EECE.4810/EECE.5730: Operating Systems

Spring 2018

Programming Project 3

Due <u>11:59 PM</u>, ~~Wednesday, 4/18/18~~ **Monday, 4/30/18**

1. Introduction

This project uses simulation to explore the effectiveness of the different scheduling algorithms described in class. While your program will not schedule actual processes running on your machine, it will model how each algorithm behaves when making scheduling decisions about a set of processes.

This assignment is worth a total of ~~100~~ **150** points. The grading rubric given in Section 4 applies to students in both EECE.4810 and EECE.5730.

This assignment was adapted, with permission, from an assignment written by Prof. Vinod Vokkarane for CIS 370 at UMass Dartmouth.

2. Project Submission and Deliverables

Your submission must meet the following requirements:

- You should submit three files:
  - `OS_program3.c` contains your `main()` function and global variables
  - `sched_sim.h` contains any structure definitions and function prototypes you need for the assignment
  - `sched_sim.c` contains the definitions of functions prototyped in the .h file. Global variables declared in `OS_program3.c` should be additionally declared as `extern` in this file—see the Project 2 specification for a brief description of the `extern` keyword.
- **Note: You may write your solution to this assignment in either C or C++; if submitting C++ files, your source file extensions should be `.cpp` instead of `.c`, `sched_sim.h` should contain class definitions, not structure definitions, and you should use the GNU C++ compiler `g++`, not `gcc`.**
- Programs must be submitted via e-mail sent to Dr. Geiger (Michael_Geiger@uml.edu), either as three separate attachments or as a .zip or .tar archive.
  - If you submit an archive file, please <u>do not</u> use the .rar format.
- You may work in groups of up to 3 on this assignment. If you work in a group, please list the names of all group members in a header comment at the top of each file in your submission, as well as in the email you use to submit your code.

3. Specification

General overview: Simulators allow a user to evaluate the effectiveness of a particular design or algorithm without fully implementing that hardware or software. In this case, you do not have to schedule actual processes to test how scheduling algorithms will handle different process workloads. Your simulator will model each of the following algorithms covered in class:

- First come, first served (FCFS)
- Shortest job first (SJF)
- Shortest time to completion first (STCF)
- Round robin (RR) with time quantum 2
- Non-preemptive priority scheduling

Input: Your program reads input from two sources:

Command line arguments: Your executable should take the following command line arguments:

- The name of an input text file containing information about the processes to be simulated
  - o A "text file" does not require a .txt extension. That description simply implies that the file's contents are human-readable. C/C++ programs can read or write files in two general formats—text or binary.
- The name of a text file that will contain your program's output
- The interval at which your program generates a snapshot of the scheduler state

For example, if your executable name is `prog3`, the following command runs the program with `test1.dat` as the input file, `out1.txt` as the output file, and a 10 cycle interval between scheduler snapshots: `./prog3 test1.dat out1.txt 10`

Input file: The input file contains the following information about all processes:

- CPU burst time
- Priority: a lower number indicates higher priority
- Arrival time: a process arriving at time T may start execution at time T + 1

Each line represents a single process. Your program must read every line in the file and store the information about each process. The file size is not given, so your program must read until it reaches the end of the file.

For example, a file describing the processes used in the example problem on slide 9 of Lecture 13 would have the following contents. Process ID numbers are not in the file; you should assign a unique number to each process, starting with process 0. Processes are listed in order of arrival:

```
10   1    0
3    4    0
7    2    2
1    2    4
5    3    6
```

3. Specification (continued)

Output: This program should print all output to the text file named in the command line arguments. The output takes three general forms:

Scheduler snapshots: A "snapshot" shows the state of the scheduler at a given time point. Each snapshot should be separated by the interval specified in the command line arguments and should contain the following information:

- Current time (for example: `t = 0`)
- Current CPU action, which can take one of three forms:
  - o Loading (preparing to start) a new process: List the full burst time of the process
  - o Running a process: List the remaining burst time of the process as it runs
  - o Finishing a process: If a snapshot is taken in a cycle in which a process finishes, describe both which process is finishing and which one is next to run
- The current state of the ready queue, shown in the order processes will run next. If there are no processes in the ready queue, indicate that it is empty.

For example, given the example file contents on the previous page and assuming FCFS scheduling and a time interval of 5, your first four snapshots would be as follows. Note that, as shown, a brief message naming the algorithm should be printed before the first snapshot:

```
***** FCFS Scheduling *****
t = 0
CPU: Loading process 0 (CPU burst = 10)
Ready queue: 1

t = 5
CPU: Running process 0 (remaining CPU burst = 5)
Ready queue: 1-2-3

t = 10
CPU: Finishing process 0; loading process 1 (CPU burst = 3)
Ready queue: 2-3-4

t = 15
CPU: Running process 2 (remaining CPU burst = 5)
Ready queue: 3-4
```

3. Specification (continued)

Output (continued): The other two forms of output are:

Algorithm summaries: For each scheduling algorithm, generate a summary that shows:

- A table of all processes, where each line includes a process ID, wait time, and turnaround time. The last line of the table should list the average wait and turnaround times.
- A list of how the processes were run—essentially a text-based version of the process schedules shown in example problems.
- The total number of context switches

For example, using STCF scheduling with the sample input file described above would yield the following summary (this information comes from slides 11 & 12 of Lecture 13, with the processes numbered slightly differently):

```
STCF Summary (WT = wait time, TT = turnaround time):

PID    WT     TT
 0     16     26
 1      0      3
 2      2      9
 3      0      1
 4      5     10
AVG   4.6    9.8

Process sequence: 2-3-4-3-5-1
Context switches: 6
```

Final summary: After the detailed output for each algorithm, your program should print a summary showing how all the algorithms perform based on the metrics measured in your simulation. The results for each metric should be listed in a table ordered from best to worst:

- Shortest average wait time
- Shortest average turnaround time
- Fewest context switches

UPDATE 4/13: Examples of the final summaries can be found in the test output files posted on the course website.

4. Grading Rubric

Your assignment will be graded according to the rubric below; partial credit may be given if you successfully complete part of a given objective. **You should not submit separate files for each objective**—your sole submission will be judged on how many of the tasks below it accomplishes successfully.

This program will be graded somewhat differently than earlier assignments. The rubric does not contain a series of progressively harder objectives; instead, your grade is based on which of the scheduling algorithms your program simulates correctly. Algorithms must include:

- First come, first served (FCFS)
- Shortest job first (SJF)
- Shortest time to completion first (STCF)
- Round robin (RR) with time quantum 2
- Non-preemptive priority scheduling

Each algorithm is worth **~~20 points~~** **30 points**, for a total of ~~100~~ **150** points. Note that a fundamental flaw in your simulator that affects all algorithms will result in a deduction for all sections of the program.

UPDATE 4/13: For this project, there will be two extra credit objectives. *If you have at least attempted either (or both) of these objectives (even if not successful), please include that information in your e-mail when submitting the project*:

- **10 points**: Implement aging in your priority scheme, such that the priority of a waiting process increases by 1 for every 25 cycles it must wait.
    - Note that "increasing priority by 1" actually means decrementing the priority value, as lower numbers represent higher priority.
- **10 points**: Assume that every process requires a 10 cycle I/O burst in the middle of the CPU burst and incorporate that additional time into your scheduling process.
    - Adding I/O bursts will require you to implement a separate I/O queue and track not just CPU burst time but I/O burst time as well.
    - This addition would effectively make every scheduling algorithm preemptive, as a process should be "preempted" after it has executed half of its CPU cycles (rounding down if the total CPU burst is odd) and added to the I/O device queue.
    - Assume the system has only a single I/O device and that device is scheduled on a first come, first served basis.
    - Once a process finishes its I/O burst, it should return to the ready queue to complete its CPU burst.

5. Hints

Tracking information: While this assignment does not require a specific implementation, you should, at a minimum, track the following information to generate the correct output:

- The current "time" as the simulator progresses
  - Time has no units in this program, but, for each algorithm, one iteration of your simulator loop should represent one time step
  - You will likely need to start a separate loop for each algorithm, since each algorithm's simulation is described in a separate section of your output file
- The currently running process
- The state of the ready queue
  - Remember, in preemptive algorithms, a process will return to the ready queue if it is forced to give up the processor before completing its CPU burst
- An ordered list of past processes that have run, so you can generate the final schedule to be shown in each algorithm summary
- All of the statistics described in the output

More hints to be added ...

6. Test Cases

Rather than listing detailed test cases in the assignment, I will post sample test files on the course web page. These files will come in pairs—an input file and the output generated based on that input file. The general form of different parts of your output—the snapshots, algorithm summaries, and final summaries—are described in Section 3.

<u>**UPDATE 4/13:**</u> I've now added two pairs of test input/output files to the website. The command lines used to generate these files were as follows (remember, the three command line arguments are input file name, output file name, and snapshot interval):

- `./prog3 testin1.dat out1.txt 1`
  (This test case is from the example process list given earlier, under "Input" in Section 3)
- `./prog3 testin2.dat out2.txt 10`