CSE410: Operating Systems (Spring 2018)
Project #3: Uniprocessor Scheduling Simulator

Deadline
Thursday, March 15, 2018 at 11:59pm (Handin)

Assignment Goals
Get familiar with different scheduling algorithms and techniques.

Assignment Overview

In this project, you are required to design and implement a C++ program that simulates different scheduling algorithms on a single processor. These algorithms are: First Come First Serve (FCFS), Round Robin (RR) with different quantum values (its allowance of CPU time), Shortest Process Next (SPN), Highest Response Ratio Next (HRRN), and Shortest Remaining Time (SRT).

- Your program will be invoked as follows.
  ```
  ./myScheduler inputFile outputFile –option <arguments>
  ```

  Let "myScheduler" be the name of your executable. The program (myScheduler) will read the input file and then write the result to the output file as well as print information to the console ("inputFile", "outputFile", and "options" will be described in the following description). Options are the algorithm types (e.g. -FCFS, -SRT).

- **InputFile**
  The "inputFile" contains a list of processes with their corresponding arrival time and service time. Each line of the file represents a process, so the row index of each process is its processID. The first column is an integer which is the process arrival time, and the second column is also an integer denoting the service time.

  For example:
  ```
  0 3 // row index = 1 →  processID = 1 arrivalTime = 0 serviceTime = 3
  2 6 // row index = 2 →  processID = 2 arrivalTime = 2 serviceTime = 6
  . .
  ```
  The above example shows that the process 1 is arrived on time 0 and its service time is 3. Similarly, to the second line while the process 2 arrives on time 2 and takes 6 time slots to be served. Note that, the input file is sorted by arrival time, so you do not need to sort it again. The process with lower indices represents higher priority.

- **OutputFile**
  For the "outputFile", you are going to store a table of your execution results. For instance, the information stored in the "outputFile" would be as follows while we are running the simulator using **`–FCFS`** options with the "inputFile" mentioned above. 2

Process ID	Start Time	Finish Time	Turnaround Time	Tr/Ts
1	0	3	3	3/3
2	3	9	7	7/6

Note: T_r is the turnaround time, and T_s is the service time

You are not required to draw the grids and you can simply store those values of each column directly. This is what you really need to store in the "outputFile".

```
1 0 3 3 1.0
2 3 9 7 1.1
. . . . .
```

Assignment Specifications

In this project you are required to implement the following tasks (total 100 + 10 E.C. points).

1. –FCFS (14 points)

This option takes no arguments and applies the First Come First Serve algorithm to schedule the processes.

2. –SPN (16 points)

This option takes no arguments and applies the Shortest Process Next algorithm to schedule the processes.

3. –HRRN (+10 points) ***EXTRA CREDIT***

This option takes no arguments and applies the Highest Response Ratio Next algorithm to schedule the processes.

4. –SRT (18 points)

This option takes no arguments and applies the Shortest Remaining Time algorithm to schedule the processes.

5. –RR <Quantum Value> (20 points)

This option takes **Quantum Value** as the only argument and applies the Round Robin algorithm to schedule the processes.

6. –Basic Programming Requirements (12 points)

Programming requirements as shown in the Project Guideline (on D2L) and comments. Be warned: grading for comments will be strictly enforced. Comment well.

7. File Output (10 points)

See 'OutputFile' above.

8. Console Output (10 points)

The output (standard output, **not the "outputFile"**) of the program shows how the processor is shared by different processes. Since the processor is only allocated to one process at a time (or each time slot), the first column of the output file is the time slot number and the second column shows which process (Process ID) is served by the processor in that time slot. For instance, the information printed in the terminal screen would be as follows while we are running the simulator using **`–FCFS`** options with the "inputFile" mentioned above.

```
0 1
1 1
2 1
3 2
. .
8 2
```

Assignment Deliverables

A zip file named according to your NetID_project3 (e.g. john9999_project3.zip where “john9999” is the NetID) containing the following:

***.cpp** – all the .cpp files we provided in “Project_Skeleton” folder together with those your created if any.

****.h*** – all the .h files we provided in “Project_Skeleton” folder together with those your created if any.

Makefile – make file for generating an executable “myScheduler”

Readme – Include **ONLY** if there is something the grader needs to know.

The code **MUST RUN ON** “cse410.cse.msu.edu”. Any compilation error will result in a **ZERO** for the project. So please make sure your program is clear from any compilation and runtime errors before submission. Note that your project file should be submitted with specified file name via the “Handin” system.

Assignment Notes

1. You can find all the required source files by downloading the “Project3.zip” file on D2L. In this archive file, you can find the following items.
   - Makefile
   - *.cpp
   - *.h
   - File1 (This is an input file for testing that corresponds to Table 9.4 (O.S. book page 406) and its correct output is shown in Figure 9.5 (O.S. book page 407))
   - myScheduler (this is for demonstration, and you can play with it by executing it on “cse410.cse.msu.edu” for further understanding about the required tasks of this project)
     NOTE: do NOT hand in this file.

2. All the functions you have to implement for this project.
   - ProcessScheduler::WriteToFile()
   - ProcessScheduler::ExecuteFCFS()
   - ProcessScheduler::ExecuteSPN()
   - ProcessScheduler::ExecuteSRT()
   - ProcessScheduler::ExecuteHRRN() ***Extra Credit***
   - ProcessScheduler::ExecuteRR()
   - ProcessScheduler::PrintToScreen()

3. In situations where a process finishes and a new process arrives at the same time, treat the new process with priority (have the new process run first).

4. **Section 9.2 of the textbook includes all the mentioned algorithms in detail.**