Min Kyaw - Section 05 - 018182136

Donovan Lee - Section 06 - 016741645

Lab 4 Report

In this lab we are given a CPU_Scheduler file in which we implement the three scheduling algorithms of FCFS, Priority, and Round Robin. Since we are given complementary files and text input files, we only designed the source files of the three scheduling algorithms.

In our design for **FCFS**, we have a global int TID we set equal to 1 to increment the task number in our add method. We also have a struct node pointer to be used in our add method and our schedule to traverse. In our add method, we allocate space for a Task and add the name, TID, priority, and burst to it. If the head is NULL, we allocate space for the top of the empty node and set the head task equal to the created task. Else, we would then insert the new Task. In our schedule method, we just do a simple traverse of the head with the help of the traverse method already given to us and also go into a while loop to run the program and print out the output.

In our design for **Priority**, we have a global variable TID that is set to 1 to label all node tasks created. Also created a global struct node that will contain all our node task as well. Then we created our own void **add**() that will set all the parameters inside of it equal to the struct Task. Then we check if our node head has been filled, if not then we put data into it, and we keep looping our void **add**() until all tasks have been added into the global node. In our int **max**() function it is used to check if the global node head has a max priority set where the higher the number is the bigger the priority is and set that to a local variable max. We create a temp struct variable and loop through the temp node to find the max priority. Then we loop through the head node to see if a priority matches with the max that was found previously. We call the **run**() function then set the head->task->priority to 0 so the head can find a new max without having to call the same head node. In our void **schedule**(), we set a for loop that calls **max**() that basically just finds the max priority.

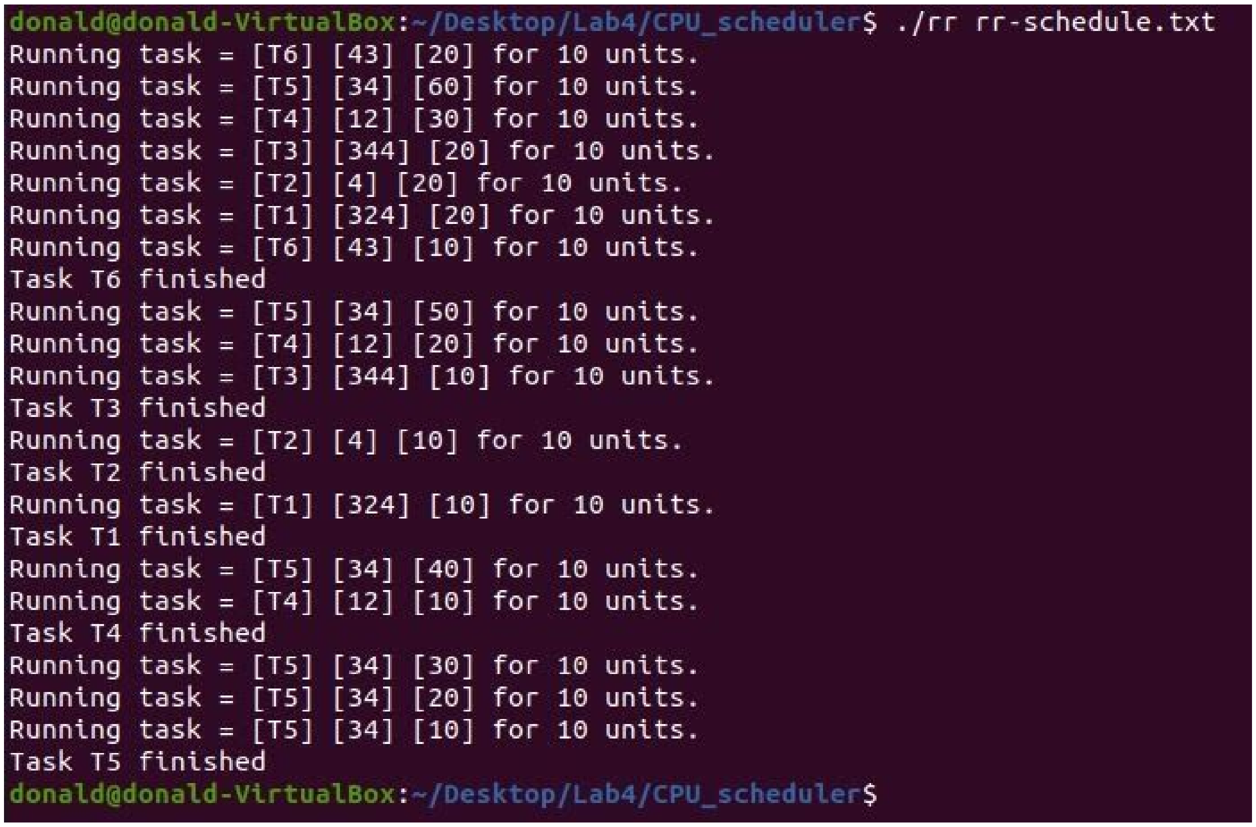
In our design for **RR**, we have a global variable TID that is set to 1 to label all node tasks created. Also created a global struct node that will contain all our node tasks as well. Then we created our own void **add**() that will set all the parameters inside of it equal to the struct Task. Then we check if our node head has been filled, if not then we put data into it, and we keep looping our void **add**() until all tasks have been added into the global node. In our **schedule**() function we set a temp node duplicate of the global head node, set a boolean x to true, and have a counter called max set to 1. Then we created a while loop that uses x that only breaks when all task have been completed. The first if statement checks if the head that we will iterate through is null. If it is then we set the head node to the temp node to reset the loop of it. The next if statement now checks if the head burst is greater than or equal to 1, if it is then we use the **run**() function and subtract its burst. Then we check if the burst is 0, if it is then we add one to max and check if max is equal to TID to break the loop by setting x to false. If it is not 0 then we go to the next node. The next if statement is to check if the burst is greater than 0 and less than or equal to 9 so we can know to subtract the burst by itself. Thus we would automatically know that the task is completed and add one to max. Then we check if max is equal to TID if not we go to the next head. The last if statement is to check if the burst is 0 because we want to skip the burst that we have already completed. We would go to the next head, then check if max equals TID. The else statement is to just print that all task are done and set x to false to exit the loop.

Contributions:

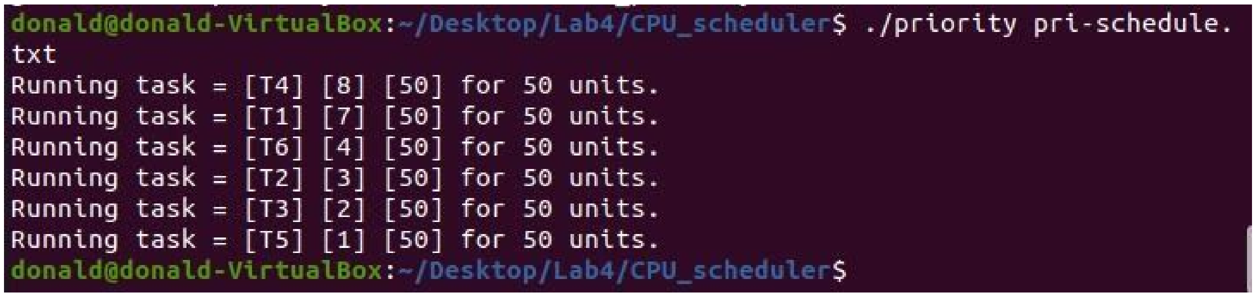
Donovan Lee: Typed the code

Minh Kyaw: Helped with the logic of the code and set up lab report

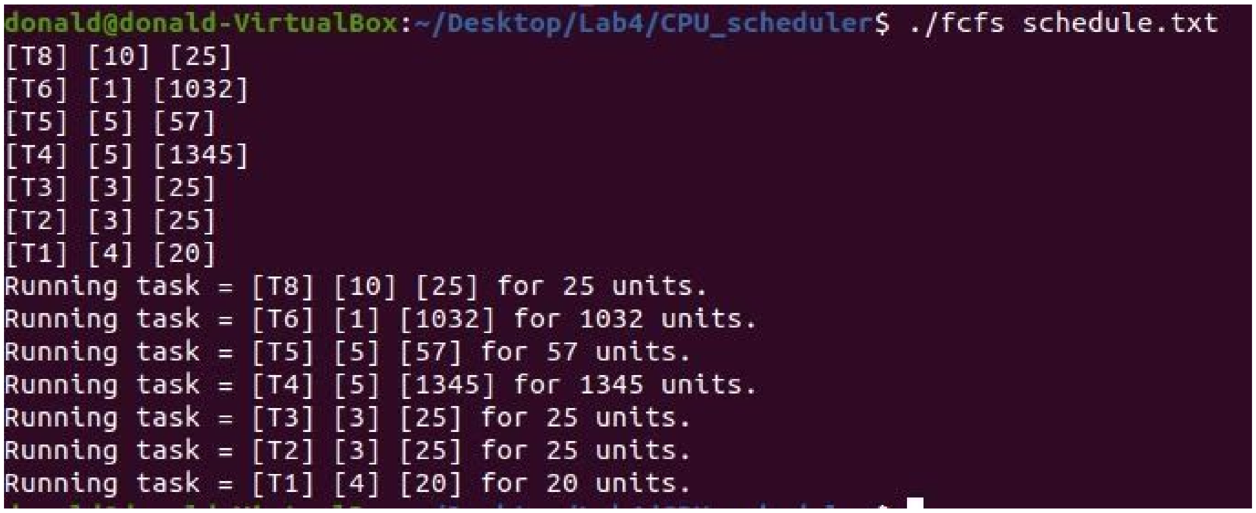
Code Compiled:



```
donald@donald-VirtualBox:~/Desktop/Lab4/CPU_scheduler$ ./rr rr-schedule.txt
Running task = [T6] [43] [20] for 10 units.
Running task = [T5] [34] [60] for 10 units.
Running task = [T4] [12] [30] for 10 units.
Running task = [T3] [344] [20] for 10 units.
Running task = [T2] [4] [20] for 10 units.
Running task = [T1] [324] [20] for 10 units.
Running task = [T6] [43] [10] for 10 units.
Task T6 finished
Running task = [T5] [34] [50] for 10 units.
Running task = [T4] [12] [20] for 10 units.
Running task = [T3] [344] [10] for 10 units.
Task T3 finished
Running task = [T2] [4] [10] for 10 units.
Task T2 finished
Running task = [T1] [324] [10] for 10 units.
Task T1 finished
Running task = [T5] [34] [40] for 10 units.
Running task = [T4] [12] [10] for 10 units.
Task T4 finished
Running task = [T5] [34] [30] for 10 units.
Running task = [T5] [34] [20] for 10 units.
Running task = [T5] [34] [10] for 10 units.
Task T5 finished
donald@donald-VirtualBox:~/Desktop/Lab4/CPU_scheduler$
```



```
donald@donald-VirtualBox:~/Desktop/Lab4/CPU_scheduler$ ./priority pri-schedule.
txt
Running task = [T4] [8] [50] for 50 units.
Running task = [T1] [7] [50] for 50 units.
Running task = [T6] [4] [50] for 50 units.
Running task = [T2] [3] [50] for 50 units.
Running task = [T3] [2] [50] for 50 units.
Running task = [T5] [1] [50] for 50 units.
donald@donald-VirtualBox:~/Desktop/Lab4/CPU_scheduler$
```



```
donald@donald-VirtualBox:~/Desktop/Lab4/CPU_scheduler$ ./fcfs schedule.txt
[T8] [10] [25]
[T6] [1] [1032]
[T5] [5] [57]
[T4] [5] [1345]
[T3] [3] [25]
[T2] [3] [25]
[T1] [4] [20]
Running task = [T8] [10] [25] for 25 units.
Running task = [T6] [1] [1032] for 1032 units.
Running task = [T5] [5] [57] for 57 units.
Running task = [T4] [5] [1345] for 1345 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T2] [3] [25] for 25 units.
Running task = [T1] [4] [20] for 20 units.
```