FIT2100 Tutorial #4
Uniprocessor Scheduling
(Suggested Solutions)
Week 7 Semester 2 2017

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

August 31, 2017

Revision Status:

```
$Id: FIT2100-Tutorial-04.tex, Version 1.0 2017/08/31 18:30 Jojo $
```

Acknowledgement

The majority of the content presented in this tutorial was adapted from:

- William Stallings (2015). *Operating Systems: Internals and Design Principles (8th Edition)*, Pearson.

Contents

1 Background

This tutorial provides students with the opportunity to explore further on the concepts of uniprocessor scheduling discussed in the lecture. (This tutorial is also aimed to provide students with the types of questions to be expected in the final examination.)

You should complete the suggested reading in Section 2 before attending the tutorial. You should also prepare the solutions for the two sets of practice tasks given in Section 3.1 and Section 3.2 respectively.

2 Pre-tutorial Reading

You should complete the following two sets of reading:

- Week 6 Lecture Notes on "Uniprocessor Scheduling"
- Chapter 9 from the Stalling's textbook (7th/8th Edition)

3 Practice Tasks

3.1 Review Questions

Question 1

Three types of processor scheduling:

(a) **Long-term scheduling**: The decision to add to the pool of processes to be executed.

(b) **Medium-term scheduling**: The decision to add to the number of processes that are partially or fully in the main memory.

(c) **Short-term scheduling**: The decision as to which available process will be executed by the processor.

Question 2

Turnaround time is the total time that a request spends in the system (**waiting time** plus **service time**). **Response time** is the elapsed time between the submission of a request until the response begins to appear as output.

Question 3

Preemptive: The currently running process may be interrupted and moved to the READY state by the operating system. The decision to *preempt* may be performed when a new process arrives, when an interrupt occurs that places a BLOCKED process in the READY state, or periodically based on a clock interrupt.

Non-preemptive: If a process in the RUNNING state, it continues to execute until: **(a)** it terminates or **(b)** it blocks itself to wait for I/O operations or to request some operating system services.

Question 4

No. Once a process gains control of the CPU, it can retain the control until it blocks (due to I/O operations) or terminates. A process could execute for an extended period of time doing neither of the above. Other processes in the system would not be able to execute, hence producing unacceptable response time.

Question 5

The scheduling is done on a *preemptive* (at time quantum) basis, and a dynamic *priority* mechanism is used. When a process first enters the system, it is placed in the `RQ0` queue (with the highest priority level). After its first iteration, when it returns to the READY state, it is placed in the `RQ1` queue (with the second highest priority level). Each subsequent time that it is preempted, it is demoted to the next lower priority queue.

A shorter process will complete quickly, without migrating very far down the hierarchy of READY queues. A longer process will gradually drift downward. Thus, newer and shorter processes are favoured over older and longer processes.

3.2 Problem-Solving Tasks

3.2.1 Task 1

There is no limit to the number of processes that can be in the READY and BLOCKED states. At most, `n` processes may be in the RUN state, since a process in the RUN state must be allocated to a CPU, and there are only `n` CPUs.

3.2.2 Task 2

Process	Arrival Time (in seconds)	Processing Time (in seconds)
A	0	3
B	1	6
C	4	4
D	6	2

(a) **First-Come-First-Served (First-In-First-Out)**:

- `A` (from 0 to 3), `B` (from 3 to 9), `C` (from 9 to 13), `D` (from 13 to 15)

(b) **Shortest Process Next**:

- `A` (from 0 to 3), `B` (from 3 to 9), `D` (from 9 to 11 as `D` would have arrived and waiting in the queue along with `C`), `C` (from 11 to 15)

(c) **Round Robin (with time slice of 2 seconds)**:

- `A` (from 0 to 2), `B` (from 2 to 4), `A` (5), `C` (from 5 to 7), `B` (from 7 to 9), `D` (from 9 to 11), `C` (from 11 to 13), `B` (from 13 to 15)

3.2.3 Task 3

The **turnaround time** is computed by subtracting the time that the process entered the system from the time it finished (terminated).

(a) **First-Come-First-Served (First-In-First-Out)**:

- ((3 - 0) + (9 - 1) + (13 - 4) + (15 - 6)) / 4 = 7.25

(b) **Shortest Process Next**:

- ((3 - 0) + (9 - 1) + (15 - 4) + (11 - 6)) / 4 = 6.25

(c) **Round Robin (with time slice of 2 seconds)**:

- ((5 - 0) + (15 - 1) + (13 - 4) + (11 - 6)) / 4 = 8.25

3.2.4 Task 4

The **throughput** is defined as the number of jobs completed in a unit of time, or its inverse, the **average time taken to complete a job**. Each of the scheduling methods completed the execution of 4 jobs in 15 seconds; hence the throughput is `4 / 15` or `15 / 4 = 3.75` seconds per job.

3.2.5 Task 5

(a) Because the READY queue has multiple pointers to the same process, the system is giving that process preferential treatment. That is, this process will get double the processor time than a process with only one pointer.

(b) The advantage of this scheme is that more important jobs could be given more processor time by just adding an additional pointer (i.e. very little extra overhead to implement).

(c) To give longer time slice to processes deserving higher priority:

  (i) Add a bit in PCB that indicates whether a process is allowed to execute two units of time slice;

  (ii) Add an integer in PCB that indicates the number of time slices a process is allowed to execute;

  (iii) Implement two READY queues — one of which has a longer time slice for higher priority jobs.