Project 1 - A Job Scheduler

CO004 Projects on Operating Systems

April 18, 2018

Due Date for Grouping: **April 10, 2018**
Due Date for Phase 1: **April 24, 2018**
Due Date for Phase 2: **May 10, 2018**

1 Objective

- Understanding how a job scheduler works.
- Increasing experience with system calls.
- Learning UNIX signals and controlling methods of signals

2 Introduction

This project will help you understand some fundamental mechanisms of process scheduling in UNIX/LINUX environments. In this project, you are required to implement a basic job scheduler, which supports two scheduling algorithms, i.e., FCFS (First Come and First Serve) and RR (Round Robin).

This is a group project. You can form a group with **at most 3 members**. You shall learn how to solve a middle-level problem with collaboration with your partners and learn how to manage your time under a tight schedule.

3 Introduction

Process control and management is an important topic to modern operating systems. Therefore, to implement a basic job scheduler can help you having a better understanding on process control and management as well as having improved program skills.

Project 1 involves with process creation, suspension, termination and signal handling. You are required to implement a basic but working job scheduler. It is not an easy task. Therefore, you must cooperate closely with your partner. Besides, you also need to manage your time in a tight schedule.

4 Programming environment

Your system shall be able to be executable under the following environments:

- *Operating Systems.* You must write and test you programs under the Linux operating systems (**MS Windows are not acceptable**). The preferred Linux OSes include CentOS, Ubuntu, Debian, etc. The final testing Linux platform will be released later.
- *Language.* You must implement it in either C or C++. Using languages other than the above two will is also not acceptable.
- You are not allowed to invoke the `system(3)` library call. Otherwise, you would score 0 marks for this assignment.

Note that your development can be done in MacOS or other Linux/Unix platforms though the final program shall be tested in our given Linux.

5 Assignment

You should write a simple job scheduler. The scheduler should read jobs from a job description file and process it and schedule the jobs specified in this description file according to different scheduling algorithms. There are some basic functionalities you need to implement: (i) processing the jobs defined in the job file; (ii) building up the job scheduling queue; (iii) scheduling the jobs according to FCFS; (iv) scheduling the jobs according to RR.

6 Specification

6.1 Basic functions

As shown in Figure 1, you are required to implement a process scheduler, which has the following functions:

1. Read the jobs information from a job description file named "`Job.txt`".
2. The scheduler creates the jobs from such job description file.
3. The jobs are scheduled to be executed according to the *scheduling policy* (such as FCFO and RR) selected *when the scheduler program starts*.
4. The scheduler should not be terminated until all the jobs are scheduled and are terminated.
5. When each job is scheduled, your scheduler should output the exact execution of the processes to the screen (i.e., your scheduler MUST TRULY execute the process).
6. After all the jobs are terminated (but before the scheduler is terminated), the scheduler should print out the job execution information to a file named "JobOut.txt", which includes the scheduling sequences of all the jobs (For more details, please refer to Section 6.3).

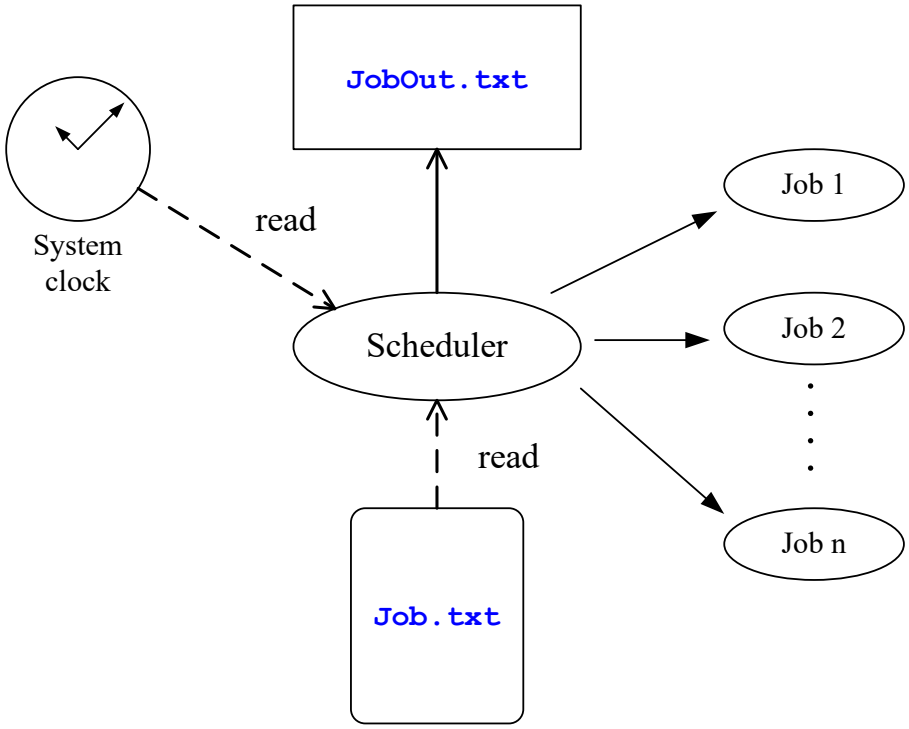

Figure 1: System architecture

6.2 The Scheduler

For one execution of the scheduler, it must follow one scheduling policy only. To invoke the scheduler, the following arguments are required:

```
$ ./scheduler [input filename] [policy]
```

Note:

- $ is the shell prompt;
- `./scheduler` is the path to the scheduler program;
- `[input filename]` is the path to the job description file;
- `[policy]` is the scheduling policy which should take the following values:
  1. `FCFS` (which means the first come first serve policy);
  2. `RR 2` (which means the round robin policy) and "2" means the time quota is "2". (Note: Do not type the quotes.)

After invoking the scheduler, it creates a set of jobs that are described in âĂIJJob.txtâĂİ, which will be described in Section 6.3. Then, the jobs will be scheduled to execute according to the given policy. After the jobs are terminated, the scheduler should print out the statistic information into a file "`JobOut.txt`".

In addition, it has more things to do and we list the properties and the functionalities of the scheduler process as follows.

Job #1	`Arrival Time`	\t	`Command`	\t	`Duration`	\n
Job #2	`Arrival Time`	\t	`Command`	\t	`Duration`	\n
Job #3	.....					

Figure 2: The format of the job description file.

- The scheduler is an ordinary process, i.e., no special privilege should be given to the scheduler.
- The scheduler creates a set of jobs that are described in the job description file. The job description file contains a list of jobs that the scheduler should execute. The jobs may not be scheduled in a particular order, but according to the scheduling policy selected when the program starts.
- While scheduling, the scheduler reads from the real-time clock of the system to determine the time in scheduling.
- The scheduler should not be terminated until all the jobs are scheduled and are terminated.

6.3 Job description file

This is a plain-text file that stores a list of job in the format shown in Figure 2. We describe the format shown as follows.

- There are **at most nine jobs** stated in a job description file.
- One line represents one job, i.e., every job description is separated by one newline character.
- The lines of job descriptions are sorted by the "`Arrival Time`" field, in ascending order.
- A line of job description must contain three non-empty fields: "`Arrival Time`", "`Command`" and "`Duration`". Every two fields are separated by exactly one tab character.
- The "`Arrival Time`" of a job is measured relative to the time that the scheduler starts executing and is measured in terms of seconds. The smallest value is 0 and the largest is the maximum value defined as "`INT MAX`" in the header file "`limits.h`".
  - When the scheduler has just been started, the time that is relative to the time that the scheduler starts executing is zero.
  - If "`Arrival Time`" of a job is i, it means the job should start executing i seconds after the scheduler starts.
- The "`Command`" is a command string with the following constraints:
  - A command has a maximum length of 255 characters.
  - There will be no leading or trailing space characters in any commands.

– A command is composed of a series of **tokens** (or words). The first token is the program name and the successive tokens are the program arguments. Every two tokens are separated by **exactly one space character**.

Note that the first token may be the name of the program or the path to the program.

- The "`Duration`" is the number of seconds that the command is required to run. The possible values of the "`Duration`" can be either -1 or all non-zero positive integers. The value -1 means that the command is allowed to run indefinitely until it terminates. It is worth mentioning that the "`Duration`" is not the **accumulated CPU time of the process**, but the time elapsed after the job has been started. Although some of you may feel that it is a lousy implementation, this consideration can greatly reduce your workload.

The following shows an example job description file:

```
1  ls -lR / 10
2  ./while1 1
3  ls -R /home -1
```

According to the above example, when the first job is running, the second job arrives at the system. The time that the second job starts depends on the scheduling policy taken by the scheduler. Note that the jobs stated in the file are sorted according to the "`Arrival Time`" field in ascending order.

6.4 System clock

Note hat the current time value is maintained by hardware. For simplicity of system implementation, we restrict the granularity of the time to be in terms of seconds. Although it is also possible for us to read the clock in the scale of microseconds (10^{-6}), this may make the design complicated. In practice, this simplified consideration may scarify the precision of the scheduling algorithms.

6.5 Assumptions

To simplify the design of the scheduler, we the following assumptions:

- The scheduler is assumed to execute one job at a time. In other words, we assume that there is only one CPU with one core only.
- The scheduler itself will not be interrupted, meaning that it will not be suspended nor be killed while it is running.
- Every job will not create **pthreads** (e.g., using the `pthreadcreate()` call) or other processes.
- Every job is assumed to be executed successfully, meaning that the scheduler process will not meet the cases that the permission is denied nor the program cannot be found.

6.5.1 Reading the job description file

To read the job description file, the following library function calls may be useful to you: "`fgets()`" and "`strtok()`". **Assumption.** You can assume that the input file always has the correct format.

Hint: Queue data structure

While the file is read, data structures must be built in order to store the data read from the job description file. You are recommended to built a queue data structure in order that you can easily manage the jobs. E.g., under the SJF policy, the "*Duration field*" plays one of the major roles in ordering the queue.

6.6 Finish scheduling

When the scheduler has finished scheduling all the jobs, it should print a **scheduling report** named as "`JobOut.txt`".

Given a job description file:

```
0 \t Command \t 6 \n
2 \t Command \t 6 \n
```

The following is the example of the scheduling report:

```
0 sec: Job 1 run
3 sec: Job 1 suspended
3 sec: Job 2 run
6 sec: Job 2 suspended
6 sec: Job 1 continued
9 sec: Job 1 terminated
9 sec: Job 2 continued
12 sec: Job 2 terminated
```

6.7 Reports

Submit a report including:

- A design report including the system architecture, the flowcharts and descriptions of each module.
- A short manual about your program including how to use your program.
- All the source files (e.g., *.c or *.cpp and *.h) and Makefile (if any).
- The executable files are **NOT necessary** to be submitted.

7 Milestones

Make sure that you can understand all the assumptions and regulations of the system in Section 6.

7.1 Phase 1: Completion of the job interpreter

The following is the list of tasks that you have to complete.

- Read job description file;
- Tokenize each job;
- Output each job with its arguments

7.2 Phase 2: Completion of the entire project

The following is the list of tasks that you have to complete.

- The FIFO scheduler;
- The RR scheduler;
- No zombie process is left in the system.
- Written Reports on your system according to Section 6.7.

8 Grading Policy

- Phase 1 (30%): Demonstration
- Phase 2 (50%): Demonstration
- Report (20%)

Late demonstration of each phase will lead to the score penalty.

9 Submission

Please submit your reports and program codes through http://moodle.must.edu.mo/.