Group Project 4: CPU scheduler

CECS 326 – Operating Systems

1. Summary

This project follows the topic of CPU scheduling, where it requires to design and implement several classic CPU scheduling algorithms.

You should submit the required deliverable materials on BeachBoard by **11:55pm, December 06th (Sunday), 2020.**

2. Description

This project involves implementing several different process scheduling algorithms. The scheduler will be assigned a predefined set of tasks and will schedule the tasks based on the selected scheduling algorithm. Each task is assigned a priority and CPU burst. The following scheduling algorithms will be implemented:

- **First-come, first-served** (FCFS), which schedules tasks in the order in which they request the CPU.
- **Priority scheduling**, which schedules tasks based on priority.
- **Round-robin** (RR) scheduling, where each task is run for a time quantum (or for the remainder of its CPU burst).

Priorities range from 1 to 10, where a higher numeric value indicates a higher relative priority. For round-robin scheduling, the length of a time quantum is 10 milliseconds.

You need to complete the three scheduling algorithms in the attached sample code (***schedule_fcfs.c, schedule_priority.c, schedule_rr.c***).

3. Implementation

The implementation of this project should be completed in C, and supported program files are provided in the sample code. These supporting files read in the schedule of tasks, insert the tasks into a list, and invoke the scheduler.

The schedule of tasks has the form **[task name] [priority] [CPU burst]**, with the following example format:

T1, 4, 20

T2, 2, 25

T3, 3, 25

T4, 3, 15

T5, 10, 10

Thus, task T1 has priority 4 and a CPU burst of 20 milliseconds, and so forth. It is assumed that all tasks arrive at the same time, so your scheduler algorithms do not have to support higher-priority processes preempting processes with lower priorities. In addition, tasks do not have to be placed into a queue or list in any particular order.

There are a few different strategies for organizing the list of tasks. One approach is to place all tasks in a single unordered list, where the strategy for task selection depends on the scheduling algorithm. Alternatively, a list could be ordered according to scheduling criteria (that is, by priority). One other strategy involves having a separate queue for each unique priority. It is also worth highlighting that we are using the terms list and queue somewhat interchangeably. However, a queue has very specific FIFO functionality, whereas a list does not have such strict insertion and deletion requirements. You are likely to find the functionality of a general list to be more suitable when completing this project.

4. C Implementation Details

The file ***driver.c*** reads in the schedule of tasks, inserts each task into a linked list, and invokes the process scheduler by calling the ***schedule()*** function. The ***schedule()*** function executes each task according to the specified scheduling algorithm. Tasks selected for execution on the CPU are determined by the ***pickNextTask()*** function and are executed by invoking the ***run()*** function defined in the ***CPU.c*** file. A ***Makefile*** is used to determine the specific scheduling algorithm that will be invoked by driver. For example, to build the FCFS scheduler, we would enter

```
make fcfs
```

and would execute the scheduler (using the schedule of tasks schedule.txt) as follows:

```
./fcfs schedule.txt
```

Refer to the README file in the source code download for further details. Before proceeding, be sure to familiarize yourself with the source code provided as well as the Makefile.

3: The Required Deliverable Materials

(1) Your source code, must be submitted in the required format.
(2) Your report, which discusses the design of your program. Report should display the outputs of different scheduling algorithms from your code.

3. Submission Requirements

You need to strictly follow the instructions listed below:

1) This is a **group project**, please submit a .zip/.rar file that contains all files, only one submission from one group.

2) The submission should include your **source code** and **project report**. Do not submit your binary code. Project report should contain your groupmates name and ID.

3) Your code must **be able to compile**; otherwise, you will receive a grade of zero.

4) Your code should not produce anything else other than the required information in the output file.

5) If you code is **partially completed**, please explain the details in the report what has been completed and the status of the missing parts, we will grade it based on the entire performance.

6) Provide **sufficient comments** in your code to help the TA understand your code. This is important for you to get at least partial credit in case your submitted code does not work properly.

Grading criteria:

Details	Points
Submission follows the right formats	5 pts
Have a README file shows how to compile and test your submission	5 pts
Submitted code has proper comments to show the design	5 pts
Have a **report** (pdf or word) file explains the details of your entire design	25 pts
Report contains clearly individual contributions of your group mates	10 pts
Code can be compiled and shows correct outputs	50 pts

4. Policies

1) Late submissions will be graded based on our policy discussed in the course syllabus.
2) Code-level discussion is **prohibited**. We will use anti-plagiarism tools to detect violations of this policy.