CECS 326 Assignment 5 (5 points)

Due: 10/14/2020 for Wednesday Section

10/19/2020 for Monday Section

Submit by lab class time on BeachBoard

As you have come to understand, the shmp1.cpp and shmc1.cpp (or shmp1.c and shmc1.c) you compiled and ran in Assignment 4 have serious deficiency due to race condition. In this assignment you are to correct the problem using one of the semaphore mechanisms that Linux provides. Note that any corrections you make should not include the removal or changes of the sleep() calls in the shmc1 program.

Two implementations of semaphore are commonly available on most distributions of UNIX and Linux operating systems: System V and POSIX. In this assignment you will use the System V implementation. System V implementation includes system calls **semget()**, **semctl()**, and **semop()**, and a **struct** **sembuf** used in the **semop()** calls, all of which are defined in <**sys/sem.h**>. Details on the definition of these system calls and their use may be found on Linux man pages.

The program must run successfully on Linux.

Do the following for this assignment:

1. Add necessary synchronization code in your Assignment-4C/C++programs to correct problems due to race condition, and compile them into executables *shmp1* and *shmc1*, respectively. Make sure that sufficient and proper comments are included on the added code as well as the existing code.
2. Run your corrected version of *shmp1* (with *shmc1*) to make sure that the output is correct.
3. Submit on BeachBoard the two corrected programs, along with the *registration.h* file, a screenshot that shows successful compile of both programs as well as a successful run, and a cover page that provides your name, your student ID, course # and section, assignment #, due date, submission date, and a clear program description detailing what you have done for the correction. Format of the cover page should follow the cover page template on BeachBoard.
4. The programs must be properly formatted and adequately commented to enhance readability and understanding. Detailed documentation on all system calls are especially needed.

/\* registration.h \*/

/\* Header file to be used with

\* shmp1.cpp and shmc1.cpp

\*/

struct CLASS {

char class\_number[6];

char date[7];

char title[50];

int seats\_left;

};

/\* shmp1.cpp \*/

#include "registration.h"

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/wait.h>

#include <unistd.h>

#include <stdlib.h>

#include <iostream>

#include <stdio.h>

#include <memory.h>

using namespace std;

CLASS myclass = { "4321", "082620", "Operating Systems", 20 };

#define NCHILD 3

int shm\_init( void \* );

void wait\_and\_wrap\_up( int [], void \*, int );

void rpterror( char \*, char \* );

main(int argc, char \*argv[])

{

int child[NCHILD], i, shmid;

void \*shm\_ptr;

char ascshmid[10], pname[14];

shmid = shm\_init(shm\_ptr);

sprintf (ascshmid, "%d", shmid);

for (i = 0; i < NCHILD; i++) {

child[i] = fork();

switch (child[i]) {

case -1:

sprintf (pname, "child%d", i+1);

rpterror ((char \*)"fork failed", pname);

exit(1);

case 0:

sprintf (pname, "shmc%d", i+1);

execl("shmc1", pname, ascshmid, (char \*)0);

rpterror ((char \*)"execl failed", pname);

exit (2);

}

}

wait\_and\_wrap\_up (child, shm\_ptr, shmid);

}

int shm\_init(void \*shm\_ptr)

{

int shmid;

shmid = shmget(ftok(".",'u'), sizeof(CLASS), 0600 | IPC\_CREAT);

if (shmid == -1) {

perror ("shmget failed");

exit(3);

}

shm\_ptr = shmat(shmid, (void \* ) 0, 0);

if (shm\_ptr == (void \*) -1) {

perror ("shmat failed");

exit(4);

}

memcpy (shm\_ptr, (void \*) &myclass, sizeof(CLASS) );

return (shmid);

}

void wait\_and\_wrap\_up(int child[], void \*shm\_ptr, int shmid)

{

int wait\_rtn, w, ch\_active = NCHILD;

while (ch\_active > 0) {

wait\_rtn = wait( (int \*)0 );

for (w = 0; w < NCHILD; w++)

if (child[w] == wait\_rtn) {

ch\_active--;

break;

}

}

cout << "Parent removing shm" << endl;

shmdt (shm\_ptr);

shmctl (shmid, IPC\_RMID, (struct shmid\_ds \*) 0);

exit (0);

}

void rpterror(char \*string, char \*pname)

{

char errline[50];

sprintf (errline, "%s %s", string, pname);

perror (errline);

}

/\* shmc1.cpp \*/

#include "registration.h"

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/wait.h>

#include <unistd.h>

#include <stdlib.h>

#include <iostream>

#include <stdio.h>

#include <memory.h>

using namespace std;

CLASS \*class\_ptr;

void \*memptr;

char \*pname;

int shmid, ret;

void rpterror(char \*), srand(), perror(), sleep();

void sell\_seats();

main(int argc, char\* argv[])

{

if (argc < 2) {

fprintf (stderr, "Usage:, %s shmid\n", argv[0]);

exit(1);

}

pname = argv[0];

sscanf (argv[1], "%d", &shmid);

memptr = shmat (shmid, (void \*)0, 0);

if (memptr == (char \*)-1 ) {

rpterror ((char \*)"shmat failed");

exit(2);

}

class\_ptr = (struct CLASS \*)memptr;

sell\_seats();

ret = shmdt(memptr);

exit(0);

}

void sell\_seats()

{

int all\_out = 0;

srand ( (unsigned) getpid() );

while ( !all\_out) { /\* loop to sell all seats \*/

if (class\_ptr->seats\_left > 0) {

sleep ( (unsigned)rand()%5 + 1);

class\_ptr->seats\_left--;

sleep ( (unsigned)rand()%5 + 1);

cout << pname << " SOLD SEAT -- "

<< class\_ptr->seats\_left << " left" << endl;

}

else {

all\_out++;

cout << pname << " sees no seats left" << endl;

}

sleep ( (unsigned)rand()%10 + 1);

}

}

void rpterror(char\* string)

{

char errline[50];

sprintf (errline, "%s %s", string, pname);

perror (errline);

}