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);

}

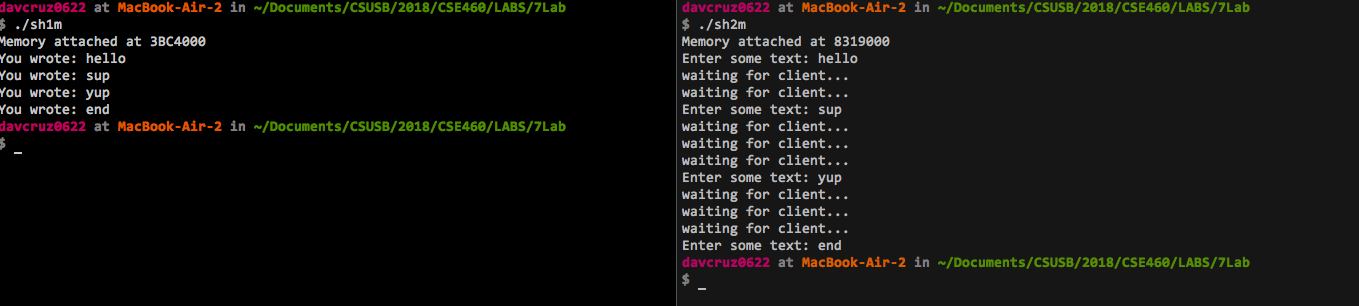
sem\_close(sem);

sem\_unlink(SNAME);

exit(EXIT\_SUCCESS);

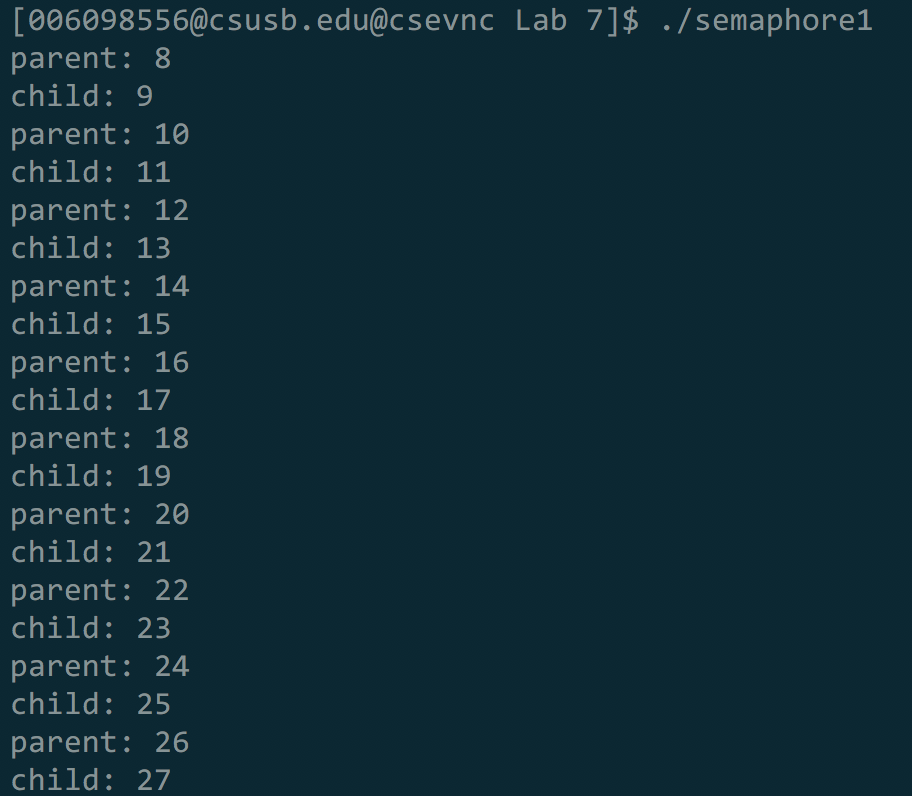
}

Output:



1. **POSIX Semaphores:**

Output:

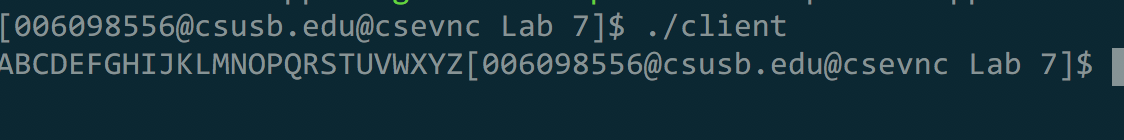
**

**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:





**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);

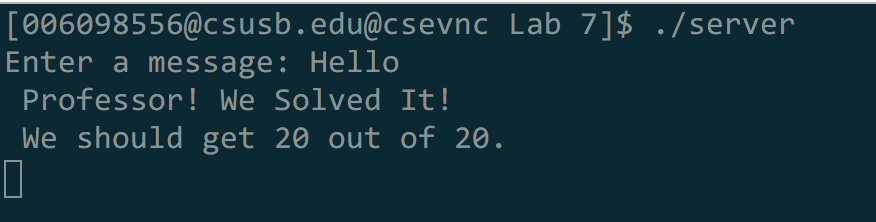
sem\_unlink(SEM\_NAME);

shmctl(shmid, IPC\_RMID, 0);

\_exit(0);

}

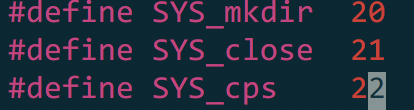
Output:



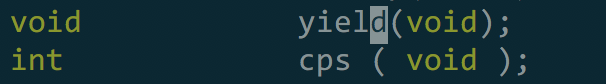


3. **XV6 – System Calls**

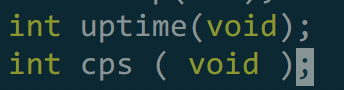
1. Adding the c*ps* name to *syscall.h*



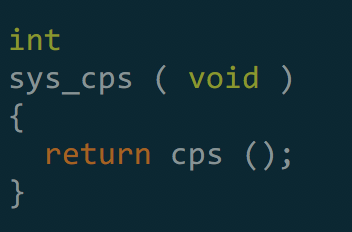
1. Adding the function prototype to *defs.h*



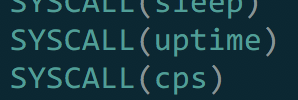
1. Adding function prototype to *user.h:*

**

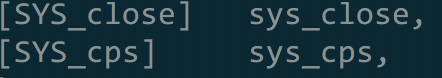
1. Adding function call to *sysproc.c:*



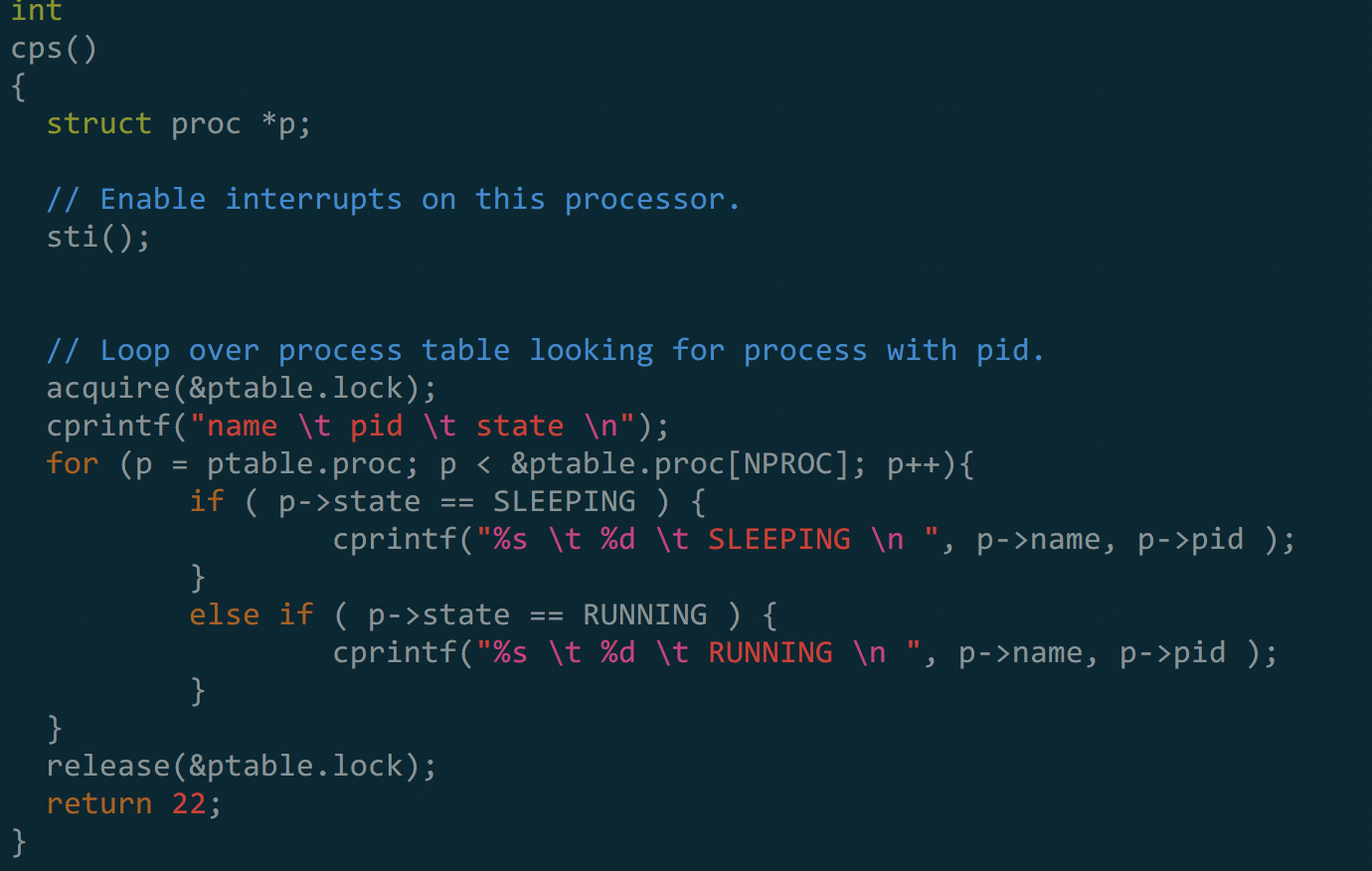
1. Adding call to *usys.S:*



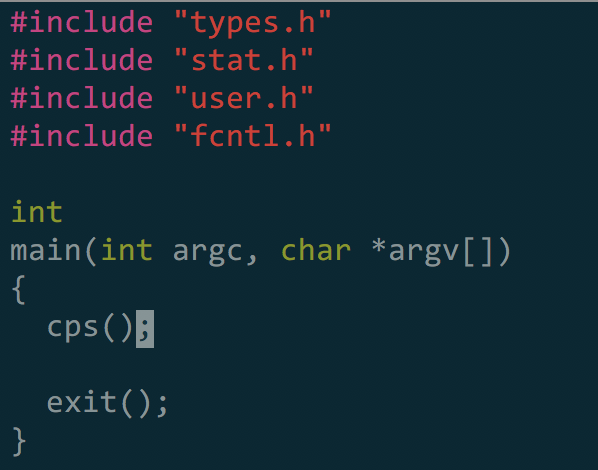
1. Add call to *syscall.c:*



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

**

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

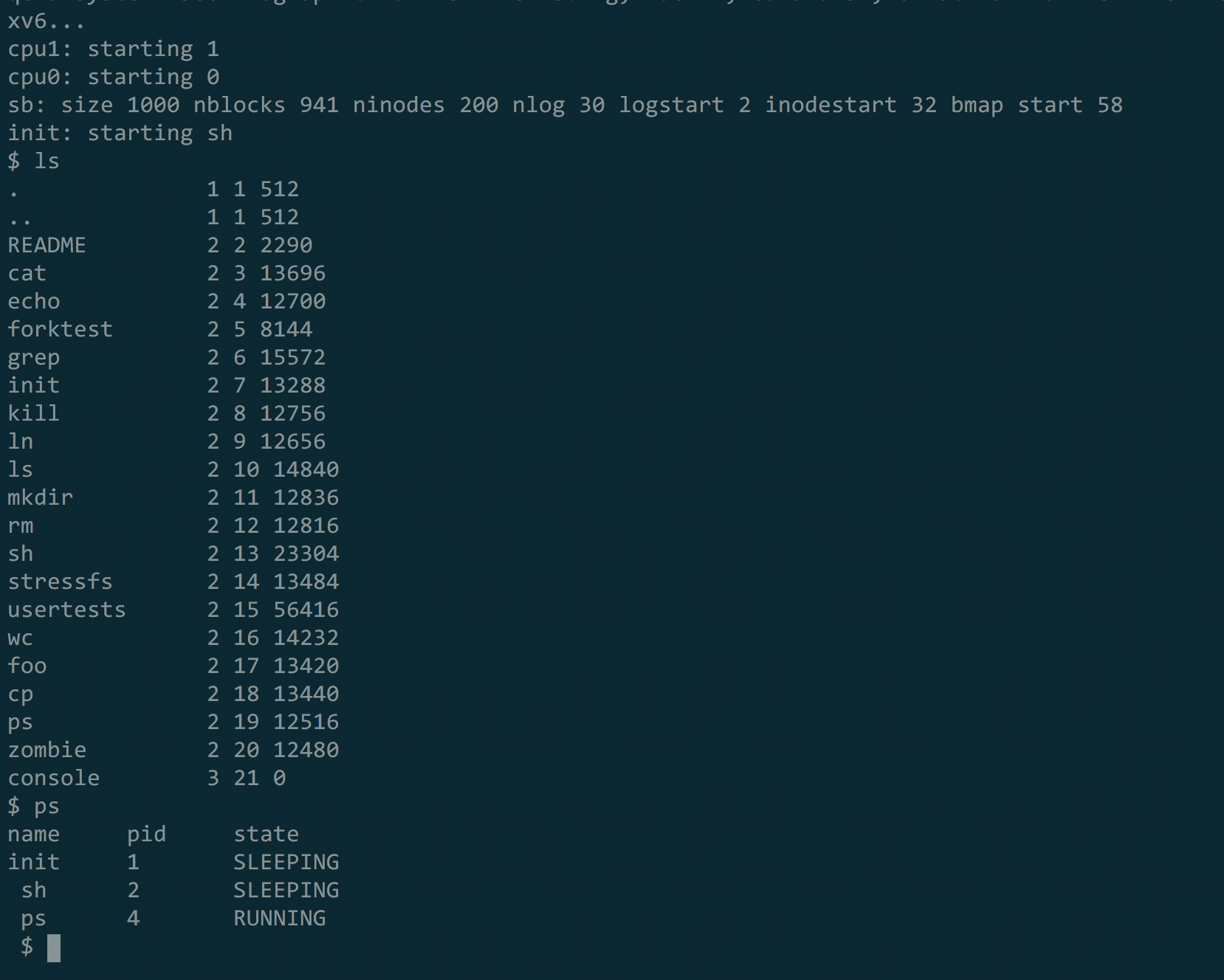


1. Modifying *Makefile:*

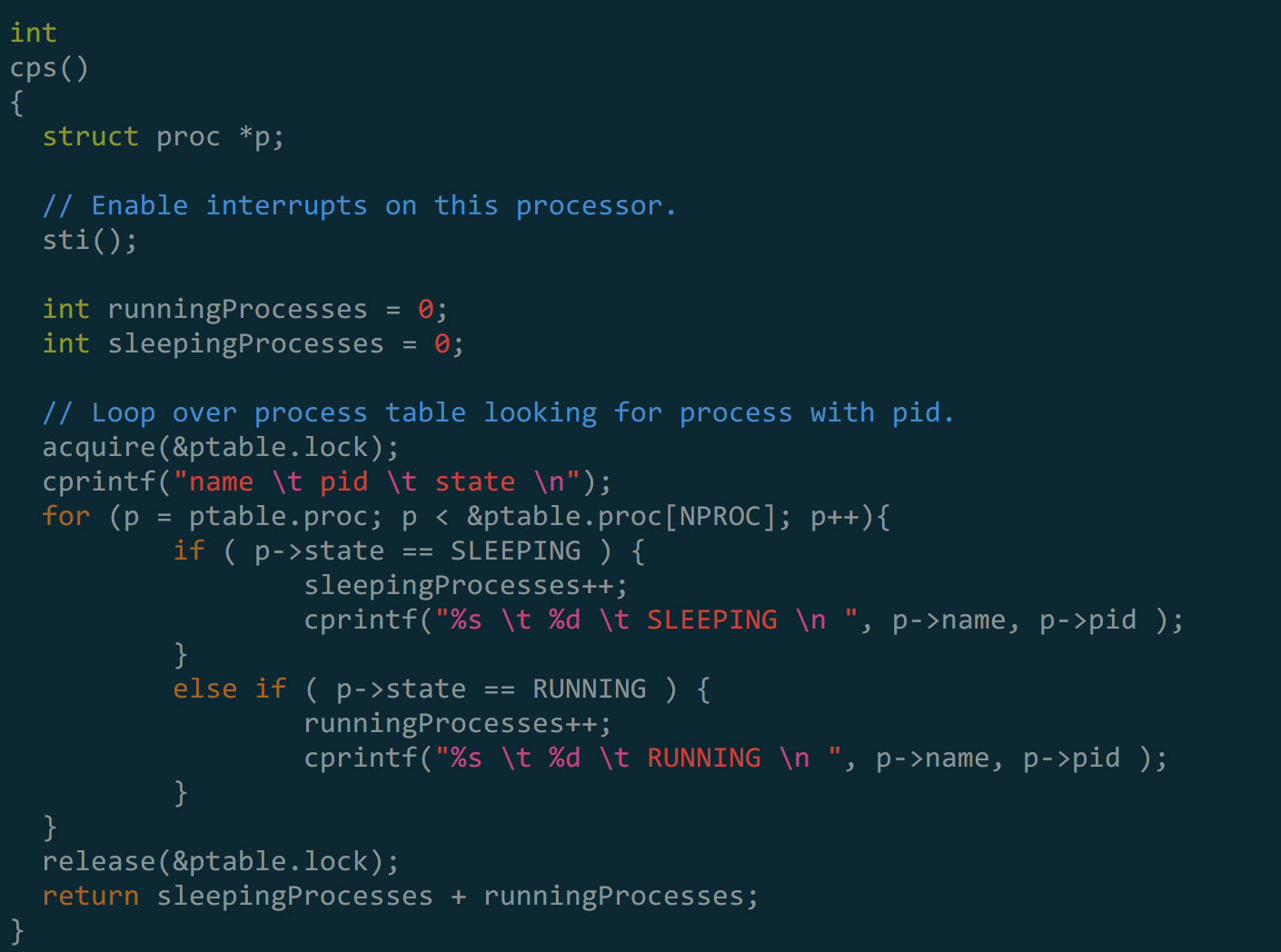




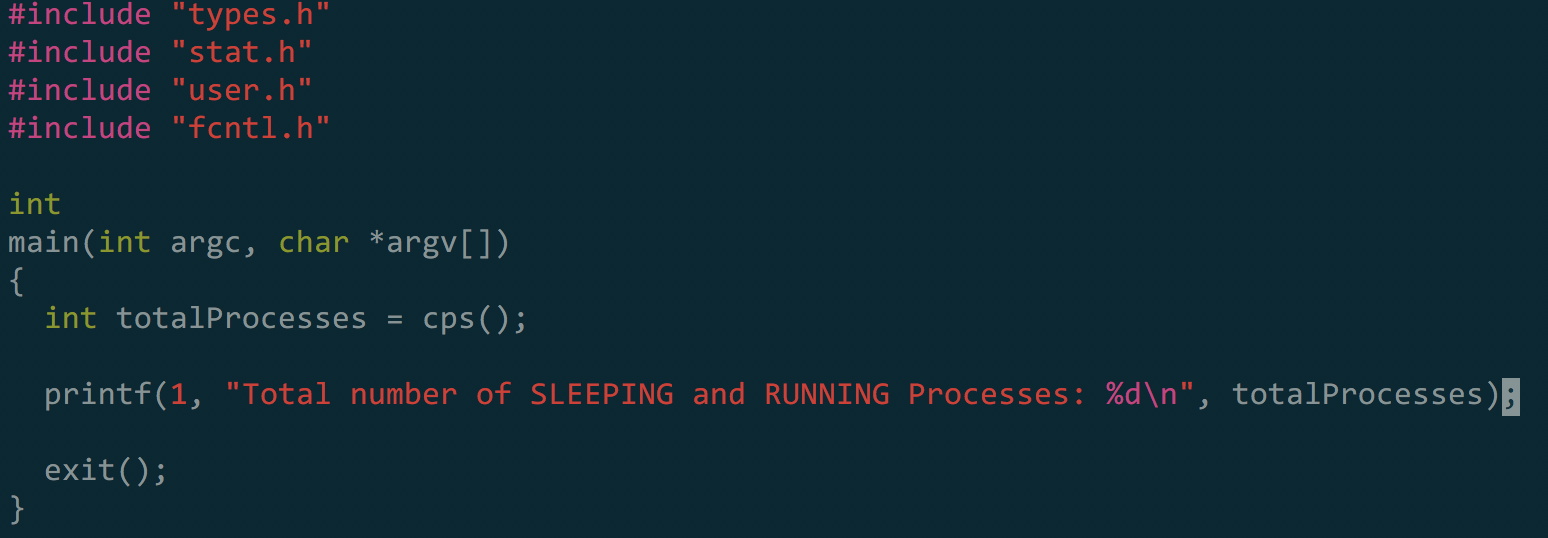
1. After running *$make qemu-nox*:



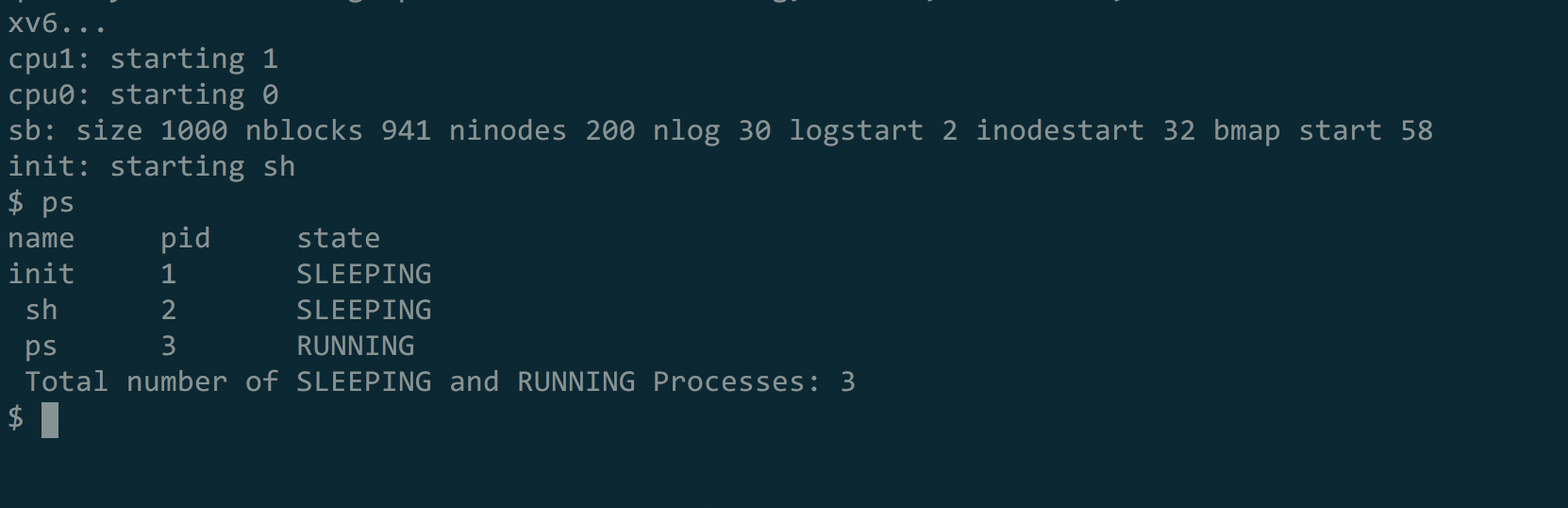
1. Modified *proc.c* code:



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



1. Output:



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