SYSC4001 Operating Systems, Fall 2013
Assignment 3
Due: Monday Dec 9th

Purpose: The intention of assignment 3 is to design scheduling software using Linux threads.

Some basic information:
To solve this assignment, you need to understand basic Linux threads and scheduling. The textbook together with the reference book provide a solid background for this assignment. The scheduling part is a simplified Linux scheduling policy.

Assignment description:
The assignment is to design and implement a variation of the multilevel scheduling algorithm using multiple threads. The following describes the main components:

- Queues. There are three ready queues (RQ0, RQ1, and RQ2), representing decreasing order of priority. Either circular buffers (similar to assignment 2) or linked lists can be used for the queues.

- Threads and scheduling. There are two types of threads: main thread and scheduling thread. The **main thread**, similar to the producer in the producer/consumer problem, will randomly "generate" a few processes (only process information, e.g., ID and priority, ***not*** actual process creation using *fork()*) with an expected execution time for each process (details are described later). The main thread creates all processes, e.g., 30 (or more) for clear demonstration, with the process information, in the beginning and put them in the appropriate RQ's.

  There are three different types (representing three priority levels) of processes: Gold (real-time), Silver, and Bronze. The ratio of those three types is about 1:2:2. This can be realized using random number (N) generation in the range between 0 and 1:

  - Gold: $0 <= N < 0.2$
  - Silver: $0.2 <= N < 0.6$
  - Bronze: $0.6 <= N < 1.0$

  After the main thread creates a "process" (only the process information), it will put the process into one of those queues depending on the random number. The process information includes:
  - PID (you can number them incrementally)
  - Priority value (Gold, Silver, or Bronze, or equivalent numerical values, e.g., 0, 1, 2),
  - Expected processor time needed, e.g., from 100 ms to 5 sec.

  The other four **scheduling threads** are used to emulate four processors (quad-core). These four scheduling threads, similar to the consumers in the producer/consumer problem, will select a process each time from a ready queue and "execute" the process. By executing, it simply means that the thread will sleep until the time quantum or the expected service time expires, see below for more details. Either pointers or arrays (circular buffers) can be used for the queue management. But synchronization (similar to the producer/consumer problem) needs to be handled. Synchronization can be done using either semaphores or pthread_lock/unlock().

The execution time interval or time quantum is typically a multiple of 10ms (in general, this is the minimum granularity for current Linux OS). The expected execution time for Gold real-time processes ranges from 50ms to 120ms. The policy for RQ0 is **FCFS**, **non-preemptive**.

For Silver and Bronze levels, the expected execution time will be in the range of 0.5 sec to a few seconds, e.g., 5 or 10. A process selected from RQ1 and RQ2 is allowed to run up to 50ms and 100ms, respectively, for each iteration before preemption, i.e., the default time quantum values for those two queues. After a process is selected from the ready queue and exhausts the time quantum (or runs to completion), the remainder execution time needs be updated.

The four **scheduling threads** always start from the highest-priority queue, e.g., RQ0, first and then check each queue in sequence. Only when there is no task in a RQ0 will the threads move to the next level. For RQ1 and RQ2, the ratio for selection is 2:1, i.e., two tasks from RQ1 will be selected before the threads check RQ2 and select one task in RQ2, unless RQ1 is exhausted.

Output: A print statement is needed when a process is selected and finishes the iteration. In the output, messages regarding the execution of each process need to be printed. Examples include: process ID, priority, initial expected execution time, queue level, service time for the current iteration, accumulated service/execution time, and turnaround time if a process completes its entire execution.

Check the course slides or textbook for the definition of turnaround time.

Note that if the program runs very fast and it becomes difficult to see the output clearly, the time quantum values can be increased.

To sleep at the *ms* level, sleep() does not work. You can use usleep() or select() to do it. To use select(), you should include <sys/types.h>. Here is a code segment for your reference. Take a look at the Linux manual ($man select) for more details.

```
struct timeval timeout;

// set second to 0, since we are dealing with ms level
timeout.tv_sec = 0;

// convert timeSlice in milliseconds to microseconds
// as used in select ()
timeout.tv_usec = (long)timeSlice * 1000;

select(0,NULL,NULL,NULL,&timeout);
```

Grading criteria
Marking is based on the following criteria:

Correctness (including error checking, message labeling): 80%
Documentation: 10%
Program structure: 5%
Style and readability: 5%

Again, as we know that customer requirements are constantly changing in practice and requirements rarely are perfect, I also reserve the right to make changes (only minor ones) to the assignment, but the requirement will be finalized as least one week before.