**CS2106 Operating Systems**

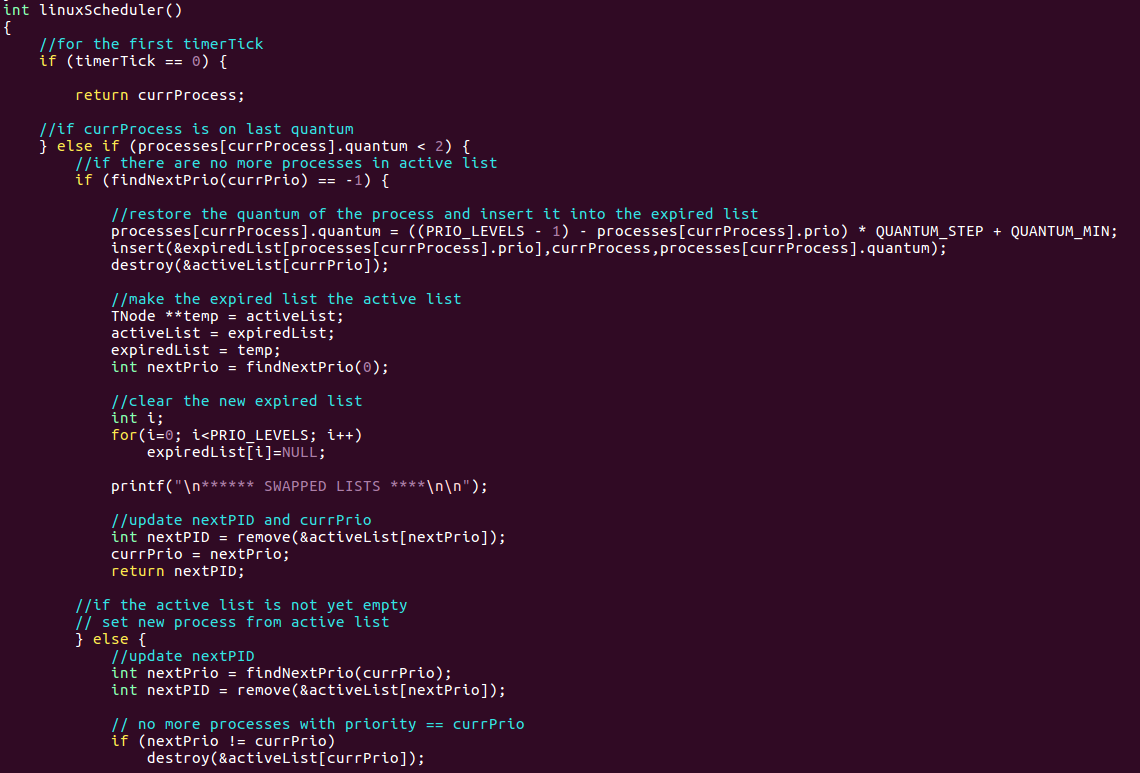
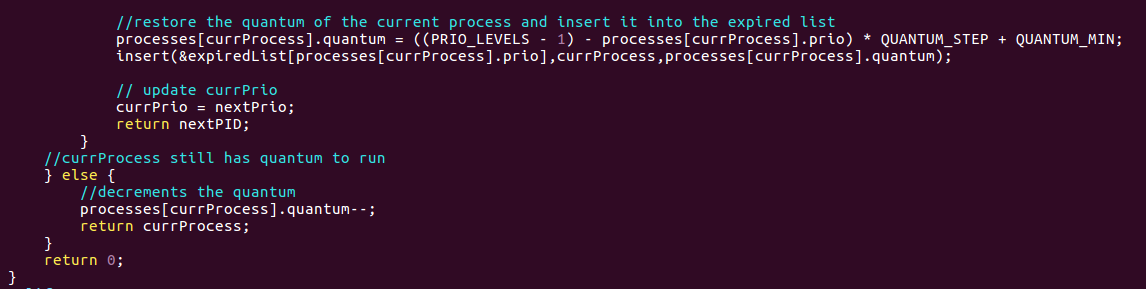
**2017/18 Semester II**

**Term Assignment Answer Book**

|  |  |
| --- | --- |
| STUDENT NUMBER 1:  **A0161868R** | STUDENT NAME 1:  **Joshua Lee Kai Sheng** |
| STUDENT NUMBER 2:  **A0161551M** | STUDENT NAME 2:  **Ryan Tan Wei Keat** |
| STUDENT NUMBER 3:  **A0155156E** | STUDENT NAME 3:  **Ng Jun Wei** |

**Question 1** (15 marks)

My code and explanation for the LINUX scheduler is shown below:

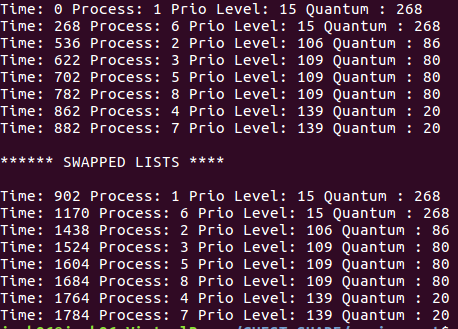
 

**Explanations**

**The linuxScheduler is broken down into the following segments:**

1. **If the current timerTick is 0**
   1. **Return currProcess**
2. **If currProcess is in its last quantum**
   1. **If the active list is already empty**
      1. **Restore the quantum of the current process and insert it into the expired list**
      2. **Swap the expired and active lists**
      3. **Update the nextPID and currPrio**
   2. **If the active list is not empty**
      1. **Set new process from the active list**
         1. **Update nextPID**
         2. **Restore the quantum of the current process and insert it into the expired list**
         3. **Update the currPrio**
3. **If currProcess still has quantum to run**
   1. **Decrements its quantum**

Here is a screenshot showing the output of my LINUX scheduler:



**Question 2** (5 marks)

One advantage of using array of queues:

**The insertion of a process into the array of queues takes only O(1) time: O(1) to search for the correct queue using the priority of the process, and O(1) to insert to the end of the particular queue.**

**In contrast, a single priority queue sorted in descending order by quantum will incur a process insertion time of O(log(N)), N being the number of processes, assuming a binary-search-like algorithm is used.**

One disadvantage of using array of queues:

**If the number of processes is much fewer than the number of priority levels, the array of queues might be more inefficient in dequeuing the next process. For example, there are two processes, A and B, with priority levels 139 and 0 respectively. For the array of queues method, after Process A has been executed, findNextPrio() will iterate through all priority levels (140) to detect Process B.**

**On the other hand, the single queue method will only incur O(1) time for dequeue as it simply just needs to dequeue Process B from the single list after Process A has completed its execution.**

**Question 3** (5 marks)

My pseudocode for “renice” is shown below:

**void renice (int adjust) {**

**Loop though active and expired queue to locate target process**

**Store all the parameters of the target process in temp**

**Remove the target process in the queue it belongs to**

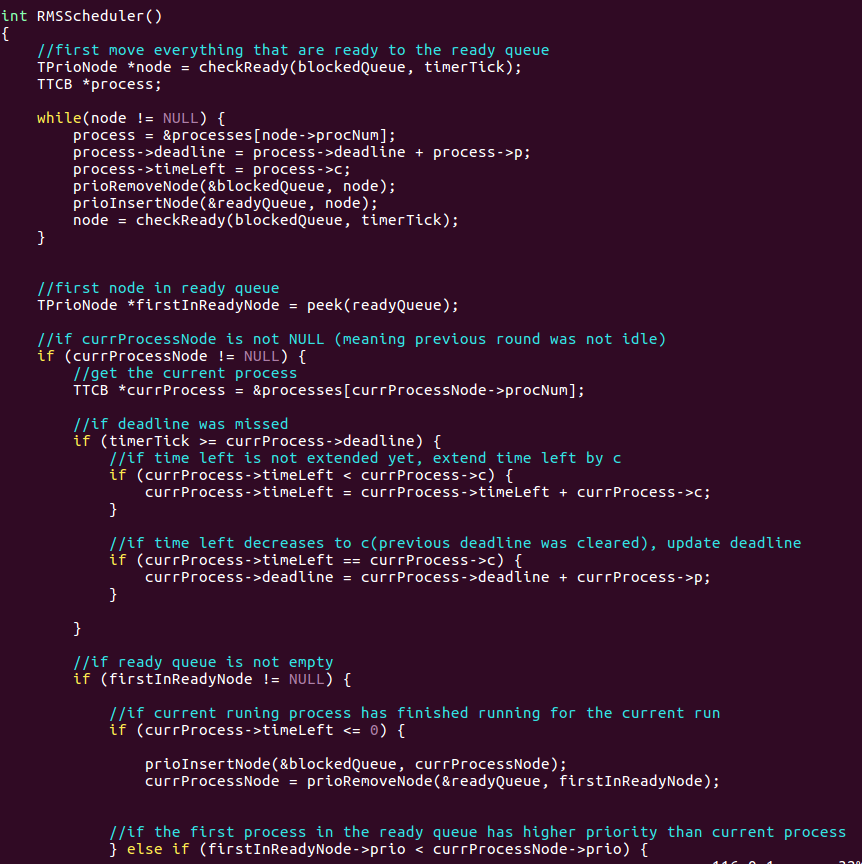
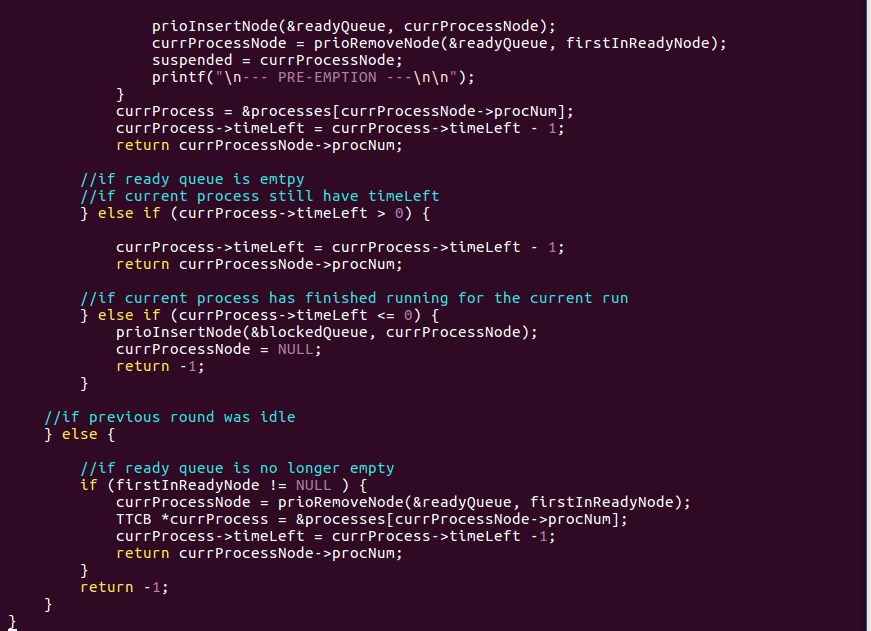
**Update the target’s (temp) priority by “adjust” and quantum (using formula)**

**Insert temp as a process into the queue it came from**

**}**

**Question 4.** (20 marks)

My code and explanation for the RMS scheduler is shown below:

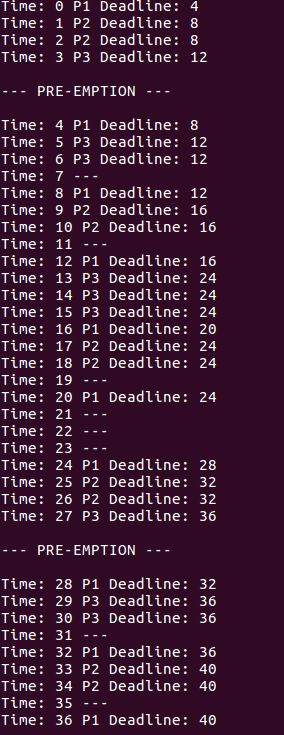
 

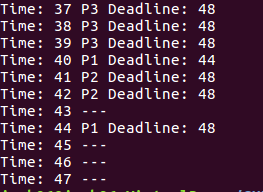
**Explanations**

**The RMSScheduler is broken down into the following segments:**

1. **First, move all processes that are ready from the blockedQueue to the readyQueue and update their deadline and restore their timeLeft**
2. **Then check if the currProcessNode is NULL**
   1. **If the currProcessNode is not NULL (previous round was not idle)**
      1. **If deadline was missed**
         1. **Extend the process’ timeLeft by c if it has not been done so**
         2. **Update the process’ deadline if the previous missed deadline has been met**
      2. **If the readyQueue is not empty**
         1. **If the current process has finished running**
            1. **Put the currProcessNode into the blockedQueue**
            2. **Remove the first node in the readyQueue and make it the currProcessNode**
         2. **If the first process in the readyQueue has higher priority than the current process**
            1. **Put the currProcessNode back to the readyQueue**
            2. **Remove the first node in the readyQueue and make it the currProcessNode and suspended**
         3. **Finally, reduce the timeLeft of the current process by 1 and return the current process number**
      3. **If the readyQueue is empty**
         1. **If the current process has not finished running**
            1. **Reduce the timeLeft of the current process by 1 and return the current process number**
         2. **If the current process has finished running**
            1. **Put the currProcessNode into the blockedQueue**
            2. **Make the currProcessNode NULL to make the next slot idle**
            3. **Return -1**
   2. **If the currProcessNode is NULL (previous round was idle)**
      1. **If readyQueue is not empty**
         1. **Remove the first node in the readyQueue and make it the currProcessNode**
         2. **Reduce the timeLeft of the current process by 1 and return the current process number**
      2. **Else (no process ready yet)**
         1. **Return -1**

Here is my screenshot of my RMS scheduler running:





**Question 5.** (5 marks)

**CPU Utilization using the formula is:**

**CPU Utilization by counting cycles is:**

They are the **same**. This is why:

**The theoretical CPU Utilization calculates the fraction of the total number of cycles that is spent doing work, with the condition ALL processes have complete runs.**

**For the counting cycle case, the timerISR() called executes for two complete runs, calculated using LCM. This ensures that all the processes would have completed their last runs during the execution time. Hence, the CPU Utilization by counting cycles is the same as that using the formula in this case.**

**Question 6.** (10 marks)

My modifications to turn this into an EDF scheduler are:

**On a high level, change the priority level to be the deadline of the process.**

1. **To accomplish this, first create a priority queue which sorts by deadline.**
   1. **The implementation of this priority queue is highly similar to that of the current one**
2. **Use this priority queue as the readyQueue**

**For the readyQueue,**

1. **Initially, to add a new process into the readyQueue,**
   1. **Input period as the prio as the deadline is the same as period at timerTick 0:**

**prioInsert(&readyQueue, procNum, period, period)**

1. **For each round, when moving processes that are ready from the blockedQueue to the readyQueue, update the deadline (priority):**

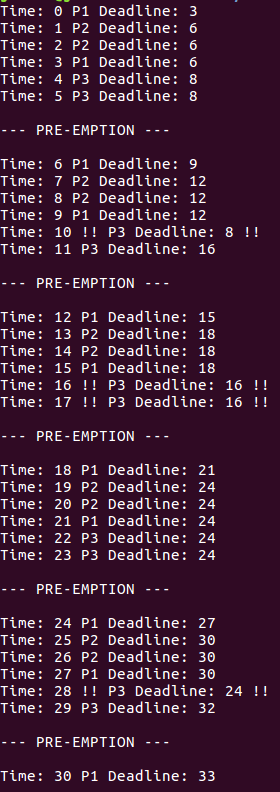
**prioRemoveNode(&blockedQueue, node);**

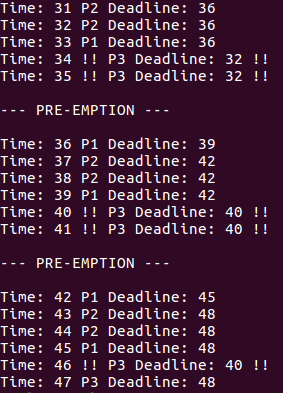
**node->prio = node->prio + node->p;**

**prioInsertNode(&readyQueue, node);**

**Question 7.** (2 marks)

This is the output of my RMS scheduler with missed deadlines:





**Question 8.** (8 marks)

CPU utilization (using the utilization formula) is:

Here is my Criticial Instance Analysis (CIA) of the 3 processes:

**For Process 1,**

**1 ocess thatilization by counting cycles is the same as that using the formula in this case.ng work, assuming that**

**For Process 2,**

**For Process 3,**

Based on CPU utilization and CIA we have missed deadlines because:

**The CPU Utilization is higher than 1 (1.04). This is impossible (not schedulable) as it would mean that the CPU is used more intensively than its 100% capacity. In other words, at least one process would miss its deadline.**

**From the Critical Instance Analysis (CIA) of the three processes, Process 1 and Process 2 would meet their deadlines as their response times, S, are lower than their deadlines, P (by default).**

**However, for Process 3, its response time, S, is higher than its deadline, P. This means that Process 3 is not schedulable in this scenario and will hence miss its deadline.**

TOTAL: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / 70