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.