COMP3520 Operating Systems Internals Assignment 3 – Memory Requirement Aware Dispatcher

General Instructions

This assignment is about dynamic memory allocation and memory aware scheduling. It consists of two compulsory tasks:

1. Implement the dispatcher as described in the section "Memory Requirement Aware Dispatcher (MRAD)"; 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 **must not** incorporate the work of any other student (past or present) into your source code or discussion document. Also, you must **not** use or refer to material from pirate websites.

You will be required to submit your source code and discussion document to *Turnitin* for similarity checking as part of assignment submission. The examiner 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.

Memory Requirement Aware Dispatcher (MRAD)

In this assignment, you will implement a dispatcher called the "Memory Requirement Aware Dispatcher (MRAD)" that meets the required specifications. MRAD is a multiprogramming system with a round robin dispatcher that supports swapping jobs into and out of simulated physical memory of size 512 MB.

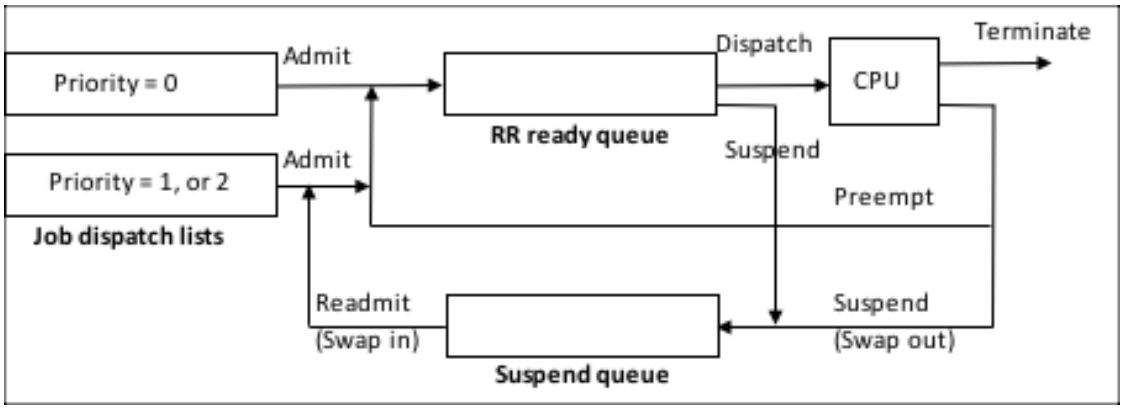

Figure: Overview of Job Scheduling in MRAD

A program for a round robin dispatcher is provided; however, the memory allocation routines are incomplete. In this assignment, you need to modify this program such that it complies with the following requirements:

1. Each job is explicitly assigned a priority level (0, 1 or 2) and memory requirement (in megabytes). For example, consider the following job list, which is in <arrival time>, <priority>, <memory requirement>, <processor time> format:
   1, 1, 128, 4
   3, 0, 64, 2
   4, 2, 384, 6

   The meaning of this job list is as follows:

   1st job: Arrival at time 1 second, priority 1, requires 128 MB of simulated memory and 4 seconds of CPU time

   2nd job: Arrival at time 3 seconds, priority 0, requires 64 MB of simulated memory and 2 seconds of CPU time

   3rd job: Arrival at time 4 seconds, priority 2, requires 384 MB of simulated memory and 6 seconds of CPU time
2. The time quantum of the dispatcher is one second.
3. During the lifetime of each job, its priority **never** changes.
4. MRAD must support dynamic contiguous memory allocation for jobs as described in exercise 5; however, the "**next-fit**" (not first-fit) memory partitioning scheme must be used.
5. The total amount of simulated memory available for allocation to jobs is 512 MB.
6. MRAD must support Memory Aware Job Swapping as described in the following subsection, which is titled "Memory Aware Job Swapping".
7. Whenever a job in the RR ready queue is selected for execution, MRAD launches it if it is a new job or sends the **SIGCONT** signal to it if it has already been launched. Also, MRAD calls the *printPcb()* and *printPcbHdr()* functions to print the offset and size of the simulated memory block that has been allocated to the job.
8. No more than one job may be in the running state at any time.
9. Whenever a running job has run for its allotted execution time, MRAD terminates it by sending the **SIGINT** signal to it.
10. Whenever a job is pre-empted or swapped out, MRAD sends a **SIGTSTP** signal to it.
11. There are two job dispatch queues (marked "Job dispatch lists" in the figure). One of these (priority 0 job dispatch queue) is for priority 0 jobs (high priority jobs); the other (priority 1 or 2 job dispatch queue) is for priority 1 (normal jobs) or priority 2 jobs (low priority jobs). For each job dispatch queue, jobs are enqueued and dequeued on a first-in-first-out (FIFO) basis according to their arrival times.
12. When MRAD starts, it loads all jobs from a job list file into the appropriate job dispatch queues before admitting jobs to the "RR ready queue" for execution. It must reject all jobs with a memory request in excess of 512 MB (in other words, such jobs must not be loaded from the job list file into the job dispatch queues).
13. Jobs are admitted to the "RR ready queue" at the beginning of every time quantum, subject to the requirements listed below.
    a. An "arrived" job is admitted to the "RR ready queue" if and only if the job has been allocated a suitable free simulated memory block that satisfies the job's memory requirement.

   b. In admitting jobs to the "RR ready queue", the dispatcher gives "arrived" high priority jobs in the priority 0 dispatch queue absolute priority over jobs in the other dispatch queue or the "Suspend queue". No "arrived" lower priority (i.e., priority 1 or 2) jobs (or "Suspended" jobs) can be admitted (or re-admitted) from the priority 1 or 2 job dispatch queue (or the "suspend queue") to the "RR ready queue" for execution when there is one or more "arrived" high priority jobs (i.e., priority 0 jobs) in the priority 0 job dispatch queue waiting to be admitted into the "RR ready queue".
   c. If there are no "arrived" jobs in the priority 0 job dispatch queue, then:
      i. If there is no "arrived" priority 1 job at the front of the priority 1 or 2 job dispatch queue, then the dispatcher attempts to re-admit as many jobs (if any) in the "Suspend queue" to the "RR ready queue" as possible in accordance with the procedure described in the subsection "Memory Aware Job Swapping";
      ii. If there is an "arrived" priority 2 job at the front of the priority 1 or 2 job dispatch queue and there are also jobs in the "Suspend queue", then jobs in the "Suspend queue" will be considered for re-admission first.

14. When the dispatcher attempts to admit a job in the priority 0 dispatch queue to the "RR ready queue", but there is not sufficiently large free memory space available, the dispatcher attempts to swap out one (and only one) priority 2 job that is either running or in the "RR ready queue" as described in the subsection "Memory Aware Job Swapping".
    a. If the job swap is successful, the priority 0 job is allocated the free memory block that became available as a result of the job swap. Also, this job is admitted to the "RR ready queue".
    b. If the job swap is unsuccessful, the priority 0 job must wait in the dispatch queue until its memory requirement is satisfied; subsequently, the job is admitted to the "RR ready queue" for execution.
15. If an "arrived" job that is not currently in the "Suspend queue" cannot be admitted to the "RR ready queue" due to memory allocation failure, then no further jobs may be admitted to the "RR ready queue" for the remainder of the current time quantum.

However, you must not modify the *printPcb()* or *printPcbHdr()* functions in the "pcb.c" file.

Memory Aware Job Swapping

MRAD supports swapping of jobs into and out of simulated memory. The steps to swap a job out of simulated memory are as follows.

When a priority 0 job that has "arrived" is waiting in the job dispatch queue for admission and there is no sufficiently large contiguous free memory space available, the system will attempt to suspend ONE (and only one) priority 2 job either running or in the "RR ready queue" and find a free memory block whose size is equal to or larger than the memory size requested by the priority 0 job.

1. The search for a "suitable" priority 2 job to suspend starts from the head of the "RR ready queue" and the first such priority 2 job will be selected for swapping.
2. The purpose of swapping is to find a sufficiently large free memory block for the priority 0 job. Thus, the search may involve the free memory block merge operation.

   a. You need to consider the case when the allocated memory block for a priority 2 job is not large enough, but the combination of this block and its adjacent free memory blocks will be equal to or larger than the requested size. For example, a priority 0 job requests a 64 MB memory block. One priority 2 job is allocated 50 MB which is smaller than 64 MB. However, its adjacent memory block is a free memory block of size 20 MB. In this case, this priority 2 job is swapped out and its memory is deallocated. Now, the two free memory blocks can be merged together to become a single memory block of size 70 MB which is larger than the requested 64 MB.
   b. Note that when a simulated memory block is deallocated from a job (whether larger than, equal to or smaller than the requested size), if there are adjacent free simulated memory blocks, then the free blocks must be merged to form one larger contiguous free simulated memory block.
3. After the simulated memory is deallocated from the selected job, this priority 2 job is placed at the tail end of the "Suspend queue".

The rules for swapping a job from the "Suspend queue" into the "RR ready queue" are as follows:

1. If there are no "arrived" jobs in the priority 0 dispatch queue and there is no "arrived" priority 1 job at the front of the other dispatch queue, then the dispatcher attempts to admit as many jobs in the "Suspend queue" to the "RR ready queue" as possible.
2. Start at the head of the "Suspend queue" and traverse the queue. For each job in this queue, if a simulated memory block can be immediately allocated to this job, then the dispatcher allocates a suitable memory block to this job and admits this job to the "RR ready queue"; otherwise, the dispatcher does not admit this job and attempts to admit the next job in the "Suspend queue".
3. This process continues until the end of the "Suspend queue" is reached.

Additional Requirements

Source Code

The source code for a round robin dispatcher is available on the COMP3520 Canvas website. This code consists of the files "mrad.c", "mrad.h", "pcb.c", "pcb.h", "mab.c" and mab.h". **You may only modify this code for your MRAD implementation. A completely new program will not be accepted for assessment.**

Additionally, the source code for the test process is supplied as "sigtrap.c". Source code for a random job generator is supplied in "mem2_random.c".

Your solution must be implemented in the C language. C++ features are not permitted except those that are part of an official C standard.

Your source code needs to be properly commented in *Academic English* and appropriately structured to allow another programmer who has a working knowledge of C to understand and easily maintain your code.

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.

You should include your University student identification number (SID) (but **not** your name) in the space provided in the "mrad.c" file.

Readme File

You need to write a *readme* file to explain to the user how to compile and run your dispatcher. This document must be written in *Academic English* and be appropriate to audience, purpose and context. The file format should be Plain Text (txt) format.

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 and that the resulting binary functions as intended. 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.

Other Matters

Non-serious attempts, non-attempts, and acts of academic dishonesty are contrary to the spirit of this assignment and COMP3520; accordingly, such behaviour may be sanctioned with **DISQUALIFICATION** from this assignment.

You must **not** include frivolous or offensive material such as inappropriate filenames in your assignment submission. Marks may be reduced for inappropriate or irrelevant material.

Marking criteria for the source code, *readme* file and discussion document will be provided separately.