Project 3: Short-Term Process Scheduler

CST-315-99 (Operating Systems)

Cameron Verini

11/27/2022

**Project Description:**

Within this project, we were tasked with updating our program from our previous projects by adding in a short-term process scheduler based on one of the scheduling algorithms discussed in class.

**Methodology/Approach:**

In order to decide which processes would be run, I allowed the user to choose which processes to run through the “run” command. The processes chosen will enter an array with their indexes being based on the order they were chosen. Because I have chosen to implement a first-come, first-served scheduling algorithm, these indexes will also determine the order in which the processes occur. Once the user has finished deciding which processes to run, the array of processes is sent to the first-come, first-served scheduling algorithm. In there, the processes will be run in the order they were input. A sleep repeatedly occurs until the previous process finishes, at which point a single more sleep occurs to prevent outputs from overlapping. This prevents the processes from all occurring at once.

**11 Points:**

1. How are you implementing the queue of processes?

The queue of processes is based on the order in which the user chooses to run the processes.

1. How is the scheduler assigning a reasonable share of the CPU to each process?

Before the processes run, the program estimates the amount of time that the processes will take. Considering that the processes are nearly identical in this example, they are all estimated to be around the same time, which is essentially the amount of time it would take for the program to print a statement.

3. How do you maximize efficiency (i.e. minimize CPU idle time)?

Considering that the processes should technically take a very small amount of time more than the estimated time, the CPU idle time should be very small. It is probably more likely for multithreading to occur than it is for there to be idle time.

4. How do you ensure quick reaction to user input?

The user’s input receives no delay. There is nothing to hamper it.

5. How do you ensure that the same job takes approximately the same amount of time when repeated?

So long as the processes themselves don’t change, they will take approximately the same amount of time. The only exception to this that I foresee is the CPU suddenly tanking and causing computation times to increase, but my time estimation takes that into account.

6. How do you minimize the overhead (e.g. switching states)?

Switching states only occurs in process that the user has specifically chosen to run. Otherwise, they simply remain in the ready state.

7. How do you account for efficient resource utilization?

Resources are only used as they are needed and processes only occur one at a time.

8. How do you avoid starving a process?

After a certain period of time, if a process is starved the program will exit.

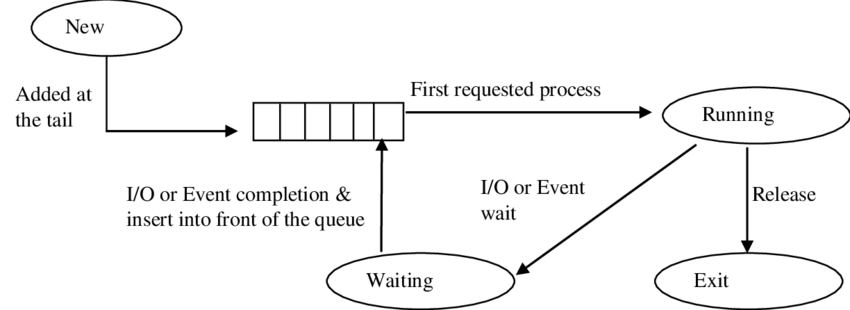
9. How do you ensure that the shell does not collapse under increased load, but its performance is gradually degraded?

The user is in direct control over how large of a load the shell undergoes.

10. Is your scheduler preemptive or cooperative?

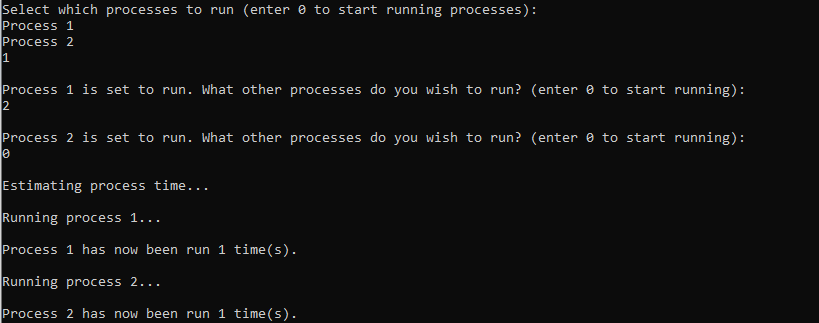
Preemptive.

11. Provide the processes queuing diagram

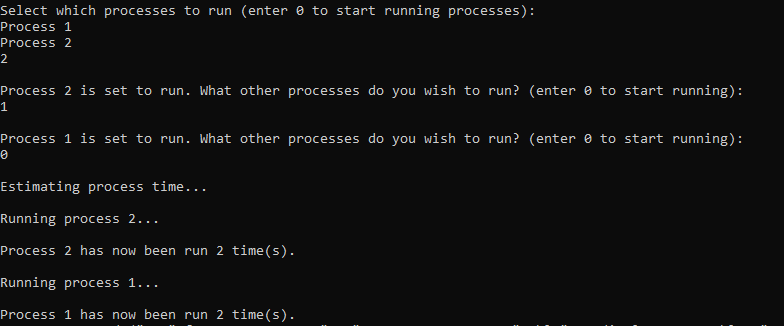


**Test and Validate:**

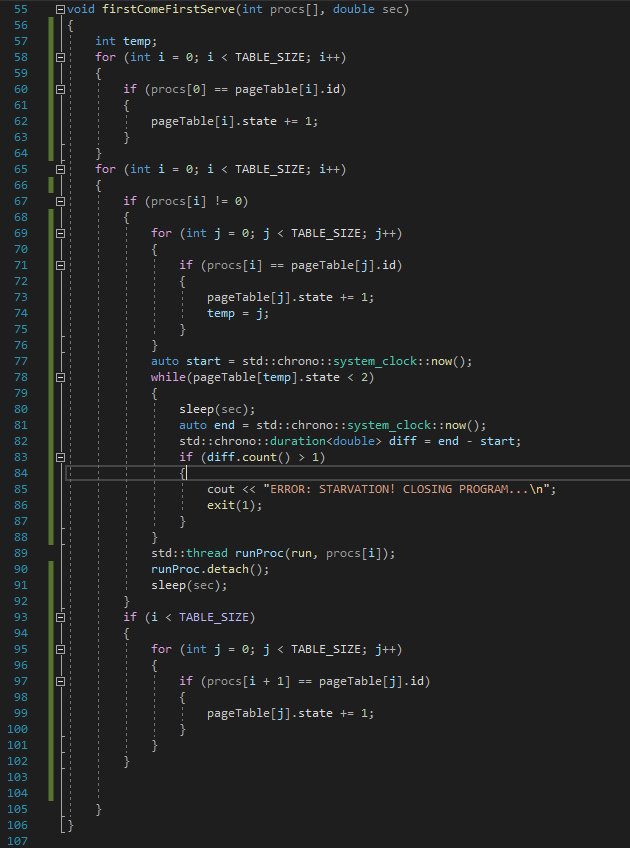
Process 1 scheduled before process 2:



Process 2 scheduled before process 1:



Scheduling Algorithm:



The program has no major issues with executing. The only minor error that occurs is that the message prompting users to enter commands is printed twice after running processes.