Assignment 4

Kyle Pamintuan

CECS 326

Professor Ratana Ngo

12/11/17

Overview:

The purpose of this programming assignment is to help us better understand the concept of child processes and semaphores.

First, we have the parent spawn four child processes. Each child constantly generates random integers and doesn’t stop until it generates an integer that is less than 100 OR generates a factor of U (U=827,395,609) or V (V=962,094,883).

The requirements for the program states that only 2 processes be running while the other 2 processes wait. Also, running processes can only be operating on one variable (either U or V). In other words, running processes can’t be operating on the same variable. To fulfill these requirements, we will have to use semaphores and a shared memory buffer.

My program exhibits starvation because while 2 child processes are making progress by making calculations on U or V, the other 2 child processes are waiting and not making progress. Also, my program does not exhibit a deadlock because I’ve used semaphores and a toggle variable in shared memory that ensures no occurrence of a deadlock.

**Assignment4.cpp**

#include <sys/types.h>

#include <sys/stat.h>

#include <sys/wait.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include "semaphore.h"

#include <iostream>

using namespace std;

// Size of shared memory buffer

const int BUFFSIZE = 1;

// Name of semaphore variable

enum {semVar};

// Process functions

void parent\_cleanup(SEMAPHORE &, int);

void child\_proc(SEMAPHORE &, int \*);

int main()

{

int shmid;

int \*shmBUF;

srand(time(NULL));

// Create 1 set of semaphores

SEMAPHORE sem(1);

// Initial value: semVar = 2

sem.V(semVar);

sem.V(semVar);

// Create shared memory buffer

shmid = shmget(IPC\_PRIVATE, BUFFSIZE\*sizeof(int), PERMS);

shmBUF = (int \*)shmat(shmid, 0, SHM\_RND);

// shmBUF[0] will serve as TOGGLE varaible

\*(shmBUF+(0%BUFFSIZE)) = 1;

// Spawn child processes

pid\_t pid;

for (int i = 0; i < 4; i++)

{

pid = fork();

if(pid == 0)

break;

}

for(;;)

{

if(pid) // parent process

{

string answer;

//cout << "Quit?" << endl;

//cin >> answer;

if (answer == "!wq")

{

parent\_cleanup(sem, shmid);

break;

}

}

else if(pid == 0) // child process

{

child\_proc(sem, shmBUF);

}

}

exit(0);

}

// Child Process Function

void child\_proc(SEMAPHORE &sem, int \*shmBUF)

{

sem.P(semVar);

long int U = 827395609;

long int V = 962094883;

// Decide which variable (U or V) to use for the dividend

long int dividend;

if (\*(shmBUF+(0%BUFFSIZE)) == 1)

{

cout << getpid() << ": Operating on U" << endl;

dividend = U;

\*(shmBUF+(0%BUFFSIZE)) = 0; //toggle

}

else if (\*(shmBUF+(0%BUFFSIZE)) == 0)

{

cout << getpid() << ": Operating on V" << endl;

dividend = V;

\*(shmBUF+(0%BUFFSIZE)) = 1; //toggle

}

// Perform calculations

int randomNum = (rand()%10000000) + 1;

int result = dividend%randomNum;

while (randomNum > 100 || result != 0)

{

randomNum = (rand()%10000000) + 1;

result = dividend%randomNum;

}

cout << getpid() << ": finished" << endl;

sem.V(semVar);

}

// Parent Cleanup Function

void parent\_cleanup (SEMAPHORE &sem, int shmid)

{

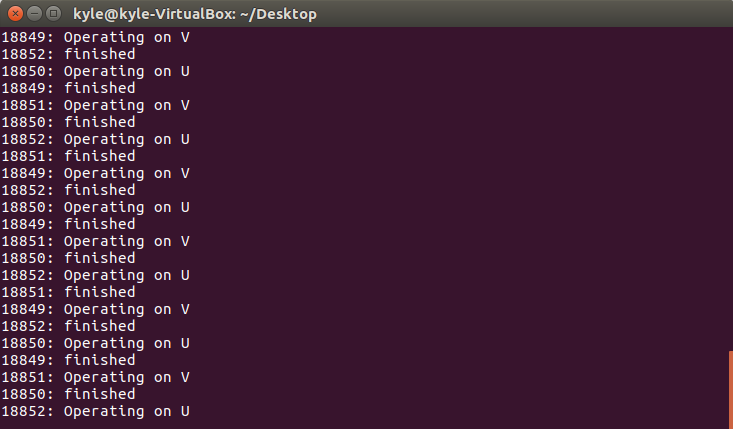
kill(getppid(), SIGKILL);

shmctl(shmid, IPC\_RMID, NULL);

sem.remove();

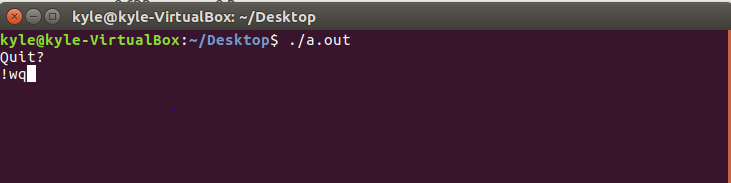
}

**Output (Children):**



Child processes run constantly…

**Output (Parent):**



Parent process terminates and terminal closes after entering “!wq” into the console.