COMP 3500 Introduction to Operating Systems
Project 5 - CPU Scheduling

Short Version: 1.1
11/5/2019

This is an individual assignment; no collaboration among students.

1. Learning Objectives

You will achieve the following objectives upon the completion of the project.

- To design a simple CPU scheduler
- To implement three scheduling policies in C/C++
- To conduct scheduling simulation experiments on a Linux machine
- Use GDB to debug your C/C++ program

2. Requirements

- Each student should independently accomplish this project assignment. You may discuss with other students to solve the coding problems. Students should not share any project code with any other student. Collaborations among students in any form will be treated as a serious violation of the University's academic integrity code.
- Important! You must follow the specified output format.
- You must write a C/C++ program on the Linux operating system.

3. Implementing a CPU Scheduler

3.1 Scheduling Policies

Your simulator should manage the following three scheduling policies.

- First Come First Serve (FCFS),
- Round Robin or RR, and
- Shortest Remaining Time First or SRTF).

3.2 Task Information and an Input File

The task information will be read from an input text file, the format of which is

```
pid arrival_time burst_time
```

- `pid` is a unique numeric process ID
- `arrival_time` is the time instance at which a task arrives
- `burst_time` is the CPU time requested by a task

The time unit of `arrival_time`, `burst_time`, and `interval` is millisecond.

3.3 Command-line Usage

Users should run your scheduler in a command-line as:

```
$scheduler task_list_file [FCFS|RR|SRTF] [time_quantum]
```

where `input_file` is the file name with task information described in Section 3.2 on page 1. `FCFS`, `RR`, and `SRTF` are names of the scheduling policies (a.k.a., algorithms). Parameter `time_quantum` is only applicable to the `RR` policy.

Sample Usage	Description
`scheduler list1 FCFC`	Simulate the FCFS policy with a task file named "input1"
`scheduler list2 RR 5`	Simulate RR with the "input2" task file; time quantum is set to 5.
`scheduler list3 SRTF`	Simulate the SRTF policy with a task file named "input3"

3.4 System Design

Data Flow Diagram. You are suggested to start the design of your scheduler by crafting a data flow diagram.

Loading Input File. After the simulator reads a task information list from an input text file, all the task information should be stored in a task list (e.g., an array, a dynamic array, or a linked list). Then, the simulator schedules tasks according to a scheduling policy specified in the command-line.

Time-Driven Simulation. Your simulator manages tasks in a time-driven manner. More specifically, at each time unit (a.k.a., time slot), your simulator inserts any newly arrived task into the ready queue, followed by invoking the scheduler to choose an appropriate task from the ready queue. When a task is elected to run, the simulator prints out a message indicating the process ID of a task running in this time slot. If no task is running (i.e., the ready queue is empty) during the current time unit, the simulator prints out an "idle" message. Before advancing to the next time unit, the simulator should update all necessary changes in the running task and the ready queue status.

3.5 Output and Statistical Information

The scheduler is expected to compute the following statistical information referred to performance metrics. You will learn these metrics in our lectures. Please refer to Section 6 for a sample output.

- average waiting time
- average response time
- average turnaround time
- overall CPU usage

3.6 Suggested Functions

You are recommended to implement the following functions.

- **A command-line parser.**
- **Three scheduling algorithms.**
- **Read an input file.**
- **Print statistic information.**
- **Error handler.**

4. Programming Requirements

4.1 Programming Environment

Write your simulated virtual memory system in either C or C++. Compile and run your system using the gcc or g++ compiler on a Linux box.

4.2 Function-Oriented Approach

You are *strongly suggested* to use a structure-oriented approach for this project.

4.3 File Names

You will lose points if you do not use the specific program file name, or do not have a comment block on every source code file you hand in.

4.4 Usability Concerns and Error-Checking

You should appropriately prompt your user and assume that they only have basic knowledge of the system. You should provide enough error-checking that a moderately informed user will not crash your system.

5. Separate Compilation

You must use separate compilation and create a makefile for this project. Check out the following links for more information on makefiles:

https://www.gnu.org/software/make/manual/html_node/Introduction.html

6. Sample Input and Output

Suppose an input task file is called "task.list", your scheduler should run as follows to simulate the FCFS scheduling policy.

```
%more task.list
1 0 10
2 0 9
3 3 5
4 7 4
5 10 6
6 10 7

%scheduler
Usage: scheduler task_list_file [FCFS|RR|SRTF] [time_quantum]

%scheduler task.list FCFS
Scheduling Policy: FCFS
There are 6 tasks loaded from "task.list". Press any key to continue ...
=================================================================
<time 0> process 1 is running
<time 1> process 1 is running
<time 2> process 1 is running
<time 3> process 1 is running
<time 4> process 1 is running
<time 5> process 1 is running
<time 6> process 1 is running
<time 7> process 1 is running
<time 8> process 1 is running
<time 9> process 1 is running
<time 10> process 1 is finished...
<time 10> process 2 is running
<time 11> process 2 is running
<time 12> process 2 is running
<time 13> process 2 is running
```

```
<time 14> process 2 is running
<time 15> process 2 is running
<time 16> process 2 is running
<time 17> process 2 is running
<time 18> process 2 is running
<time 19> process 2 is finished...
<time 19> process 3 is running
<time 20> process 3 is running
<time 21> process 3 is running
<time 22> process 3 is running
<time 23> process 3 is running
<time 24> process 3 is finished...
<time 24> process 4 is running
<time 25> process 4 is running
<time 26> process 4 is running
<time 27> process 4 is running
<time 28> process 4 is finished...
<time 28> process 5 is running
<time 29> process 5 is running
<time 30> process 5 is running
<time 31> process 5 is running
<time 32> process 5 is running
<time 33> process 5 is running
<time 34> process 5 is finished...
<time 34> process 6 is running
<time 35> process 6 is running
<time 36> process 6 is running
<time 37> process 6 is running
<time 38> process 6 is running
<time 39> process 6 is running
<time 40> process 6 is running
<time 41> process 6 is finished...
<time 41> All processes finish ......
============================================================
Average waiting time:    14.17
Average response time:   14.17
Average turnaround time: 21.00
Overall CPU usage:       100.00%
============================================================
```

7. Project Report

Important! Your project report should provide sample input and output from your scheduler.

7.1 (6 points) Solution Description

You must answer the following questions.

(1) How did you separate scheduling mechanism from scheduling policies?
(2) How did implement the three scheduling algorithms?
(3) How did you calculate waiting times?
(4) How did you calculate response times?
(5) How did you calculate turnaround times?
(6) How did you implement the command-line parser?

7.2 (6 points) Generality and Error Checking

(1) How general is your solution?
(2) How easy would it be to add a new scheduling policy into your scheduler?
(3) Does your program offer input error checking?

7.3 (3 points) Miscellaneous factors

(1) Is your code elegant?
(2) How innovative is your solution? Did you try any ideas not suggested here?
(3) Did you document all outside sources?

8. Deliverables

8.1 Final Submission

Your final submission should include:

- A project report (see also Section 7)
- A copy of the complete source code of your CPU scheduler
- Makefile (see also Section 5 on page 3)

Important! You must submit a single compressed file (see Section 8.2) as a .tgz file, which includes both a project report, source code, and Makefile.

8.2 A Single Compressed File: `proejct5.tgz`

8.3 What happens if you can't complete the project?

If you are unable to complete this project for some reasons, please describe in your final design document the work that remains to be finished. It is important to present an honest assessment of any incomplete components.

9. Grading Criteria

The approximate marks allocation will be:

1) Project Report: 15%
2) Implementation (i.e., Source Code): 60%
3) Makefile: 15%
4) Clarity and attention to details: 10%