FIT2070 Operating Systems

Assignment Three
Semester 2, 2016

Introduction

This assignment is due by **11:55PM** on **Sunday 23rd October 2016**. It is worth 10% of the marks for your final assessment in this unit. A penalty of 5% per day will apply for late submission. Refer to the FIT2070 Unit Guide for the policy on extensions or special case considerations.

Note that **this is an individual assignment and must be your own work**. Please note the section on *Academic Integrity, Plagiarism, and Collusion* at the end this document.

This third assignment consists of three implementation tasks. Each task can be implemented as a separate program or an individual function to be invoked within a main program. Any observation on the results of the three tasks should be discussed and documented. All the program files and documentation should to be zipped while submitting on Moodle.

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

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 queing 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 these scheduling algorithms on the overall system performance.

All the programs should be implemented on the Unix operating system with either C or Python 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 all these processes are processor-bound only (i.e. no I/O operations are involved).

```
P1    0    3
P2    1    6
P3    4    4
P4    6    2
```

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. 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)

Task 2: Round Robin Scheduling

For the second task, you will implement a *pre-emptive* 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 jobs complete its execution. (For the purpose of this assignment, assume that the time quantum is set to **2** seconds.)

As in Task 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)

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)

Task 4: Discussion

For this final task, based on the results presented by each scheduling algorithm, discuss what you may observe on the overall system performance. For example, which algorithm gives a shortest waiting time or a shortest turnaround time on average, or with a better system throughput?

Other discussion may include which algorithm is in favour of either the shorter processes or the longer processes.

Important Notes

Commenting your code is essential as part of the assessment criteria. 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 a high-level description of the program. In-line comments within the program are also part of the required documentation.

Submission

There is 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 were **12345678** then you would submit a file named **12345678_A3.zip**. Marks will be deducted if this requirement is not strictly complied with.

Your submission is via the assignment submission link on the FIT2070 Moodle website by the deadline specified in this document, i.e. **11:55PM** on **Sunday 23rd October 2016**.

Deliverables

Your submission should contain the following:

- A completed Assignment Cover Sheet. This document is available for download from the Assignment Section of the FIT2070 Moodle website.
- A documentation in MS Word or PDF format, stating:
  1. Clear and complete instructions on how to install and run your programs.
  2. Discussion on the system performance with different scheduling algorithms (i.e. Task 4).
- Electronic copies of ALL your files that are needed to run your programs.

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

Note that your programs must run in the University's student labs in which the practical classes for FIT2070 are held in this semester. Any submission that does not run accordingly will receive no marks.

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% Working of the program;
- 20% Code architecture (algorithms, use of procedures and libraries, etc.);
- 10% Coding style (clarity, variable names, etc.);
- 10% Documentation (both program and user documentation).

Academic Integrity, Plagiarism, and Collusion

Students should consult the University’s policies on this matter at:
http://www.monash.edu.au/students/policies/academic-integrity.html

The following excerpts are from the aforementioned URL:

Plagiarism
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 acknowledgment. This includes material sourced from the Internet, staff, other students, and from published and unpublished works.

Collusion
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.