Name: Vishal Ramane
Roll.No: 2019430010

Experiment No.2

Aim: Implementation of Red-Black trees and its various operations.

Problem Statement:
Application of Red-Black Tree as Completely-Fair Scheduler in Linux Operating System.

Objective:

- Understand Red-Black Tree Data-structure operations.
- Understand the Algorithm behind Completely-Fair Scheduler in Linux Operating System.
- Define and Implement Node Structure for Processes in Red-Black Tree.
- Implement Class Red-Black Tree and supporting operations as member functions of the class, with Scheduler algorithm as one of the member function.

Methodology:

I. The data structure used for the **scheduling algorithm** is a red-black tree and the nodes are indexed by processor "execution time" in nanoseconds.

II. A "maximum execution time" is also calculated for each process. This time is based upon the idea that an "ideal processor" would equally share processing power amongst all processes,defined as the time the process has been waiting to run, divided by the total number of processes,

III. When the scheduler is invoked to run a new process, the operation of the scheduler is as follows:

   1. The **leftmost node** of the scheduling tree is chosen (as it will have the lowest spent execution time), and sent for execution.
   2. If the process simply completes execution, it is **removed** from the system and scheduling tree.
   3. If the process reaches its **maximum execution** time or is otherwise stopped (voluntarily or via interrupt) it is **reinserted** into the scheduling tree based on its new spent execution time.
   4. The new leftmost node will then be selected from the tree, **repeating** the iteration.
   5. If the process spends a lot of its time sleeping, then its spent time value is low and it automatically gets the priority boost when it finally needs it. Hence such tasks do not get less processor time than the tasks that are constantly running.

IV. Complexity :

   1. Space : O(n)
   2. Time :   search = insert = delete : O (log n)

Name: Vishal Ramane
Roll.No: 2019430010

Experiment No.2

Implementation:

I. *class Node* has following attributes defined:
   i. *data* (execution time)
   ii. *color*(Red/Black)
   iii. *left* (left child)
   iv. *right* (Right child)
   v. *parent*
   vi. *pID* (Process ID)

II. *class Red-Black Tree* which uses Node class in instantiation has following member functions defined:

i. *minimum(self, node)*	: (find leftmost node)
ii. *maximum(self, node)*	: (find rightmost node)
iii. *left_rotate(self, x)*	: (left-rotate about node x)
iv. *right_rotate(self, x)*	: (left-rotate about node x)
v. insert(self, key, pid=0)	: (insert operation in RBT)
vi. delete_node(self, data)	: (delete operation in RBT)
vii. *pretty_print(self)*	: (to print RBT on terminal)
viii. cfscheduler(self, max_exectime)	: (to invoke Scheduler)

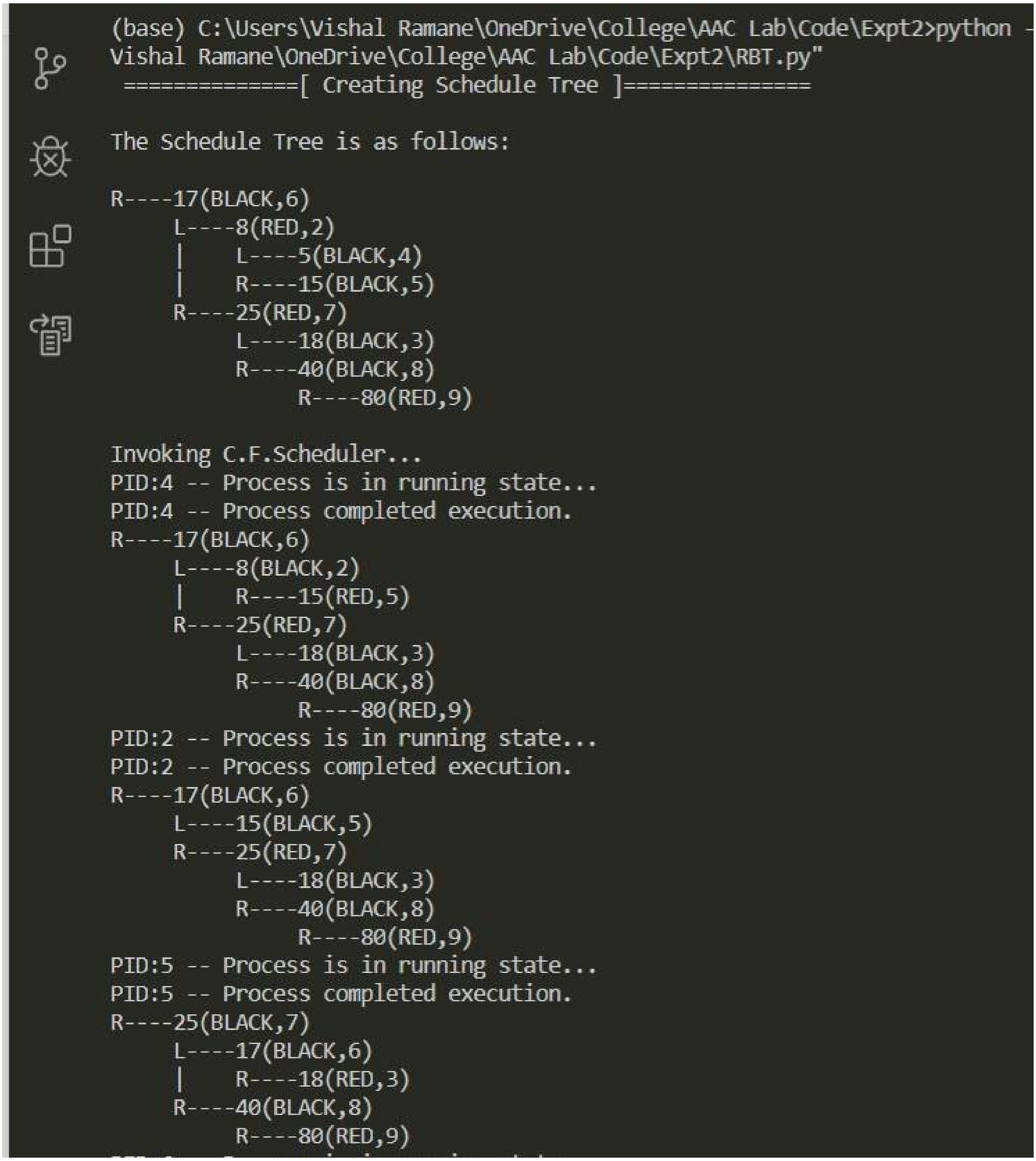
III. And we define following helper functions to support some of above operations:
   i. __fix_insert(self, k)
   ii. __print_helper(self, node, indent, last)
   iii. *__delete_node_helper*

IV. Scheduler Algorithm (some modification):
   i. Max-execution time on processor = (rightmost node/ leftmost node) for the given Schedule tree.
   ii. If process execution time <= max-execution time:
         Process execution complete.
   iii. Else :
      new execution time = difference(max-execution time,process execution time) and re-insert the process into schedule tree.

Name: Vishal Ramane
Roll.No: 2019430010

Experiment No.2

Results:



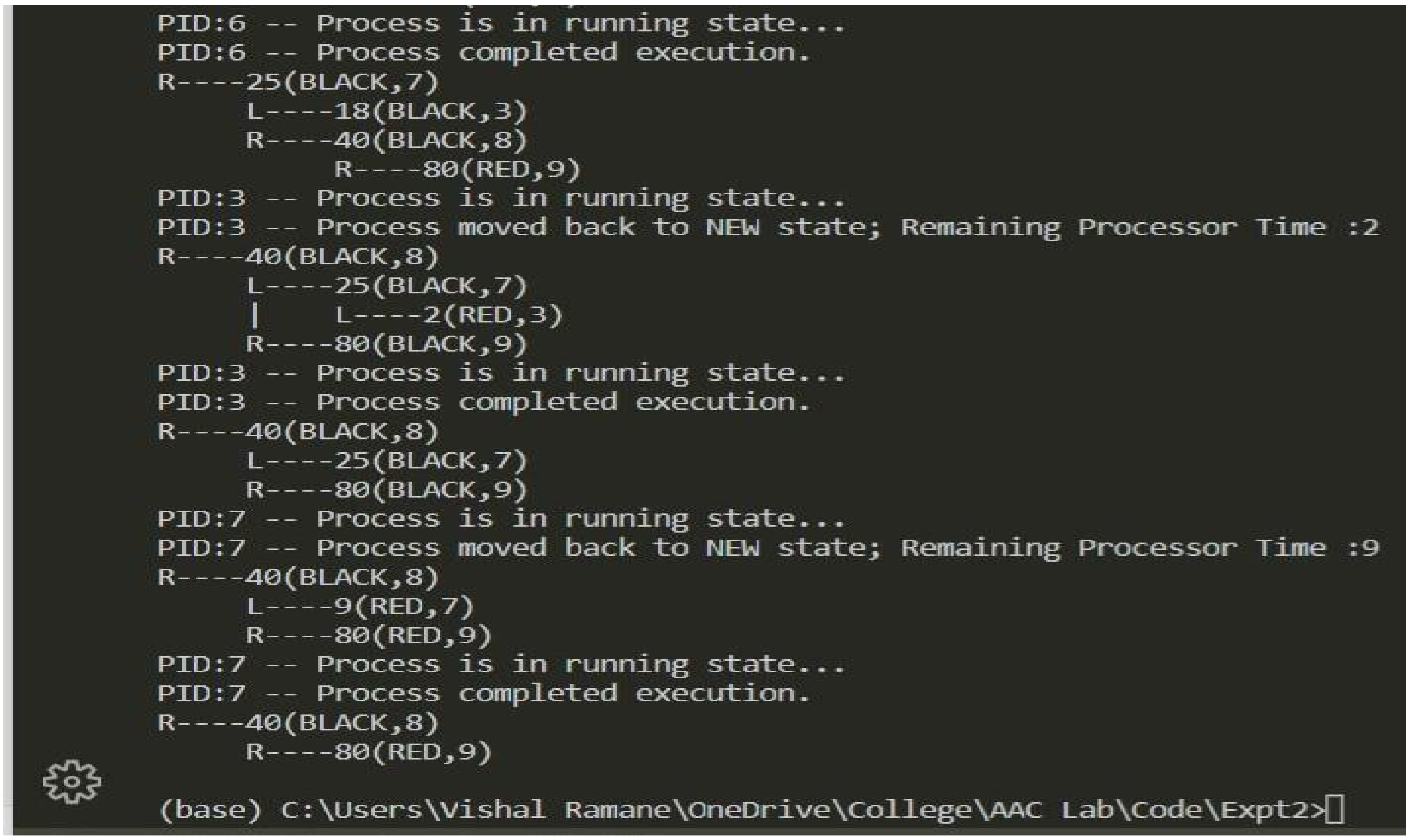
```
(base) C:\Users\Vishal Ramane\OneDrive\College\AAC Lab\Code\Expt2>python -
Vishal Ramane\OneDrive\College\AAC Lab\Code\Expt2\RBT.py"
 ==============[ Creating Schedule Tree ]===============

The Schedule Tree is as follows:

R----17(BLACK,6)
     L----8(RED,2)
     |    L----5(BLACK,4)
     |    R----15(BLACK,5)
     R----25(RED,7)
          L----18(BLACK,3)
          R----40(BLACK,8)
               R----80(RED,9)

Invoking C.F.Scheduler...
PID:4 -- Process is in running state...
PID:4 -- Process completed execution.
R----17(BLACK,6)
     L----8(BLACK,2)
     |    R----15(RED,5)
     R----25(RED,7)
          L----18(BLACK,3)
          R----40(BLACK,8)
               R----80(RED,9)
PID:2 -- Process is in running state...
PID:2 -- Process completed execution.
R----17(BLACK,6)
     L----15(BLACK,5)
     R----25(RED,7)
          L----18(BLACK,3)
          R----40(BLACK,8)
               R----80(RED,9)
PID:5 -- Process is in running state...
PID:5 -- Process completed execution.
R----25(BLACK,7)
     L----17(BLACK,6)
     |    R----18(RED,3)
     R----40(BLACK,8)
          R----80(RED,9)
```

Experiment No.2



```
PID:6 -- Process is in running state...
PID:6 -- Process completed execution.
R----25(BLACK,7)
     L----18(BLACK,3)
     R----40(BLACK,8)
          R----80(RED,9)
PID:3 -- Process is in running state...
PID:3 -- Process moved back to NEW state; Remaining Processor Time :2
R----40(BLACK,8)
     L----25(BLACK,7)
     |    L----2(RED,3)
     R----80(BLACK,9)
PID:3 -- Process is in running state...
PID:3 -- Process completed execution.
R----40(BLACK,8)
     L----25(BLACK,7)
     R----80(BLACK,9)
PID:7 -- Process is in running state...
PID:7 -- Process moved back to NEW state; Remaining Processor Time :9
R----40(BLACK,8)
     L----9(RED,7)
     R----80(RED,9)
PID:7 -- Process is in running state...
PID:7 -- Process completed execution.
R----40(BLACK,8)
     R----80(RED,9)

(base) C:\Users\Vishal Ramane\OneDrive\College\AAC Lab\Code\Expt2>
```

In above experiment Scheduler was invoked 9 times.

Conclusions:
Thus we have successfully implemented Red-Black Tree in-terms of Completely Fair Scheduler.