**CS2106 Operating Systems**

**2017/18 Semester II**

**Term Assignment Answer Book**

|  |  |
| --- | --- |
| STUDENT NUMBER 1:  A0168721B | STUDENT NAME 1:  Ang Wei Ming |
| STUDENT NUMBER 2:  A0126400Y | STUDENT NAME 2:  Quek Yang Sheng |
| STUDENT NUMBER 3: | STUDENT NAME 3: |

**Question 1** (15 marks)

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

|  |
| --- |
| int linuxScheduler()  {  if(timerTick != 0)  --processes[currProcess].timeLeft;  if(processes[currProcess].timeLeft == 0) {  processes[currProcess].timeLeft = processes[currProcess].quantum;  insert(&expiredList[currPrio], currProcess, processes[currProcess].quantum);  int nextProcNo = findNextPrio(currPrio);  if(nextProcNo < 0) {  printf("\n\*\*\*\*\*\*\* SWAPPED LIST \*\*\*\*\*\*\*\n\n");  std::swap(activeList, expiredList);  }  return remove(&activeList[findNextPrio(currPrio)]);  }    return currProcess;  } |

When a processes reaches the end of its current quantum, it will reset its time left and then insert its pointer into the queue of its priority under the expired list. It then finds the next priority level in the active list that has a process. If it does not have any then it swaps the active list and expired list pointer and then removes the next process under the active list.

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

Time: 0 Process: 1 Prio Level: 15 Quantum : 268

Time: 268 Process: 6 Prio Level: 15 Quantum : 268

Time: 536 Process: 2 Prio Level: 106 Quantum : 86

Time: 622 Process: 3 Prio Level: 109 Quantum : 80

Time: 702 Process: 5 Prio Level: 109 Quantum : 80

Time: 782 Process: 8 Prio Level: 109 Quantum : 80

Time: 862 Process: 4 Prio Level: 139 Quantum : 20

Time: 882 Process: 7 Prio Level: 139 Quantum : 20

\*\*\*\*\*\*\* SWAPPED LIST \*\*\*\*\*\*\*

Time: 902 Process: 1 Prio Level: 15 Quantum : 268

Time: 1170 Process: 6 Prio Level: 15 Quantum : 268

Time: 1438 Process: 2 Prio Level: 106 Quantum : 86

Time: 1524 Process: 3 Prio Level: 109 Quantum : 80

Time: 1604 Process: 5 Prio Level: 109 Quantum : 80

Time: 1684 Process: 8 Prio Level: 109 Quantum : 80

Time: 1764 Process: 4 Prio Level: 139 Quantum : 20

Time: 1784 Process: 7 Prio Level: 139 Quantum : 20

**Question 2** (5 marks)

When choosing the next process to schedule, if the next process is of the same priority as the current process, it will just use the pointer to get the next process. If the next process is of different priority, it will iterate through the array of queues to find the next process. Both of which is O(1). However, in the priority queue implementation to get the next process to schedule, we have to extract the top element of the priority queue which is O(logN).

One advantage of using array of queues: In the scenario where there is a large number of processes running in the system, the queue implementation will be faster since O(logN) is larger than O(1).

One disadvantage of using array of queues: In the scenario where there is only a few processes running in the system, O(logN) will be faster since the queue implementation will have to iterate through the array of 140 queues to find the next process to schedule.

**Question 3** (5 marks)

My pseudocode for “renice” is shown below:

Look for the process under the process table and offset the value of prio with adjust, while keeping the value within 0 to 139. The LinuxScheduler will insert the process under the new priority’s queue after it finishes the current quantum cycle.

**Question 4.** (20 marks)

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

|  |
| --- |
| int RMSScheduler()  {  if(timerTick != 0)  --processes[currProcess].timeLeft;  TPrioNode \*node = checkReady(blockedQueue, timerTick);  while(node != NULL){  prioRemoveNode(&blockedQueue, node);  prioInsertNode(&readyQueue, node);  node = checkReady(blockedQueue, timerTick);  }  if(currProcess == -1) {  if(readyQueue == NULL)  return currProcess;  else {  currProcessNode = prioRemove(&readyQueue);  currProcess = currProcessNode->procNum;  }  }  if(processes[currProcess].timeLeft == 0) {  processes[currProcess].timeLeft = processes[currProcess].c;  processes[currProcess].deadline += processes[currProcess].p;  prioInsertNode(&blockedQueue, currProcessNode);  if(readyQueue == NULL && suspended == NULL){  currProcessNode = NULL;  return -1;  } else if (suspended != NULL){  if(readyQueue != NULL){  if(readyQueue->prio < suspended->prio){  currProcessNode = prioRemove(&readyQueue);  return currProcessNode->procNum;  }  }  currProcessNode = prioRemove(&suspended);  return currProcessNode->procNum;  }  currProcessNode = prioRemove(&readyQueue);  return currProcessNode->procNum;  } else {  if(readyQueue != NULL && readyQueue->prio < currProcessNode->prio) {  printf("\n====== Pre-Emption ======\n\n");  prioInsertNode(&suspended, currProcessNode);  currProcessNode = prioRemove(&readyQueue);  return currProcessNode->procNum;  }  return currProcess;  }    return 0;  } |

Here is my screenshot of my RMS scheduler running:

Time: 0 P1 Deadline: 4

Time: 1 P2 Deadline: 8

Time: 2 P2 Deadline: 8

Time: 3 P3 Deadline: 12

====== Pre-Emption ======

Time: 4 P1 Deadline: 8

Time: 5 P3 Deadline: 12

Time: 6 P3 Deadline: 12

Time: 7 ---

Time: 8 P1 Deadline: 12

Time: 9 P2 Deadline: 16

Time: 10 P2 Deadline: 16

Time: 11 ---

Time: 12 P1 Deadline: 16

Time: 13 P3 Deadline: 24

Time: 14 P3 Deadline: 24

Time: 15 P3 Deadline: 24

Time: 16 P1 Deadline: 20

Time: 17 P2 Deadline: 24

Time: 18 P2 Deadline: 24

Time: 19 ---

Time: 20 P1 Deadline: 24

Time: 21 ---

Time: 22 ---

Time: 23 ---

Time: 24 P1 Deadline: 28

Time: 25 P2 Deadline: 32

Time: 26 P2 Deadline: 32

Time: 27 P3 Deadline: 36

====== Pre-Emption ======

Time: 28 P1 Deadline: 32

Time: 29 P3 Deadline: 36

Time: 30 P3 Deadline: 36

Time: 31 ---

Time: 32 P1 Deadline: 36

Time: 33 P2 Deadline: 40

Time: 34 P2 Deadline: 40

Time: 35 ---

Time: 36 P1 Deadline: 40

Time: 37 P3 Deadline: 48

Time: 38 P3 Deadline: 48

Time: 39 P3 Deadline: 48

Time: 40 P1 Deadline: 44

Time: 41 P2 Deadline: 48

Time: 42 P2 Deadline: 48

Time: 43 ---

Time: 44 P1 Deadline: 48

Time: 45 ---

Time: 46 ---

Time: 47 ---

**Question 5.** (5 marks)

CPU Utilization using the formula is: 1/4 + 2/8 + 3/12 = 0.75

CPU Utilization by counting cycles is:

Total CPU: 48

Total idle: 12

Actual CPU: 48 - 12 = 36

36/48 = 0.75

They are the same (choose one). This is why: The formula is a sum of all the process’ CPU time over period, which is the percentage of CPU time each process needs. When we sum the percentage of CPU time each process needs, it will give the percentage of CPU time all process needs. However, a CPU cannot have a utilization of more than 100%. Therefore, if the utilization by counting cycles is below 100%, the percentage of CPU time all process needs will also be the same value.

**Question 6.** (10 marks)

My modifications to turn this into an EDF scheduler are:

Modify prioll.h to add deadline to the TPrioNode struct.

Modify prioll.cpp insert function to sort queue according to node’s deadline.

Under the scheduler function

{

if(timerTick != 0)

--processes[currProcess].timeLeft;

TPrioNode \*node = checkReady(blockedQueue, timerTick);

while(node != NULL){

prioRemoveNode(&blockedQueue, node);

prioInsertNode(&readyQueue, node);

node = checkReady(blockedQueue, timerTick);

}

if(currProcess == -1) {

if(readyQueue == NULL)

return currProcess;

else {

currProcessNode = prioRemove(&readyQueue);

currProcess = currProcessNode->procNum;

return currProcess;

}

}

if(processes[currProcess].timeLeft == 0) {

processes[currProcess].timeLeft = processes[currProcess].c;

processes[currProcess].deadline += processes[currProcess].p;

currProcessNode->deadline = processes[currProcess].deadline;

prioInsertNode(&blockedQueue, currProcessNode);

if(readyQueue == NULL){

currProcessNode = NULL;

return -1;

}

currProcessNode = prioRemove(&readyQueue);

return currProcessNode->procNum;

} else {

return currProcess;

}

Assuming in our case there will be no more incoming new tasks.

Since our EDF scheduler does not pre-empt any processes, we do not need the code for suspending the process. The highlighted part is the code that is modified. When time left is 0 (process finished quantum), it resets the information in the process table, updates the new deadline and inserts into the blocked queue. It then retrieves the next ready process in the queue, which already sorted by earliest deadline. If time left is not 0, it continues running the current process since it still has the earliest deadline.

Sketch modifications and write down pseudocode for how to modify the RMS scheduler you have built into an EDF scheduler. You do not have to build the EDF scheduler, only explain in as much detail as possible how you can convert your RMS scheduler into an EDF scheduler.

**Question 7.** (2 marks)

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

Time: 0 P1 Deadline: 3

Time: 1 P2 Deadline: 6

Time: 2 P2 Deadline: 6

Time: 3 P1 Deadline: 6

Time: 4 P3 Deadline: 8

Time: 5 P3 Deadline: 8

====== Pre-Emption ======

Time: 6 P1 Deadline: 9

Time: 7 P2 Deadline: 12

Time: 8 P2 Deadline: 12

Time: 9 P1 Deadline: 12

Time: 10 !! P3 Deadline: 8 !!

Time: 11 ---

Time: 12 P1 Deadline: 15

Time: 13 P2 Deadline: 18

Time: 14 P2 Deadline: 18

Time: 15 P1 Deadline: 18

Time: 16 !! P3 Deadline: 16 !!

Time: 17 !! P3 Deadline: 16 !!

====== Pre-Emption ======

Time: 18 P1 Deadline: 21

Time: 19 P2 Deadline: 24

Time: 20 P2 Deadline: 24

Time: 21 P1 Deadline: 24

Time: 22 !! P3 Deadline: 16 !!

Time: 23 ---

Time: 24 P1 Deadline: 27

Time: 25 P2 Deadline: 30

Time: 26 P2 Deadline: 30

Time: 27 P1 Deadline: 30

Time: 28 !! P3 Deadline: 24 !!

Time: 29 !! P3 Deadline: 24 !!

====== Pre-Emption ======

Time: 30 P1 Deadline: 33

Time: 31 P2 Deadline: 36

Time: 32 P2 Deadline: 36

Time: 33 P1 Deadline: 36

Time: 34 !! P3 Deadline: 24 !!

Time: 35 ---

Time: 36 P1 Deadline: 39

Time: 37 P2 Deadline: 42

Time: 38 P2 Deadline: 42

Time: 39 P1 Deadline: 42

Time: 40 !! P3 Deadline: 32 !!

Time: 41 !! P3 Deadline: 32 !!

====== Pre-Emption ======

Time: 42 P1 Deadline: 45

Time: 43 P2 Deadline: 48

Time: 44 P2 Deadline: 48

Time: 45 P1 Deadline: 48

Time: 46 !! P3 Deadline: 32 !!

Time: 47 ---

At time 11, the scheduler did not choose P3 to run due to the way the checkReady function is implemented. Though P3 is supposed to be ready at time 11, checkReady function checks whether a process is ready by use the current timer tick to mod with the period. At time 8, P3 is still under the suspended state which is why the checkReady cannot update P3 from blocked to ready. Therefore P3 is unable to run at time 11 because it is not ready.

**Question 8.** (8 marks)

CPU utilization (using the utilization formula) is: 1/3 + 2/6 + 3/8 = 1.04166

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

The S value for task 3 can be seen converging towards 11

S30 > 8

Which misses the deadline.

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