COMP3520 Operating Systems Internals
Programming Exercise 8
Round Robin Dispatcher

General Instructions

In this exercise, you will study a Round Robin dispatcher (RRD) for a single-processor system. You are asked to revise the program you have completed for FCFS dispatcher in Programming Exercise 7 and implement a Round Robin dispatcher. (A suggested solution program for FCFS dispatcher, i.e., fcfs_sol.c is provided. You may revise this program for a RR dispatcher.)

Please bear in mind that this exercise is intended to introduce you the concepts that you will need in order to complete COMP3520 Assignment 2.

This exercise will not be marked; assessment for COMP3520 Assignment 2 will be based only on your final source code and discussion document.

Round Robin Dispatcher

For this exercise, there is a list of randomly generated jobs that need to be scheduled for execution depending on the job's time of arrival and its allowed execution time. Unlike Programming Exercise 7, 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 7:

<arrival time>, <cputime>

0, 3

2, 6

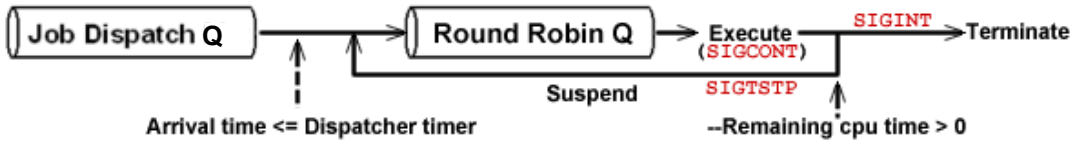
4, 4

6, 5

8, 2

The arrival times and corresponding job execution durations will be in the same manner as that in Programming Exercise 7.

For this exercise, 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 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 dispatcher 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 programming exercise, a recommended pseudo code for the Round Robin dispatcher is provided below.

Note: The subroutines for manipulation of a singly linked list, i.e., *createnullPcb()*, *enqPcb()* and *deqPcb()* have been provided in Programming Exercise 7. In this exercise you need to implement subroutines *suspendPcb()* and *resumePcb()* to suspend and restart a process in *pcb.h* and *pcb.c*. 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 Dispatcher

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.

- Your program needs to ask the user to enter an integer value for time_quantum.
- You should keep the logical structure of the program for FCFS dispatcher in Programming Exercise 7 intact and only make necessary changes for Round Robin dispatcher.
- Your program needs to correctly calculate and print out on the screen the average turnaround time, average waiting time and average response time.