COMP3520 Operating Systems Internals Assignment 3: Stage 3 – Swapping

General Instructions

This is Stage 3 of Assignment 3. It is about multi-level queue dispatcher under the memory constraints **with swapping** and consists of two compulsory tasks:

1. **Revise Assignment 3 Stage 2 program** to further add a simple swapping management scheme for a uniprocessor scheduling system (**70%**); and
2. Answer the discussion document questions (**30%**).

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.

Your work for Stage 3 will be marked (40% of the total marks for Assignment 3). You must first complete tasks for Stages 1 & 2 before tackling problems for Stage 3. You will obtain a **Failing** mark if you only submit your work for Stage 3 without submitting your work for Stages 1 & 2.

You are required to submit your **source code**, **job dispatch list files**, **makefile**, and **discussion document** together in a **zip/tar file** to the submission inbox in the COMP3520 Canvas website.

Job Scheduling under Memory Constraints with Swapping

In Stage 2 of Assignment 3, you added a simple memory management scheme to a multi-level queue dispatcher which you have done in Stage 1. There we assumed that there are two types of jobs, i.e., real-time jobs, and normal jobs. Real-time jobs are given a higher priority and will be scheduled to run only if the required memory can be allocated. One potential problem with that scheme is that high priority real-time jobs may be blocked for a long time before being admitted into the system for execution if the memory space is all occupied by long running low priority jobs. One way to alleviate this problem is to temporarily swap out some low priority jobs. In Stage 3 of the assignment you will further add a simple swapping management scheme, which is described as follows.

1. The requirements for job admission and dispatching are the same as that described in Stage 2, except for admitting real-time jobs and swapping in/out “Level 2” jobs.
2. When a real-time job “arrives”,
    a. if the required memory can be allocated, it will be moved from the “Arrived RT job queue” to “Level-0 queue” in the multi-level ready queue as that in Stage 2.
    b. if there is no sufficient free memory, a “Level-2” or “priority 2” job will be swapped out of main memory and the allocated memory is freed. The freed memory will immediately be allocated to the new real-time job which is

then loaded to Level-0 queue in the multi-level ready queue and ready to be scheduled for execution. **Note:** only Level-2 jobs can be swapped out.

3. When a Level-2 job is swapped out, it will be queued to the end (or tail) of a "Swapped job queue".
4. When there are several Level-2 jobs, for swapping we select one which has **the longest remaining execution time**. (You may add a subroutine in pcb.c for this.)
5. Jobs in the Swapped job queue can be swapped in and scheduled to run again only when all the Arrived job queues and the Three-level ready queue are empty.
6. When being able to be swapped in, jobs in the Swapped job queue are swapped in and scheduled to run one by one in a FCFS manner, that is, they can only be swapped in one at a time.
7. When being swapped in, memory is reallocated, and the job will remain as a Level-2 job and be scheduled to run the same way as other Level-2 jobs.
8. To simplify the scheme, we assume that each real-time job will require 32MB of memory, while each normal job requires a minimum of 32MB memory. An example of the job dispatch list is given as

<arrival time>, <cputime>, <priority>, <mem requirement>

0, 13, 1, 128

2, 6, 0, 32

4, 14, 1, 65

6, 15, 1, 350

8, 2, 0, 32

Requirements

Source Code

1. Your program must ask the user to enter an integer value for time-quantum for Level-1 queue.
2. You must keep the logical structure of the program for Multi-level Queue Dispatcher under memory constraints in Stage 2 of Assignment 3 intact, i.e., you can only make necessary changes to those programs.
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. It is crucial that you ensure that your source code compiles correctly on the School of Computer Science servers. If your source code cannot be compiled on the School servers, you will receive a **failing** mark for it.

Discussion Document

1. Write a pseudo code for your multi-level queue dispatcher under memory constraints with swapping. Your pseudo code must have a similar style to that for Round Robin Dispatcher given in Assignment 2 description.
2. Give **2, or 3** job dispatch list files, each list containing **at least 6** jobs, you designed for testing and debugging various aspects of your source code to make sure your code is bug free and can produce correct results. You must explain **how** these job dispatch lists are designed for **what** purposes.