HCMC University Of Technology
Faculty of Computer Science and Engineering

– CO2018 Operating Systems – Fall 2018 –

Assignment 2
Scheduling Algorithms Simulator

October 19, 2018

Contents

1 Introduction

In this assignment, you will have to devise a simulator for various CPU scheduling algorithms. You will work in groups of 2-3 members. Each group will be assigned one general task and three group tasks. The group tasks will be assigned randomly to each group.

The simulator must be developed using C or C++. You are free to design your software as long as it meets the basic requirements of assigned tasks. Therefore, the design of the program should be unique to each group. Additionally, you are provided with a "dummy" scheduler which can be used as a reference for your design. An example of First Come First Served simulator is also provided along with this material.

After completing this assignment, you are expected to possess all of the followings:

- Perceive the scheduling principle in operating systems.
- Understand the trade-off involved in scheduling.
- Practice with code-based simulators.

1.1 Grading policy

The results of your work will be graded based on your BKeL submission and presentation.

- Simulator: **50 points** (20 points for the general task and 10 points for each group task). The general task must be completed in order to gain points from group tasks.
- Final report: **30 points**
- Presentation: **20 points**. The score of presentation will be graded differently for each member based on his or her performance.

All the tasks in your simulator are mainly graded based on your presentation. Therefore, you have to prove the correctness of your completed tasks when you present them. It is recommended that you should prepare some demonstrations for your simulator.

After this section, the rest of this document is organized in the following manners. Section 2 gives information the requirements your program. Input and output data are illustrated in Section 3. Section 4 presents a referenced framework of a simulator. Finally, Section 5 provides submission information.

2 Problem statement

2.1 General task

The general task is twofold: implement four primitive scheduling algorithms and evaluate the performance of each algorithm. The four mandatory scheduling algorithms include:

1. First Come First Served (FCFS)
2. Shortest Job First (SJF)
3. Shortest Remaining Time First(SRTF)
4. Round Robin (RR).

The performance of each algorithm is assessed based on 4 metrics below (refer to Lab 7 and the lecture for more information on theses metrics):

- Throughput
- Average waiting time
- Average response time
- Average turnaround time

Note: this task should be completed before proceeding with other tasks.

2.2 Group tasks

These group tasks will be assigned randomly to each group:

A. Investigate and implement two more algorithms other than what we have learned (e.g., Earliest deadline first, Highest Response Ratio Next, etc.).

B. Extend the scheduler routine to support the dynamic adding of new processes. The new processes can be sent to the scheduler deamon IPC (socket, shared memory, message queue).

C. Support dynamic queue mechanism for managing ready processes who are waiting to be allocated CPU resource. This mechanism supports adding/removing tasks to/from the queue.

D. Implement multi-level queues with at least 5 priority levels.

E. Create an analysis tool to compare and produce a report of simulation result among algorithms.

F. Develop a tool which generates random input data using statistical distributions and evaluates the correctness of a simulator.

2.3 Simulator configuration

Your project must be managed by Makefile or CMake. To compile the general task, these keywords are defined: **fcfs**, **sjf**, **srtf** and **rr**.

```
# This command create executable file named fcfs
# denote First-Come-First-Serve
$ make fcfs

# This command create executable file named sjf
# denote Shortest-Job-First
$ make sjf

# This command create executable file named srtf
# denote Shortest-Remain-Time-First
$ make srtf

# This command create executable file named rr
# denote Round-Robin
$ make rr

# This command remove all other files except source code file
$ make clean
```

3 Input and output data

The input and output format below are used for the general task. In regard to group tasks, you can extend the input and output according to your design.

3.1 Input data

The input of the simulation program is a text file described by the option `-i filepath`. If the parameter is omitted then the default value is `input.txt`. Input file's content is a list of processes, each of them is represented by an input entry.

There are N lines ($0 < N < 1000$) in the input file. Each line illustrates a process' attributes including arrival time and CPU burst time (time required for completing job). Each attribute is separated by a blank space. The process indexes are assigned based on the order of input data. For example:

```
0 7
2 4
4 1
5 4
```

In this example, the processes are numbered from 0 to 3. An instance of the command to run the simulation with RR is as follows:

```
$ ./rr -i /home/tc/fcfs_input1.txt
```

3.2 Output data

The output of the simulation program is a text file indicated by the option `-o filepath`. If the parameter is omitted then the default value is `output.txt`. An example of running a simulation with explicit declaration of input and output data files:

```
$ ./rr -i /home/tc/fcfs_input.txt -o /home/tc/fcfs_output.txt
```

The content of the output file must indicate the time slot of each process as follows:

```
Process  0 start     0 end     2
Process  0 start    11 end    16
Process  1 start     2 end     4
Process  1 start     5 end     7
Process  2 start     4 end     5
Process  3 start     7 end    11
```

The system time of your simulator starts from 0. Furthermore, your program must print out the value of each performance metric mentioned above.

4 Scheduling Simulator Framework

This is a framework of a simulator, you can use it as a reference for your design. This framework is implemented in `sched_dummy.c`

4.1 Process-related data structures

Processes are simulated using a set of integer attributes as in Listing 1. The timestamp is represented by an integer starting from 0 and is incremented over time units. The index of the process is assigned according to the order of the process in the input file (index starts from 0).

Listing 1: Process definition

```
struct timeslot
{
   int starttime;   /* Timestamp at the start of the execution */
   int endtime;     /* Timestamp at the end of the execution */
}

struct process
{
   /* Values initialized for each process */
   int arrivaltime;  /* Timestamp at which process arrives and
                        wishes to start */
```

```
    int bursttime;     /* Amount of time process
                          requires to complete the job */

    /* Values used to track process */
    struct timeslot* assigned_timeslot;
    int timeslot_count;
    int flag;
}
```

4.2 Simulation Clock and a Time-advancing Mechanism

A simulator has a global variable representing simulated time. There are two methods for this mechanism. The first way, called an **unit time** approach, consecutively increments the system clock and checks to see if there are any incoming events. The second approach, called an **event-driven** approach, increase the system clock according to the time of the next earliest occurring event. The unit time approach is not widely used in computer simulations [1].

5 Submission

5.1 Deliverables

After you have finished the assignment, you need to compress the followings into **assignment2_studentID.zip**:

- source-code/: This folder contains all of your source code and makefiles
- report.pdf (Your report in PDF format)
- Presentation file(s)

Requirement: you have to develop your program with a proper coding style. Reference: `https://www.gnu.org/prep/standards/html_node/Writing-C.html`.

5.2 Report

Your report should comprise the following content:

- Introduction: Make a case for your work. Explain to the readers what problems did you solve.
- Methodology: What did you do? Elaborate on what you did to solve the problems/tasks. This is the most important part of your report. In developing this section, be sure to describe your design in detail.
- Results: What did you accomplish? Present the results of your work
- Discussion: What does it mean? Discuss the meaning of your results, do not leave the readers asking "so what?"

- Appendix: add an appendix explaining how to use your program.

FYI: This is called as an IMRAD (Information, Methods, Results and Discussion) structure/format, you can lookup this term online.

Your report MUST be from 4 to 8 pages (font size 10-11pt), do NOT include a title page. The score of your report will be based on its structure and how well your present your work.

5.3 Deadline and Presentation

Your presentation will be on **November 9th**. On your presentation day, your simulator and presentation will be graded, therefore, complete your simulator by that day. A detail schedule regarding the specific presentation day for each group will be announced later.

The whole package (source code, report, presentation) must be submitted to BKeL by **November 16th, 2018**. This deadline and the designated presentation day are fixed and will NOT be extended.

References

[1] Raj Jain. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons, 1990.

A FAQs

1. *> Explain task C in detail*

Implement a ready queue in your program. This ready queue enables adding/removing tasks while running, you will have to deal with **race condition**. If you are using C++, you should create a class that has methods for adding and removing tasks from the queue, the same applies for struct in C. The scheduler will use those methods to manage processes in the ready queue. For example, in case of RR, your program invokes removing a task from the queue and allocate CPU for that process, after that, your program invokes adding that process back into the ready queue when its time slice is over.

Furthermore, the ready queue could be used by multiple threads, so it should handle race condition when adding/removing tasks.