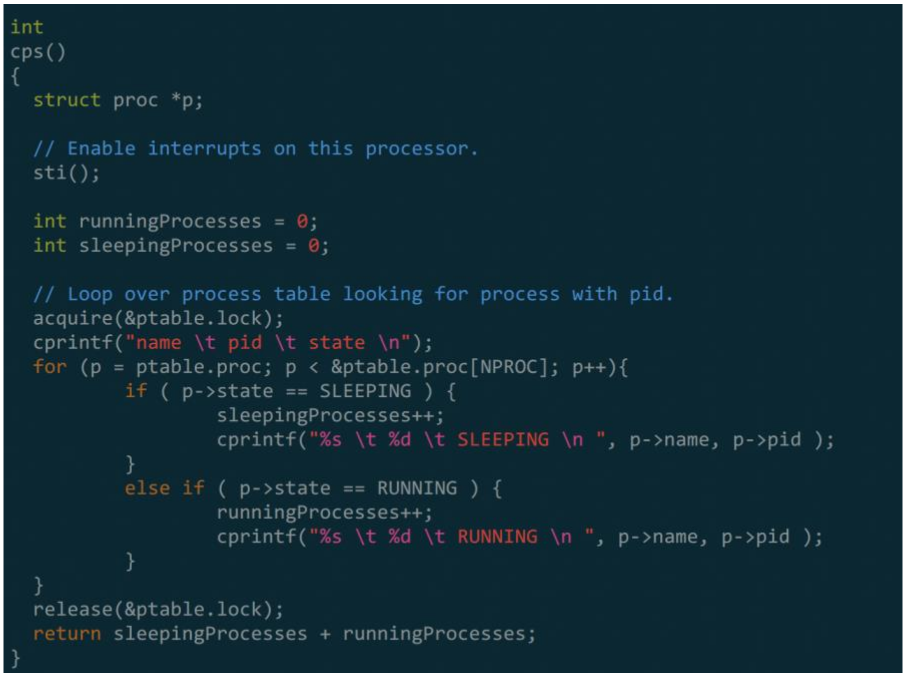
```
	_wc\
	_foo\
	_cp\
	_ps\
	_zombie\
```

j. After running *$make qemu-nox*:



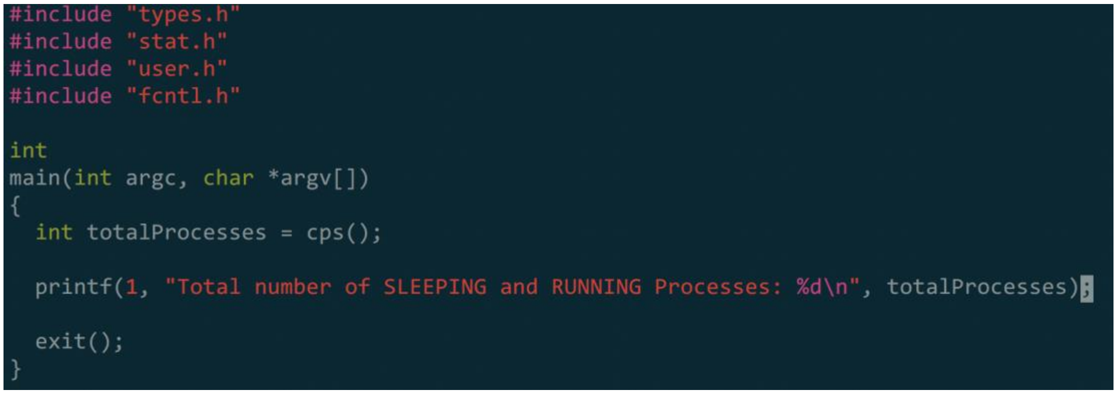
```
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ ls
.              1 1 512
..             1 1 512
README         2 2 2290
cat            2 3 13696
echo           2 4 12700
forktest       2 5 8144
grep           2 6 15572
init           2 7 13288
kill           2 8 12756
ln             2 9 12656
ls             2 10 14840
mkdir          2 11 12836
rm             2 12 12816
sh             2 13 23304
stressfs       2 14 13484
usertests      2 15 56416
wc             2 16 14232
foo            2 17 13420
cp             2 18 13440
ps             2 19 12516
zombie         2 20 12480
console        3 21 0
$ ps
name     pid     state
init     1       SLEEPING
 sh      2       SLEEPING
 ps      4       RUNNING
 $
```

k. Modified *proc.c* code:



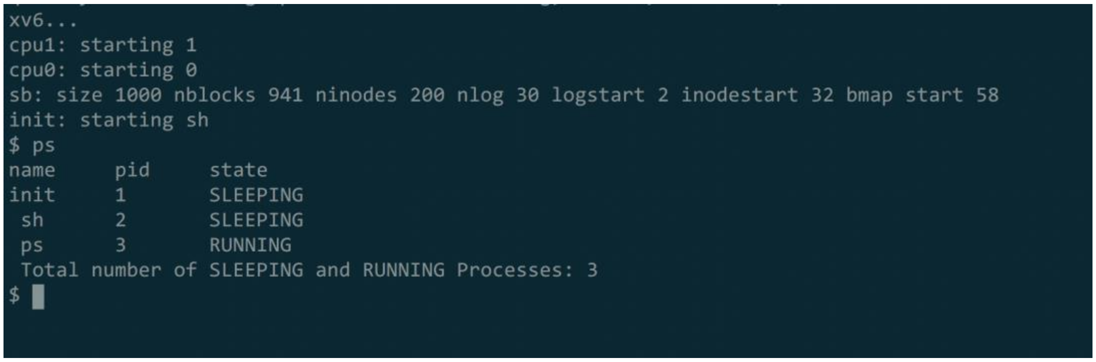
```c
int
cps()
{
  struct proc *p;

  // Enable interrupts on this processor.
  sti();

  int runningProcesses = 0;
  int sleepingProcesses = 0;

  // Loop over process table looking for process with pid.
  acquire(&ptable.lock);
  cprintf("name \t pid \t state \n");
  for (p = ptable.proc; p < &ptable.proc[NPROC]; p++){
          if ( p->state == SLEEPING ) {
                  sleepingProcesses++;
                  cprintf("%s \t %d \t SLEEPING \n ", p->name, p->pid );
          }
          else if ( p->state == RUNNING ) {
                  runningProcesses++;
                  cprintf("%s \t %d \t RUNNING \n ", p->name, p->pid );
          }
  }
  release(&ptable.lock);
  return sleepingProcesses + runningProcesses;
}
```

l. Modified the testing file *ps.c*:



```c
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"

int
main(int argc, char *argv[])
{
  int totalProcesses = cps();

  printf(1, "Total number of SLEEPING and RUNNING Processes: %d\n", totalProcesses);

  exit();
}
```

m. Output:



```
xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ ps
name     pid     state
init     1       SLEEPING
 sh      2       SLEEPING
 ps      3       RUNNING
 Total number of SLEEPING and RUNNING Processes: 3
$ 
```

Discussion: We have completed all sections in this lab. We should get **20/20.**