COMP3520 Operating Systems Internals Assignment 2 – Uniprocessor Scheduling

General Instructions

This assignment is about uniprocessor scheduling. It consists of two compulsory tasks:

1. **Revise Programming Exercise 3** and implement a Round Robin Dispatcher (RRD); and
2. Answer the discussion document questions that are provided in a separate document.

This assignment is an individual assignment. Whilst you are permitted to discuss this assignment with other students, the work that you submit must be your own work. You should not incorporate the work of other students (past or present) into your source code or discussion document.

You will be required to submit your source code and discussion document to *Turnitin* for similarity checking as part of assignment submission. The marker may use other similarity checking tools in addition to *Turnitin*. Your source code may also be checked.

Submit your source code and discussion document to the appropriate submission inboxes in the COMP3520 Canvas website.

Round Robin Dispatcher

For Assignment 2, you will have a list of randomly generated jobs that you will need to schedule for execution depending on the job's time of arrival and its allowed execution time. Unlike Programming Exercise 3, each job will only be allowed to run for a certain amount of time ("time quantum") and then pre-empted (i.e, suspended and moved back to the ready queue if it has not finished yet **and** there are waiting jobs).

The process list format is the same as in Programming Exercise 3:

<arrival time>, <cputime>

0, 3

2, 6

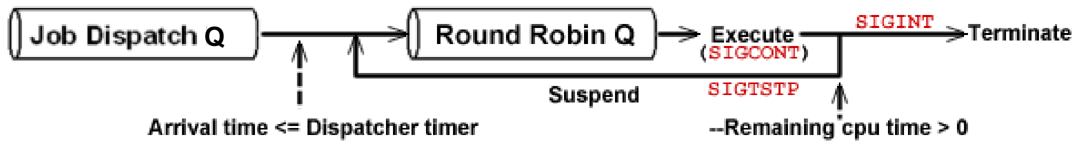
4, 4

6, 5

8, 2

The arrival times and corresponding job execution durations will be in the same manner as that in Programming Exercise 3.

For this Assignment, you will need to use a Job Dispatch queue **and** a Round Robin queue. At the beginning all jobs in the job dispatch list file are loaded to the Job



Dispatch queue; when a job has "arrived", it is moved from the Job Dispatch queue to the Round Robin queue. Whenever there is no currently running process, the job at the front of the Round Robin queue (if "arrived") is either launched as a new process (if it has not yet started running) or resumed (if it is a suspended process).

The Round Robin queue has a time quantum that is specified by the user. Whenever a running process is launched or resumed, it runs until one of the following conditions is met:

- The process's remaining execution time reaches zero; or
- The process has run for at least the time quantum without interruption **and** there are waiting jobs in the Round Robin queue.

In the first case, the process is terminated; in the second case, the process is pre-empted (i.e., suspended and moved back to the end of the Round Robin queue). The despatcher will immediately schedule another job in the Round Robin queue (if there are "arrived" and/or "suspended" ones) for execution when the currently running process is terminated or suspended.

Process Suspension, Resumption and Termination

The signals that will be used for process control in this assignment are as follows:

- **SIGTSTP** – Suspend a running process;
- **SIGCONT** – Resume execution of a suspended process; and
- **SIGINT** – Terminate a process.

Round Robin Algorithm

To assist you to tackle this assignment, a recommended pseudo code for the Round Robin dispatcher is provided.

Note: The subroutines for manipulation of a singly linked list, i.e., *createnullPcb()*, *enqPcb()* and *deqPcb()* have been provided in Programming Exercise 3. Also, the subroutines for creating/restarting and terminating a process *startPcb()* and *terminatePcb()* are the same as those in Programming Exercise 3. In this assignment you need to implement a subroutine *suspendPcb()* to suspend a process. Recall that the *kill()* function may be used to send various signals to processes. To ensure synchronization between the dispatcher and the child process, you need to make the following function call immediately after a **SIGTSTP** or **SIGINT** signal has been sent:

```
waitpid(p->pid, NULL, WUNTRACED);
```

Pseudo code for Round Robin Despatcher

```
1. Initialize the queues (Job Dispatch and Round Robin queues);
2. Fill Job Dispatch queue from job dispatch list file;
3. Ask the user to enter an integer value for time_quantum;
4. While there is a currently running process or either queue is not empty
     i.  Unload any arrived pending processes from the Job Dispatch queue
         dequeue process from Job Dispatch queue and enqueue on Round Robin
         queue;
     ii. If a process is currently running
```

```
            a.  Decrease process remaining_cpu_time by quantum;
            b.  If times up
                A.  Send SIGINT to the process to terminate it;
                B.  Free up process structure memory;
            c.  else if other processes are waiting in Round Robin queue
                A.  Send SIGTSTP to suspend currently running process;
                B.  Enqueue it back to the tail of Round Robin queue;
    iii.    If no process is currently running and Round Robin queue is not empty
            a.  Dequeue a process from the head of Round Robin queue;
            b.  If the process job is a suspended process
                    Send SIGCONT to resume it;
            c.  else start it (fork & exec)
            d.  Set the process as the currently running process;
    iv.     Sleep for quantum (may/may not be the same as time_quantum, and may
            need to be calculated based on different situations);
    v.      Increase timer by quantum;
    vi.     Go back to 4.
5.  Terminate the Round Robin dispatcher.
```

Additional Requirements

Source Code

1. Your program must ask the user to enter an integer value for time_quantum.
2. You must keep the logical structure of the program for FCFS despatcher in Programming Exercise 3 intact, i.e., you can only make necessary changes to the program provided in Programming Exercise 3 for Round Robin despatcher.
3. Your program must be able to correctly calculate and print out on the screen the average turnaround time and average waiting time.
4. Your program must be implemented in the C programming language. C++ features are not permitted except those that are part of an official C standard.
5. Your source code needs to be properly commented and appropriately structured to allow another programmer who has a working knowledge of C to understand and easily maintain your code.
6. You need to include a *makefile* that allows for the compilation of your source code using the *make* command on the School of Computer Science servers.

Testing and Debugging

You are responsible for testing and debugging your source code.

It is crucial that you ensure that the source code you submit compiles correctly on the School of Computer Science servers. If you submit source code that cannot be compiled on the School servers, you will receive a **failing** mark for it.

Discussion Document

You are required to answer all assigned questions in a separate written document. The questions and requirements specific to the discussion document will be provided in a separate document.