CSE 410
Summer 2018

Computer Project #2 Release
Date: 25th May, 2018 Due
Date: 1st June, 2018

Assignment Overview

This assignment focuses on process management within an operating system, including processor scheduling. You will design and implement the C/C++ program which is described below.

It is worth 30 points and must be completed no later than 11:59 PM on Friday, 6/1.

Assignment Deliverables

The deliverables for this assignment are the following files:

project02.makefile – the makefile which produces
project02.student.c – the source code file for your solution

Be sure to use the specified file names and to submit your files for grading via the CSE Handin system before the project deadline.

Assignment Objectives

To familiarize you with the use of semaphores, thread usage, and to get a better understanding of process management

Assignment Specifications

The program will simulate the steps that an operating system takes to manage user processes. The operating system uses a five-state process model: New, Ready, Running, Blocked, and Exit.

When a process is created, it is placed in the New state; it moves into the Ready state when a Process Control Block (PCB) becomes available. When a process is dispatched, it moves into the Running state.

While in the Running state, a process may issue two types of requests: a request to halt (which is issued when a process has completed its processing and is ready to leave the system), and a request to block (which is issued when the process must wait for some event, such as an I/O operation).

When a process becomes unblocked, it moves from the Blocked state to the Ready state.

1. Your program will process a file which contains the simulation parameters and the process stream. The first four lines of the file will contain the simulation parameters (one item per line):

a) Number of CPUs in the system (the integer value 1 for this assignment)
b) Number of PCBs in the system (the integer value 1 for this assignment)
c) Length of the simulation (in ticks)
d) Short-term scheduling algorithm (the string "FCFS" for this assignment)

2. Exactly one line will follow the simulation parameters to represent the process stream. That line will contain the following fields, separated by blanks:

a) Process identification number
b) Priority (from 1 to 3)
c) Number of CPU bursts
d) Burst time (in ticks)
e) Blocked time (in ticks)
f) Arrival time (in ticks since start of simulation)

The number of CPU bursts is an integer value representing the number of times that the process needs access to the CPU.

The burst time is an integer value representing the length of time that the process uses the CPU before issuing a service request.

The blocked time is an integer value representing the length of time that the process is blocked after issuing a service request.

3. When the process terminates, the program will display the following information about that process:

a) All process parameters (items (2a) to (2f) above)
b) Departure time (in ticks since start of simulation)
c) Cumulative time in the New state (in ticks)
d) Cumulative time in the Ready state (in ticks)
e) Cumulative time in the Running state (in ticks)
f) Cumulative time in the Blocked state (in ticks)
g) Turnaround time (in ticks)
h) Normalized turnaround time

*** The following (i.e. #4) would be nice to have, but not a requirement yet.
4. The program will accept two command-line arguments: an integer representing the value of N and the name of the input file.

After every N ticks of simulated time, the program will display the following information:

a) Process ID of each process in the New state
b) Process ID of each process in the Ready state
c) Process ID of each process in the Running state
d) Process ID of each process in the Blocked state

The program will only display the information if N is a positive integer value.

5. The program will include appropriate error-handling.

Assignment Overview (or additional specs)

For this class, we're going to do something a bit different from the previous classes. We're going to implement semaphores to synchronize the process management stages. Additionally, we'll be using threads to represent the various stages. Below is a general, and probably incomplete, overview of this project.

*** All queues can be global
NewQueue: FIFO
// FreePCBQueue: NA
ReadyQueue: FIFO
RunningState: NA
BlockedQueue: FIFO
ExitQueue: FIFO

Global Variables
Int Time = 0;
Int FreePCB = 5

Semaphores
SemTimer (1)
semRunning(0)
semBlocked(0)
semReady(0)
semExit(0)

ClockThread
While
    Sleep(2) // number susceptible to change
    Time++;
    semTimer.signal()

NewThread
While
    semTimer.wait()
    Check to add new process to NewQueue, add if new process starts at time N
    If free PCB available, add process to ReadyQueue
        Decrement FreePCBQueue by one
        Remove appropriate process from NewQueue insert into ReadyQueue
    semBlocked.signal()

Blocked

```
While
        semBlocked.wait()
        for each process in BlockedQueue
                increment process.currentblocktime
                if( process.currentblockedtime == process.blockedtime)
                        process.currentblockedtime = 0
                        remove process from BlockedQueue add to ReadyQueue
        semRunning.signal()
```

Running

```
While
        semRunning.wait()
        increment process.currentRuntime
        if(currentbursttime == bursttime)
                vacate RunningState
                increment process.burstsRun
                if(process.burstsRun == process.totalBursts)
                        place process in ExitQueue
                else
                        add process to BlockedQueue
        semReady.signal()
```

Ready

```
While
        semReady.wait()
        if(RunningState is vacated)
                remove appropriate process from ReadyQueue
                process.currentRuntime = 0
                add process to RunningState
                semExit.signal()
```

Exit

```
While
        semExit.wait()
        print out data as specified above
        increment FreePCB
        free any resources associated with running process as it now exits
```

Assignment Notes

1. In a future project you will extend your program from Project #2, so you would be wise to develop your program in a modular fashion and to comment your source code.

2. For this assignment, you may assume that contents of the input file will be valid, and that there is exactly one processor, but multiple processes in the process stream.