**COP 4600-001 Operating System**

**Project #2 (Semaphores)**

**October 8, 2017**

**Project Objectives:**

The purpose of this project is to give students an opportunity to experiment with process synchronization mechanisms.

**Total points Available:** 100

**Due:** October 8, 2017, 11:59 pm

**Project Description:**

Project Description:

In this assignment, a memory location is shared by four processes. Each process independently tries to increase the content of the shared memory location from 1 to a certain value by increments of one. Process 1 has a target of 100000, Process 2’s target is 200000, Process 3 has a target of 300000, and the goal of 4 is 500000. When the program terminates, therefore, the shared memory variable will have a total of 1100000 (i.e. this value will be output by whichever of the four processes finishes last).

In this project, you are to modify the assignment1 to protect the critical section using semaphores.

After all the children have finished, the parent process should release the shared memory and semaphores and then terminate. Use the **"wait"** function so that the parent knows precisely when each of the children finishes. The parent should print the process ID of each child as the child finishes execution. Then it should release shared memory, semaphores, and print "End of Simulation".

**Sample output**

From Process 1: counter = 330547.

From Process 2: counter = 447860.

From Process 3: counter = 600059.

From Process 4: counter = 1100000

Child with ID 2412 has just exited.

Child with ID 2411 has just exited.

Child with ID 2413 has just exited.

Child with ID 2414 has just exited.

End of Simulation.

**Submitting your assignment**

* Submission via Canvas’s Assignment.
  + It is your responsibility to submit these assignments in a timely fashion.
* All files should be zipped together.
* There should be a readme file explaining in detail the exact steps to be taken to compile and execute the code files and the title page
* Testing of this work should be done only on the CS lab machines. Please make sure these machines are not locked up due to your code. The execution for grading purposes will be done on the lab machines.
* In case of any code errors, partial credit may be offered based on the code and documentation.
* A report that presents the performance evaluation of your solution.
  + The report should be properly formatted (an academic format style, such as ACM or IEEE being preferred) and contain quantitative data along with you analysis of these data.

**Late Submission Policy**

* Late work will be not accepted.

**Grading Criteria:**

* Minus 90% if code does not compile. Minus 70% if it compiles but does not run.
* If the code compiles and runs, further deductions will be made for the following:
  + Minus 40% if 4 child processes are not created
  + Minus 30% if the semaphore is not created.
  + Minus 30% if the children fail to modify the shared variable.
  + Minus 30% if parent does not release shared memory and semaphores before ending.
  + Minus 10% if the report is not written
  + Minus 10% if children do not print out their results.
  + Minus 10% if parent does not print each time a child finishes.
  + Minus 20% if anything other than the critical section is protected by the semaphores
  + Minus 5% if no comments.
  + Minus 3% if your name is not included in comments on the top of your source code

**Useful codes:**

The semaphore structures and functions:

#include <sys/sem.h>

// semaphore key

#define SEMKEY ((key\_t) 400L)

// number of semaphores being created

#define NSEMS 1

// GLOBAL

int sem\_id;// semaphore id

// semaphore buffers

static struct sembuf OP = {0,-1,0};

static struct sembuf OV = {0,1,0};

struct sembuf \*P =&OP;

struct sembuf \*V =&OV;

// semapore union used to generate semaphore

typedef union{

int val;

struct semid\_ds \*buf;

ushort \*array;

} semunion;

// POP (wait()) function for semaphore to protect critical section

int POP()

{

int status;

status = semop(sem\_id, P,1);

return status;

}

// VOP (signal()) function for semaphore to release protection

int VOP()

{

int status;

status = semop(sem\_id, V,1);

return status;

In the main()

int value, value1;

semunion semctl\_arg;

semctl\_arg.val = 1;

/\* Create semaphores \*/

sem\_id = semget(SEMKEY, NSEMS, IPC\_CREAT | 0666);

if(sem\_id < 0) printf("Error in creating the semaphore./n");

/\* Initialize semaphore \*/

value1 =semctl(sem\_id, semnum, SETVAL, semctl\_arg);

value =semctl(sem\_id, semnum, GETVAL, semctl\_arg);

if (value < 1) printf("Eror detected in SETVAL.\n");

/\* De-allocate semaphore \*/

semctl\_arg.val = 0;

status =semctl(sem\_id, 0, IPC\_RMID, semctl\_arg);

if( status < 0) printf("Error in removing the semaphore.\n");

**Development Environment**

You may write your program using any available editor Nano, Emacs, Vi or whatever editor you are most comfortable with, BUT, it must compile with gcc and be executable on one of the CS machines:

To login to these machines remotely, download PUTTY by going to: http://the.earth.li/~sgtatham/putty/latest/x86/putty-0.58-installer.exe

Then after the download, execute PUTTY. Also you need to download and install Junos pulse from USF VPN (https://www.net.usf.edu/vpn/Windows/ = for windows or https://www.net.usf.edu/vpn/index.php = for other OS types). Click PUTTY and enter one of the lab machines for the Host Name. Their hostnames are osnode[01-16].csee.usf.edu. Then enter login name and password (your netid)

**Hints:**

Build your project in an incremental fashion. Attempt to meet each objective before moving on to the next.

**Some useful UNIX commands:**

**The UNIX commands, needed for the project, are given at the end of lecture 4. Also you can use link**

<https://beej.us/guide/bgipc/output/html/multipage/shm.html>

Find more information about these command and options used by them by using UNIX manual or by simply using man command.