Programming Project 3 – Process scheduling

Deadline: **Friday March 23 by midnight**
Type: **Group of 2 students maximum**
Weight: **This programming project is worth 10% of your final grade**
Submission: **Must be on Moodle**

Policy for late hand-in:

One-day delay will result in 20% mark reduction. Two days delay will result in 40% mark reduction. After that, the programming project will not be accepted.

Marking Scheme:

- Stress tests and efficiency (30%)
- Code correctness (45%)
- Specs compliance (15%)
- Report (10%)

Goal

This programming project involves implementing and simulating a priority-based scheduler in a system with a single processor. The algorithm to be implemented is a very simplified version of the O(1) scheduler which runs in constant time. It is a kernel scheduling design used in versions of Linux kernel 2.6 prior to 2.6.63. It can schedule processes within a constant amount of time regardless of the size of the input. A document about the structure and operation of the O(1) algorithm is provided with this project.

Priority-Based Scheduler (PBS)

CPU scheduling is the basis of multiprogrammed operating systems. By switching the CPU among processes, the operating system ensures that all processes have a fair share of the CPU time. In this project, we consider the case of a single-processor system where only one process can run at a time. Others must wait until the CPU is free and can be rescheduled. By associating a priority to each process, the algorithm to be implemented and simulated in this part of the project performs as follow:

- The scheduler has two queues, one queue is flagged as active while the other is flagged as expired.
- Processes that request the CPU are selected by the scheduler from the active queue.
- Several processes may exist in the system and each process has its own PID, arrival time, CPU burst and priority.
- Priorities are indicated by a fixed range from 1 to 139.
- The scheduler starts at time unit of 1 second. It allocates a time quantum (T_q) to the first process in the active queue and dispatches it. Once the time quantum expires, the running process is stopped and moved to the expired queue. The time quantum is computed as follows:

$$T_q = (140 - \text{priority}) * 20 \text{ (milliseconds) if priority} < 100$$
$$T_q = (140 - \text{priority}) * 5 \text{ (milliseconds) if priority} \geq 100$$

- When a new process arrives, its PID is inserted in the expired queue. The expired queue is arranged in an increasing order of process priorities.
- When the active queue is empty, the scheduler switches the flags of the two queues. Then, it starts selecting a process from the processes residing in the new active queue.
- The scheduler's logic is basically a cycle of two activities:

(1) Execute the processes in the queue flagged as active.
(2) If the queue flagged as active is empty, change flags of both queues and go back to (1).

- The scheduler updates the priority of any process having been granted two successive time quanta. The priority is updated according to the following relationships:

$$\text{waiting\_time} = \sum \text{waiting\_times}$$
$$\text{bonus} = \text{ceil}(10 * \text{waiting\_time}/(\text{current\_time} - \text{arrival\_time}))$$
$$\text{New\_priority} = \max(100, \min(\text{old\_priority} - \text{bonus} + 5, 139))$$

Example of Operation

Let Q_1 and Q_2 be flagged (at time = 0) as active and expired queues, respectively. Let P_1 and P_2 be two processes in memory. Before time = 1, process P_1 arrives and it is moved into Q_2. Assume P_1 has a priority and a CPU burst of 90 and 2.5 seconds, respectively. At time = 1, the scheduler checks Q_1 and finds it empty. Accordingly, it changes the flags of both queues. Note that Q_2 contains now P_1. At this point, P_1 starts running during its first 1 second time quantum and once completes, it is moved back to Q_1. Suppose that at time = 2, a second process P_2 arrives with a priority and a CPU burst of 120 and 100 milliseconds, respectively. It is inserted into Q_1 right after P_1 (the queue should remain sorted). Since Q_2 is now empty, the scheduler switches the flags of Q_1 and Q_2, and then resumes execution of P_1 for another 1 second time quantum. Note that Q_1 is active now and it contains P_1 and P_2 in this order. As P_1 has 500 milliseconds remaining in its CPU burst, its priority should be updated by the scheduler. Note that the waiting time is 0 for P_1. Thus, its priority will be changed to 100 and it will be moved back to Q_2. At this point, P_2 begins running and completes its CPU burst after 100 milliseconds. Note that Q_2 now contains only P_1, while Q_1 is empty. Accordingly, the scheduler changes again the flags of both queues by making Q_2 active and Q_1 expired. Then, it calculates the new time quantum to be granted to P_1 which becomes 200 milliseconds. The process P_1 will be granted two additional time quanta before getting its priority upgraded to 105. Finally, it runs during 175 milliseconds before it terminates.

Input-Output Data

The input of the algorithm consists of a text file called “pbs_input.txt”. The first line of the input file represents the number of processes in the file. The subsequent lines contain pieces of information associated with each process: PID, arrival time, CPU burst, and initial priority. A text file that contains these pieces of information (in that order) will be provided to you during the demo. Note that the time values should be indicated in milliseconds (arrival time and CPU burst).

```
Sample “pbs_input.txt”:

3
P1 1000 2500 90
P2 2000 100 120
P3 3200 100 120
```

The output of the algorithm should also be written to a text file called “pbs_output.txt”. The latter contains a set of strings indicating events in the program and includes:

- Processes arrival events: **Time 3200, P3, Arrived**
- Processes start/pause/resume events and granted time slots: **Time 2000, P1, Resumed, Granted 1000**
- Processes termination events: **Time 3100, P2, Terminated**
- Processes priorities updates including new priority value: **Time 3000, P1, priority updated to 105**

```
Sample “pbs_output.txt”:

Time 1000, P1, Started, Granted 1000
Time 2000, P1, Paused
Time 2000, P2, Arrived
Time 2000, P1, Resumed, Granted 1000
Time 3000, P1, Paused
Time 3000, P1, priority updated to 105
Time 3000, P2, Started, Granted 100
Time 3100, P2, Terminated
Time 3100, P1, Resumed, Granted 200
Time 3200, P3, Arrived
Time 3300, P1, Paused
Time 3300, P1, Resumed, Granted 200
Time 3500, P1, Paused
Time 3500, P1, priority updated to 110
Time 3500, P3, Started, Granted 100
Time 3600, P3, Terminated
Time 3600, P1, Resumed, Granted 150
Time 3700, P1, Terminated
```

Implementation Requirements

Here is the set of requirements for the algorithm:

- The program **MUST** work with an arbitrary number of processes.
- Both the scheduler and simulated processes **MUST** be simulated using threads.
- The scheduler **MUST** be able to pause and resume each process.
- Only one process **MUST** be running at a time. That is, the scheduler thread **MUST** ask a simulated process (e.g., P_1) to suspend its execution and allocates the CPU to the next simulated process (e.g., P_2) based on the above algorithm specification.
- Each process **MUST** keep track of its total execution and it **MUST** inform the scheduler thread when it terminates.
- This processing continues until all processes terminate their CPU burst.

Companion Material

One file is provided:

- O1_Scheduling_Algorithm.pdf

Deliverable

The deliverable consists of the following:

1) A copy of a well-commented source code and a sample run (pbs_output.txt).
2) A three pages report specifying a high-level description of the program. The report should also include a brief conclusion.
3) In your report, indicate the status of your program by specifying whether:
   - The program runs with user-defined test cases.
   - The program runs with some errors.
   - The program compiles and runs with no output.
   - The program does not compile.