Brian Powell

012362894

CECS 326 Sec 05 5288

Assignment 4

Due: 05/03/2018

Submitted: 05/03/2018

Program Description

In assignment 3 we noticed a race condition between the child process because there wasn’t a mutex lock creating mutual exclusion. To remedy that in Assignment 4 we had to create a semaphore variable and use that to make sure only one child is entering the critical section at a time. To do this I used the POSIX version of semaphore and ran into a couple of challenges I had to face.

First, using POSIX isn’t as straightforward as using the System V version of semaphore, in compilation I had to add the -lpthread and -ltr tags to g++ so that I could include the POSIX runtime libraries and even use the semaphores.

Once I figured all that out though, it was pretty straightforward, from class we learned that a semaphore requires two atomic functions to be useful. Using both sem\_wait() to decrement a semaphore value if it is one, to allow it to enter the critical section and using sem\_post() on the semaphore after it has finished the critical section to allow it to be ready to enter again.

There was a little bit of housekeeping I had to do to initialize the semaphore such as using sem\_open() to create the semaphore in the shared memory space being used by the application. As well as, calling sem\_close() to do housecleaning when I’m all done with the program.

#!/bin/bash

#./execute.sh

#shell script to compile all files in Linux Bash

g++ shmp1.cpp -o shmp1 -lpthread -lrt

g++ shmc1.cpp -o shmc1 -lpthread -lrt

#-lpthread is the thread library snecessary for POSIX compilation in Linux

#-lrt is the realtime extensions library also needed for POSIX compilation in Linux

/\*

Brian Powell 012362894

CECS 326 - Shui Lam

Assignment 4

The implementations of semamphore are commonly available on most distributions of UNIx and Linux operating systems. The system V

implementation includes semget(), semctl(), and semop(), and a struct sembuf used in the semop() calls, which are defined in <sys/sem.h>.

The POSIX implementation includes sem\_wait(), sem\_post(), and sem\_open(), sem\_close() & sem\_unlink() for named semaphore and sem\_init() &

sem\_destroy() for unnamed semaphore, which are definted in <semaphore.h>. You may use either one of these implementations. Details on

the definition of these functions and their use may be found on Linux man pages.

File: registration.h

\*/

struct CLASS {

    char class\_number[6]; //number of class

    char date[7]; //date of class

    char title[50]; //title of class

    int seats\_left; //number of seats left in class

};

/\*

Brian Powell 012362894

CECS 326 - Shui Lam

Assignment 4

File:shmp1.cpp

\*/

//includes

#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>

#include <semaphore.h>

#include <fcntl.h>

#include <time.h>

using namespace std;

//global variables

CLASS myclass = { "1001", "120186", "Operating Systems", 15 }; //initialize class variables

#define NCHILD  3

#define SNAME "/mutex"

sem\_t \*mutex = sem\_open(SNAME, O\_CREAT, 0600, 3);

//function definitions

int     shm\_init( void \* );

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

void    rpterror( char \*, char \* );

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

{

    //creates child array of size NCHILD,

    //initiates i, shmid, semid

    int     child[NCHILD], i, shmid, semid;

    void    \*shm\_ptr; //creates void pointer for shared memory

    char    ascshmid[10], ascsemid[10], pname[14]; //declares char arrays for the process name and shared memory id's

    strcpy (pname, argv[0]);//copies calling process id to pname

    shmid = shm\_init(shm\_ptr); //initilizes shared memory and passes the ptr to shmid

    sprintf (ascshmid, "%d", shmid); //formats the shared memory id to signed decimal integer and copies is to ascshmid

    //create child processes to consume the shared memory locations

    for (i = 0; i < NCHILD; i++) {

        child[i] = fork(); //creates child processes

        switch (child[i]) { //either throws error or forks to shmc1 consumer file

        case -1: //fork error

            rpterror ("fork failure", pname);

            exit(1);

        case 0: //fork returns successful

            sprintf (pname, "shmc%d", i+1); //print child name

            execl("shmc1", pname, ascshmid, (char \*)0); //execute shmc1 and pass the child name, and shared memory location, null pointer

            perror ("execl failed"); //if execl failes

            exit (2); //exit case

        }

    }

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

}

//function defintion to initialize a shared memory space

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

{

    //return address of shared memory

    int shmid;

    //allocates a System V shared memory segment(key\_t key, size\_t size, int shmflg)

    //shmflg if IPC\_CREATE | IPC\_EXCL check for shared memory if no then create

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

    if (shmid == -1) { //checks for shared resource error

        perror ("shmget failed");

        exit(3); //exits with signal 3

    }

    //System V shared memory operations (int shmid, const, const void \*shmaddr, int shmflg)

    //attaches the System V shared memory segment indetified by shmid to the address space

    //of the calling process The attaching address is speficied by shamddr

    shm\_ptr = shmat(shmid, (void \* ) 0, 0); //smaddr set to null pointer, finds suitable unuded address to attach segment to

    if (shm\_ptr == (void \*) -1) { //checks for shmat return error

        perror ("shmat failed");

        exit(4); //exits with signal 4

    }

    //memcpy will copy memory area (void \*dest, const void \*src, size\_t n);

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

    return (shmid); //returns shmid data address to point to

}

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

{

    //variable definitions for wait

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

    while (ch\_active > 0) { //wraps up all children processes

        wait\_rtn = wait( (int \*)0 ); //wait until last child returns

        for (w = 0; w < NCHILD; w++) //cycles through active childs

            if (child[w] == wait\_rtn) { //checks to see if active child is waiting

                ch\_active--; //if child is waiting it will consume the child

                break;

            }

    }

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

    //detaches the shared memory segment located at the address specified by shmaddr

    //points to shared memory pointer created in shm\_init

    shmdt (shm\_ptr);

    sem\_close(mutex); //cleaned mutex variable

    //no error check occurs

    //performs the control operation specified by cmd on the System V shared memory

    //segment whose idetntifier is geven in shmid(int shmid, int cmd, stuct shmid\_ds \*buf)

    //IPC\_RMID marks the segment to be destroyed

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

    exit (0);

}

//prints error line of fork failure and the process name that failed to fork

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

{

    char errline[50]; //error message of length char[50]

    sprintf (errline, "%s %s", string, pname); //converts the string from a char array to string array to print

    perror (errline); //prints error line

}

/\*

Brian Powell 012362894

CECS 326 - Shui Lam

Assignment 4

File: shmc1.cpp

\*/

//includes

#include "registration.h"

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/shm.h>

#include <sys/wait.h>

#include <unistd.h>

#include <stdlib.h>

#include <iostream>

#include <stdio.h>

#include <memory.h>

#include <semaphore.h>

using namespace std;

//global variables

#define SNAME "/mutex"

CLASS   \*class\_ptr; //points to class data type

void    \*memptr; //points to memory location

char    \*pname; //points to process name

int     shmid, ret; //shared memory id, return id

sem\_t \*semaphore = sem\_open(SNAME, 0);

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

void sell\_seats();

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

{

    if (argc < 2) { //checks if child names are less than 2

        fprintf (stderr, "Usage:, %s shmid\n", argv[0]); //prints error to standard output stream

        exit(1); //exits

    }

    pname = argv[0]; //assign pname to the first argument

    sscanf (argv[1], "%d", &shmid); //scanf reads from character string from argv[1] to the memory location

    memptr = shmat (shmid, (void \*)0, 0); //attached memptr to the shared memory segment identified by shmid

    if (memptr == (char \*)-1 ) { //checks to see if shared memory location exists

        rpterror ("shmat failed"); //print error

        exit(2); //exit with error code 2

    }

    class\_ptr = (struct CLASS \*)memptr; //assign class\_ptr of type CLASS to the memptr location

    sell\_seats(); //call sell\_seats function

    ret = shmdt(memptr); //detached piece from shared memory location and assign it to ret

    exit(0); //exit with code 0

}

void sell\_seats()

{

    int all\_out = 0; //variable for all seats empty

    srand ( (unsigned) getpid() ); //get random process id

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

        sem\_wait(semaphore); //wait for semaphore to be released

        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;

            sem\_post(semaphore); //CS complete, release semaphore

        }else{

            sem\_post(semaphore); //CS complete, release semaphore

            all\_out++;

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

        }

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

    }

}

void rpterror(char\* string)

{

    char errline[50]; //error message of length char[50]

    sprintf (errline, "%s %s", string, pname); //converts error message from a char to string array to print

    perror (errline); //prints error line

}