CECS 326 Project 4

Programming with Semaphores and shared memory

Andrew Myer

012939730

[andrewmyerhb@gmail.com](mailto:andrewmyerhb@gmail.com)

5/7/18

In this project, we were required to use shared memory and semaphores to coordinate processes. The goal was to fork four child processes, and have the processes compute whether a random integer is a factor of 827395609 or 962094883. 2 processes can run concurrently while the other two wait, but no two processes can work on the same number.

In order to create four child processes, I used the fork function where the parent id is stored into an array to be used for the kill command, and the child process section ran an endless loop so they didn’t access the code after. To allow only two processes to run at once, I initialized a semaphore equal to 2. So when more than two processes tried accessing the semaphore, they lock until a spot on the semaphore is freed up. To make sure no two processes are working on the same number I created a shared Boolean that is set to true if the first number is available. If it is not available, then the second number must be available since there are only two numbers. While the child processes are doing there processes, the parent process waits for the user to enter the string “!wq” and then enters the command to kill all the child processes using the array of stored parent ids. The parent then exits the program.

This implementation won’t cause a deadlock, however it can cause starvation. This is because it favors the U process over the V process. On the off chance that the U process ends quicker than the other process gets executed, it can loop on the U process and never give the V process a chance.

This project was a good example of using shared memory and semaphores to coordinate processes that need to have critical sections.

|  |
| --- |
| /\* |
|  | CECS 326 |
|  | Project 4: Programming with semaphores and shared memory |
|  | Andrew Myer |
|  | 012939730 |
|  | \*/ |
|  |  |
|  | #include <iostream> |
|  | #include <stdio.h> |
|  | #include <stdlib.h> |
|  | #include <unistd.h> |
|  | #include <limits.h> |
|  | #include <sys/types.h> |
|  | #include <sys/stat.h> |
|  | #include <sys/wait.h> |
|  | #include <sys/ipc.h> |
|  | #include <sys/shm.h> |
|  | #include "semaphore.cpp" |
|  | using namespace std; |
|  |  |
|  | enum{UandV}; |
|  |  |
|  | void critSec(SEMAPHORE &, bool \*); |
|  | void parentProc(int killArray[], SEMAPHORE &, int shmid); |
|  |  |
|  | int main() { |
|  | /\*create shared memory for the boolean flag that is set to true |
|  | to show the u process is available\*/ |
|  | int shmid = shmget(IPC\_PRIVATE, sizeof(bool), PERMS); |
|  | bool \*shmBUF = (bool \*)shmat(shmid, 0, SHM\_RND); |
|  | \*shmBUF=true; |
|  |  |
|  | // PID for the fork |
|  | pid\_t childPID; |
|  |  |
|  | //initalize semaphore |
|  | SEMAPHORE sem(1); |
|  |  |
|  | //set semaphore to 2 |
|  | sem.V(UandV); |
|  | sem.V(UandV); |
|  |  |
|  | /\*array of childPID's to be used when the the parent process kills the |
|  | child processes\*/ |
|  | int killArray[4]; |
|  |  |
|  | //creates four child processes |
|  | for(int i=0;i<4;i++) |
|  | { |
|  | childPID = fork(); |
|  | if(childPID) //parent process |
|  | { |
|  | killArray[i]=childPID; |
|  | } |
|  | else //child process |
|  | { |
|  | while(true) //child processes can't proceed from here |
|  | { |
|  | critSec(sem,shmBUF); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | parentProc(killArray,sem,shmid); |
|  |  |
|  | return 0; |
|  | } |
|  | /\*------------------------------------------------------------------- |
|  | creates a random number in a loop until that number is divisible by |
|  | 100 or a multiple of uov. |
|  | param |
|  | int pov: the number for the P processes or the V process |
|  | return |
|  | null |
|  | --------------------------------------------------------------------\*/ |
|  | void proc(int uov) |
|  | { |
|  | int temp; |
|  | while(true) |
|  | { |
|  | temp=rand(); |
|  | if((temp%uov==0)||temp<100) |
|  | { |
|  | break; |
|  | break; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | /\*------------------------------------------------------------------- |
|  | Enters a critical section so that no other processes can work on the same |
|  | number as the current process. since sem(UandV) is initially set to 2 |
|  | allows for 2 processes to run concurrently |
|  | param |
|  | SEMAPHORE &sem: the semaphore object to be used to order the processes |
|  | bool \*shmBUF: the boolean flag to show if the U process is available |
|  | return |
|  | null |
|  | --------------------------------------------------------------------\*/ |
|  | void critSec(SEMAPHORE &sem, bool \*shmBUF) |
|  | { |
|  | int U=827395909; |
|  | int V=962094883; |
|  | sem.P(UandV); |
|  | bool u=\*shmBUF; |
|  | if(u)//u is available |
|  | { |
|  | \*shmBUF=false; //makes U unavailable for other processes |
|  | proc(U); |
|  | \*shmBUF=true; |
|  | } |
|  | else //u is not available, run V process |
|  | { |
|  | proc(V); |
|  | } |
|  | sem.V(UandV); |
|  | } |
|  | /\*------------------------------------------------------------------- |
|  | for the parent process to terminate all children processes, clean up |
|  | and exit semaphore and shared boolean when the user inputs "!wq" |
|  | param |
|  | int killArray[]: array with all the childIDs to terminate |
|  | return |
|  | null |
|  | --------------------------------------------------------------------\*/ |
|  | void parentProc(int killArray[],SEMAPHORE &sem, int shmid) |
|  | { |
|  | string end; |
|  | while(true) |
|  | { |
|  | cout<<"enter !wq to exit"<<endl; |
|  | getline(cin,end); |
|  | if(end=="!wq") //termination command |
|  | { |
|  | cout<<"exiting..."<<endl; |
|  | for(int i=0;i<4;i++) |
|  | { |
|  | kill(killArray[i],SIGTERM); //command to kill child process |
|  | } |
|  | shmctl(shmid, IPC\_RMID, NULL); |
|  | sem.remove(); |
|  | exit(0); //process terminates, close out program |
|  | } |
|  | } |
|  | } |