FIT2100 Assignment #2 (Part A)
Processor Scheduling with C Programming
Week 9 Semester 2 2017

Dr Jojo Wong
Lecturer, Faculty of IT.
Email: Jojo.Wong@monash.edu

September 4, 2017

Revision Status

$Id: FIT2100-Assignment-02.tex, Version 1.0 2017/09/03 17:30 Jojo $

Contents

1 Introduction

This assignment is due on **22nd September 2017 (Friday) by 5:00pm**. It is worth **10% of the total unit marks**. A penalty of 5% per day will be applied for late submission. Refer to the FIT2100 Unit Guide for the policy on extension or special consideration.

Note that this is **an individual assignment** and **must be your own work**. Please pay attention to Section 3.2 of this document on the University's policies on the *Academic Integrity, Plagiarism and Collusion*.

This assignment consists of **three implementation tasks**. Each task can be implemented as a separate C program or an individual C function to be invoked within the main function of a C program. Any observation on the results of the three tasks should be discussed and documented. All the program files and any supporting documents should be compressed into one single file for submission. (The submission details are given in Section 3.)

Assessment: Each of the three tasks has an equal weightage of marks with 30% **each and the discussion is worth** 10%**.**

2 Processor Scheduling

Processor scheduling is aimed to assign processes to be executed by the processor (or processors) in order to achieve certain operating system objectives, such as response time, turnaround time, throughput, and processor efficiency. Scheduling generally affects the overall performance of a system as it determines which process should wait and which process should proceed for execution. Hence, scheduling algorithms are designed to manage queues of processes with the purpose to minimise the queuing delay and to optimise the system performance.

The aim of this assignment is to implement and simulate the performance of three different scheduling algorithms and to observe the effects of each of theses scheduling algorithms on the overall system performance. (For simplicity, we are assuming that these scheduling algorithms are only applied on a uniprocessor.)

All the programs should be implemented on the Unix/Linux operating system with the C programming language.

Running your program with an input file

For each program implemented for the tasks below, it should take an input file as a command line input. You may name the input file as `processes.txt`, for example.

Note that the input file should have the following structure: the first column contains the process ID; the second column indicates the arrival time (in seconds) of each process in the Ready queue; and the last column refers to the total processing time (in seconds) required for the corresponding process.

Assume that these processes are batch jobs and are processor-bound only (i.e. no I/O operations are involved).

Process	Arrival Time	Processing Time
P1	0	3
P2	1	6
P3	4	4
P4	6	2

2.1 Task 1: First-Come-First-Served Scheduling

In the first task, you will implement the simplest scheduling algorithm that based on the *first-come-first-served* (FCFS) approach. With this strict queuing scheme, the process that first arrives in the Ready queue is the first to be allocated with the processor for execution. In other words, when the currently running process completes its execution, the next process to be selected for execution is the one that has been in the Ready queue for the longest time. This is a *non-preemptive* algorithm.

Your implementation for the FCFS scheduling should compute and output the following:

- **Turnaround time** and **waiting time** for each process;
- **Average turnaround time** and **average waiting time** for all the processes;
- Overall **throughput** of the system (i.e. the number of processes completed per second).

2.2 Task 2: Round Robin Scheduling

For the second task, you will implement a *preemptive* scheduling algorithm that based on the *Round Robin* (RR) approach. Processes in the Ready queue are still scheduled on the first-come-first-served basis; however, each process is assigned with a fixed *time quantum* for execution. Upon the given time quantum elapsed, the currently running process is preempted by the scheduler and placed to the end of the Ready queue. The next process in the Ready queue will be selected for execution; and this continues until all the jobs have completed their execution. (For the purpose of this assignment, assume that the time quantum is set to 2 seconds.)

As in Task 1 (2.1), your implementation for the RR scheduling should compute and output the following:

- **Turnaround time** and **waiting time** for each process;
- **Average turnaround time** and **average waiting time** for all the processes;
- Overall **throughput** of the system (i.e. the number of processes completed per second).

2.3 Task 3: Shortest Remaining Time Scheduling

The third scheduling algorithm that you will implement is based on the Shortest Remaining Time (SRT) concept. With this algorithm, the process with the shortest expected remaining processing time is always chosen by the scheduler. For each new process that arrives in the Ready queue, it may have a shorter remaining time than the currently running process. As such, the scheduler may preempt the currently running process and promote the newly arrived process for execution. (If two processes shared the same shortest remaining processing time, the FCFS approach is used for breaking the tie.)

Your implementation for the SRT scheduling should again compute and output the following:

- **Turnaround time** and **waiting time** for each process;
- **Average turnaround time** and **average waiting time** for all the processes;
- Overall **throughput** of the system (i.e. the number of processes completed per second).

2.4 Task 4: Discussion

For this final task, based on the results presented by each of the scheduling algorithms, discuss what you may have observed on the overall system performance, including the following:

- Which particular algorithm results in the shortest waiting time or the shortest turnaround time on average, or with a better system throughput overall?
- Which algorithm is in favour of either shorter processes or longer processes?

The discussion for each scheduling algorithm should be around 300 words.

2.5 Important Notes

Commenting your code is essential as part of the assessment criteria (refer to Section 2.6). You should also include comments at the beginning of your program file, which specify your name, your Student ID, the start date and the last modified date of the program, as well as with a high-level description of the program. In-line comments within the program are also part of the required documentation.

2.6 Marking Criteria

The assessment of this assignment will be based on the following marking criteria. The same marking criteria will be applied on each implementation task (Task 1 to Task 3):

- 60% for working program;
- 10% for code architecture (algorithms, use of procedures and libraries, etc.);
- 10% for coding style (clarity in variable names, function names, etc.);
- 20% for documentation (both program comments and user documentation).

3 Submission

There will be NO hard copy submission required for this assignment. You are required to submit your assignment as a `.zip` file named with your Student ID. For example, if your Student ID is `12345678`, you would submit a zipped file named `12345678_A2.zip`. Note that marks will be deducted if this requirement is not strictly complied with.

Your submission is via the assignment submission link on the FIT2100 Moodle site by the deadline specified in Section 1, i.e. **22nd September 2017 (Friday) by 5:00pm**.

3.1 Deliverables

Your submission should contain the following documents:

- A completed the assignment submission statement for online submission via the FIT2100 Moodle site.
- An user documentation (not more than 3 pages) in PDF format with clear and complete instructions on how to run your programs. (Note that your programs must at least run on the Unix/Linux system in the University's computer labs. Any submission that does not compile or run accordingly will receive no marks.)
- Electronic copies of ALL your files that are needed to run your programs.

Marks will deducted for any of these requirements that are not strictly complied with.

3.2 Academic Integrity: Plagiarism and Collusion

Plagiarism means to take and use another person's ideas and or manner of expressing them and to pass them off as your own by failing to give appropriate acknowledgement. This includes materials sourced from the Internet, staff, other students, and from published and unpublished works.

Collusion means unauthorised collaboration on assessable work (written, oral, or practical) with other people. This occurs when you present group work as your own or as the work of another person. Collusion may be with another Monash student or with people or students external to the University. This applies to work assessed by Monash or another university.

It is your responsibility to make yourself familiar with the University's policies and procedures in the event of suspected breaches of academic integrity. (Note: Students will be asked to attend an interview should such a situation is detected.)

The University's policies are available at: `http://www.monash.edu/students/academic/policies/academic-integrity`