**CS2106 Operating Systems**

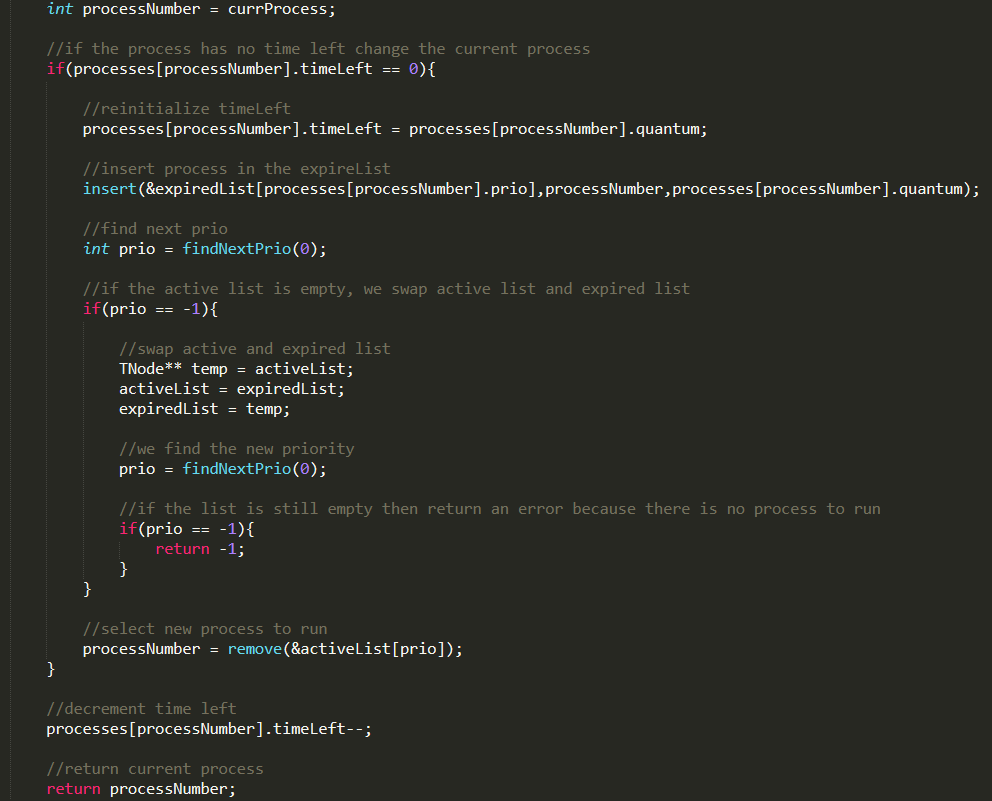
**2017/18 Semester II**

**Term Assignment Answer Book**

|  |  |
| --- | --- |
| STUDENT NUMBER 1:  A0161321X | STUDENT NAME 1:  Tran Nhat Quang |
| STUDENT NUMBER 2:  A0158438U | STUDENT NAME 2:  Nicholas Ang |
| STUDENT NUMBER 3:  A0174721H | STUDENT NAME 3:  Loïc Vandenberghe |

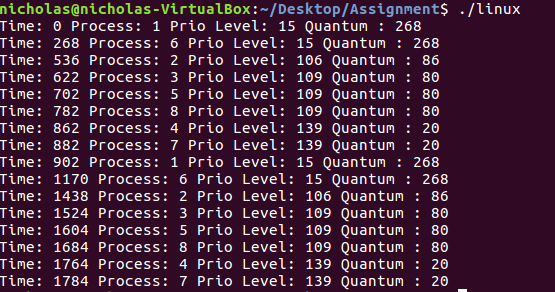
**Question 1** (15 marks)

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



At the start, check if the process has any time left to run. If it has, decrement its timeLeft value by one and return its processNumber since it is still running at that time. If that process does not have any time left to run, we will have to switch to the next process. To do so, we must reset the timeLeft for the current process and add it into the expired list and then search for the next process that has highest priority to run. If we can find such a process then we will remove it from the active list and run it. But if there are no more processes in the active list we will have to swap the active and expired list since all the processes are in the expired list now. After doing so, there will be processes in the active list and we find the process with the highest priority to run. If both active and expired lists are empty then an error is thrown as there are no more processes to run.

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



**Question 2** (5 marks)

One advantage of using array of queues:

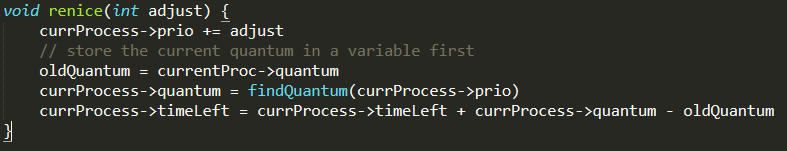
The process of adding more processes into the queue is a lot faster if we use an array of queues. Since each array represents one priority level, adding a new process with a known priority level would only take O(1) time as compared to adding new processes into a single priority queue which would take O(logn) time where n is the number or current processes in the queue.

One disadvantage of using array of queues:

There is more overhead in terms of the various pointers, queues and arrays needed to implement this method as compared to just a single priority queue. There will be no need to reassign pointers in the other implementation. Furthermore, using 140 arrays of queues times two takes up relatively a lot more space as compared to the single priority queue.

**Question 3** (5 marks)

My pseudocode for “renice” is shown below:



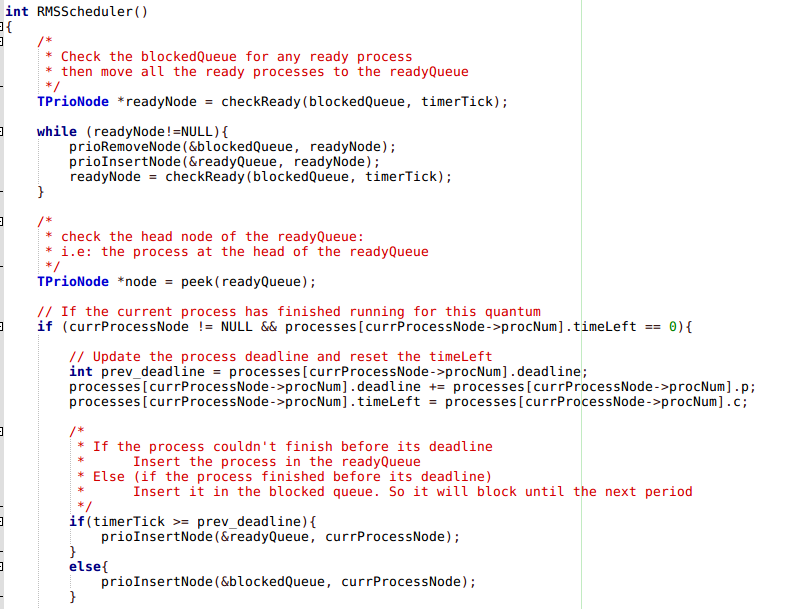
We can change the values of the attributes of the process by re-assigning values to them. We can first increase the priority level of the process, then the quantum since the quantum is dependent on the priority level, and finally the timeLeft since that is dependant on the quantum. Since the current process has already been removed from the active list before it is allowed to run, we can just let it complete its execution and add it into the expired list as usual. It will be added to the appropriate index of the array since its priority has already been updated.

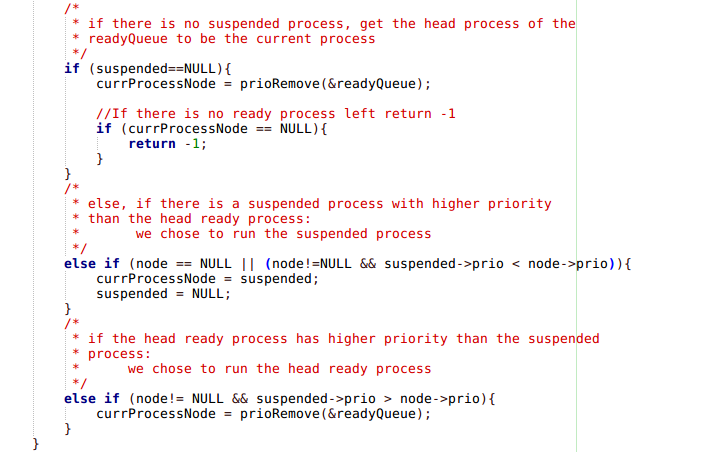


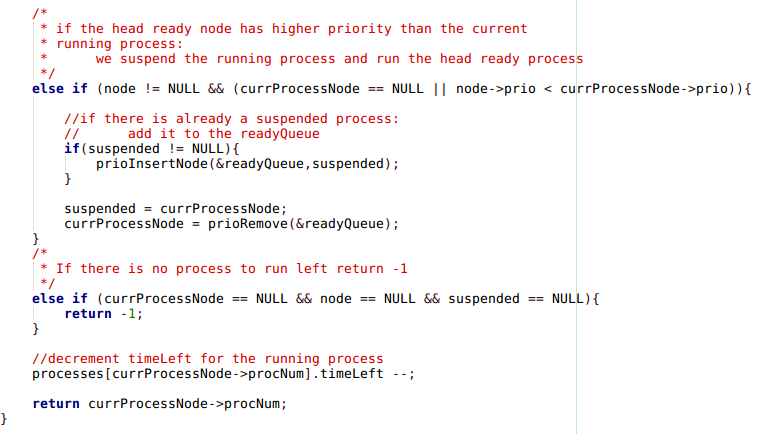
**Question 4.** (20 marks)

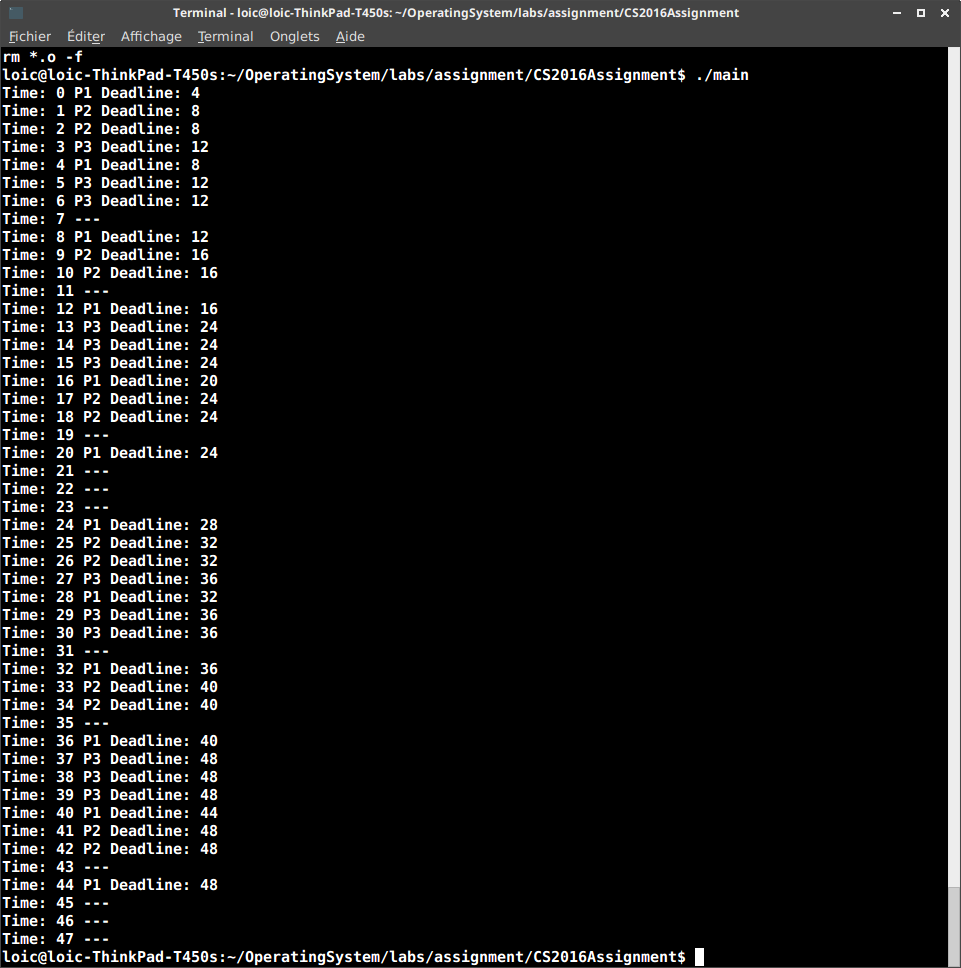
My code and explanation for the RMS scheduler is shown below:

(The explanation is shown in the comments)

Here is my screenshot of my RMS scheduler running:

****

****

****

**Question 5.** (5 marks)

CPU Utilization using the formula is:

1/4+2/8+3/12 = ¾ = 0.75

CPU Utilization by counting cycles is:

36 cycles execute a process and 12 cycles does not run anything. So:

CPU Utilization is:36/48 = 0.75

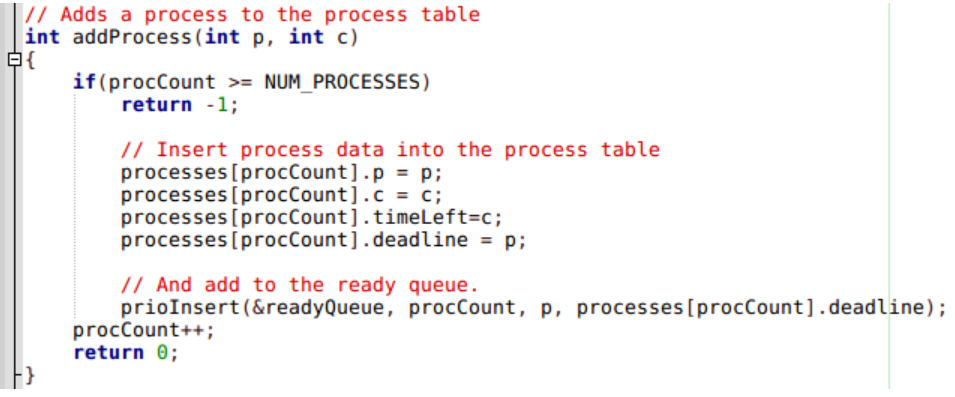
The two results are equal. This is why:

All processes finish execution before their deadlines, so for each of their respective periods they will use c cycles over p, the sum of which gives us the formula of CPU Utilization.

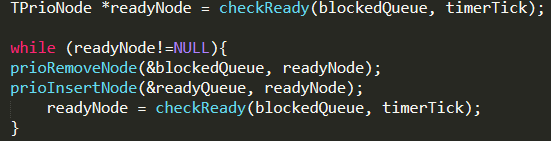
**Question 6.** (10 marks)

My modifications to turn this into an EDF scheduler are:

To turn the RMS scheduler into an EDF scheduler, we can continue using the ready and blocked queues and the suspended node. However, we will have to change what affects the priority level of the processes. The priority will be based on deadlines instead of periods in this case. Hence, the priority queues will be effectively ordered by earlier deadline in this case.

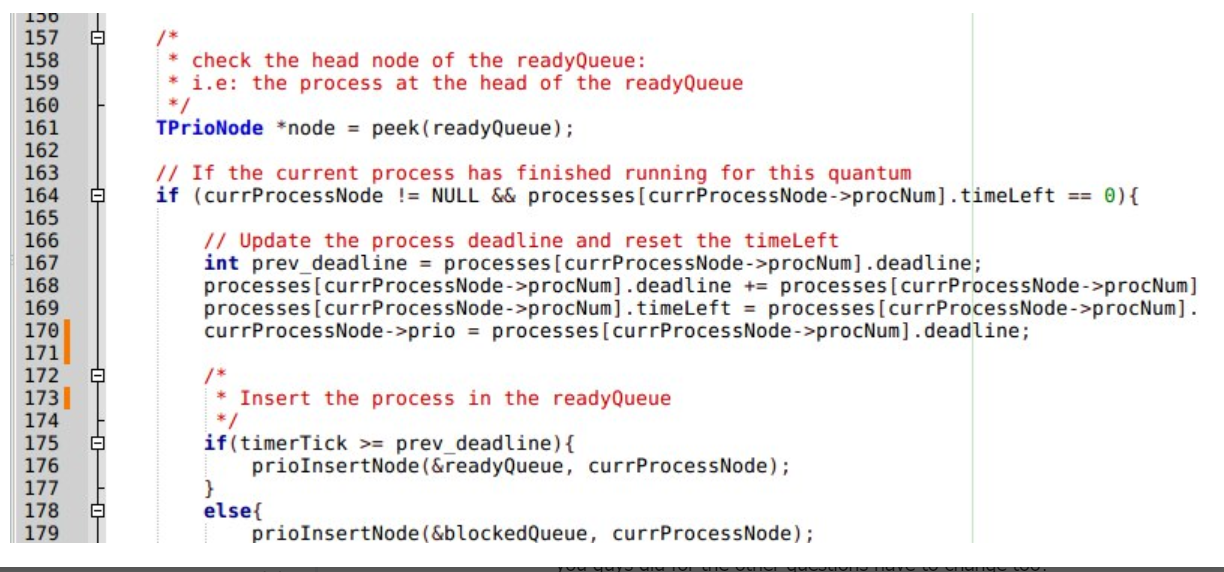


We can keep these lines of code as they will place any processes that are ready to run into the ready queue at the start.



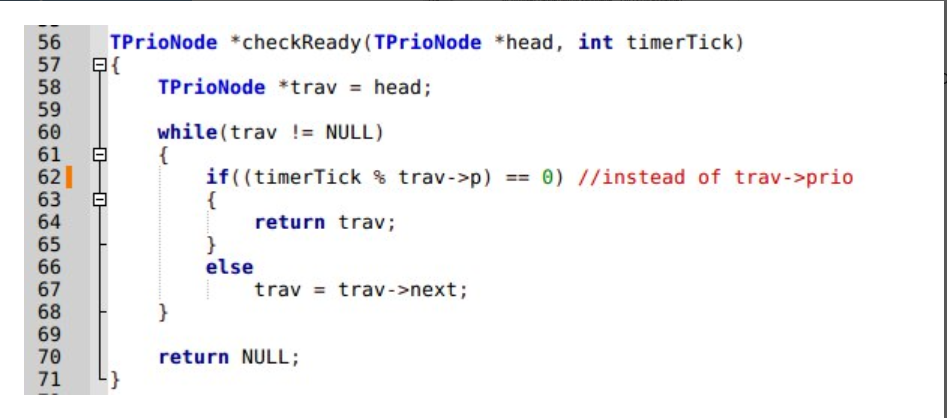
From there, we retrieve and run the first process in the ready queue since the queue is ordered by earliest deadline first. At every timer tick, since all the checks and conditions are already in place in the RMS scheduler to run the process with the highest priority first, our EDF scheduler works since we have already changed the priority levels to be dependant on the deadline of the processes.

However, we will have to update the priority of the process after every time it has completed its allocated quantum for that particular period.



In line 168, we update the deadline of the current process and in line 170 we updated the priority according to this new deadline of the process.

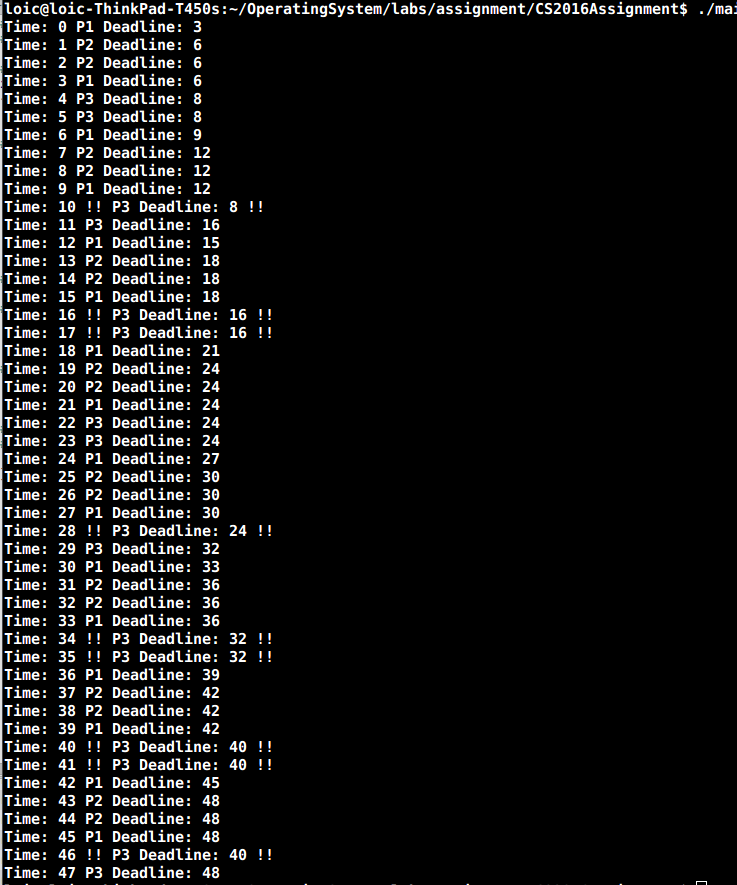
However, we must change the checkReady function of the priority queue as well since it uses the priority level, which used to be based on period, to check for readiness. Since our priority levels are now based on the constantly changing deadlines, the old implementation would not work. We changed the function to calculate based on the period of the process instead.



With these few changes, we will be able to convert our RMS scheduler into an EDF scheduler.

**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:

1/3 +2/6 +3/8 = 25/24 = ~1.04

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



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

the final value S3,F is equal to 11 which is larger than P3 = 8 so P3 will miss is deadline.

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