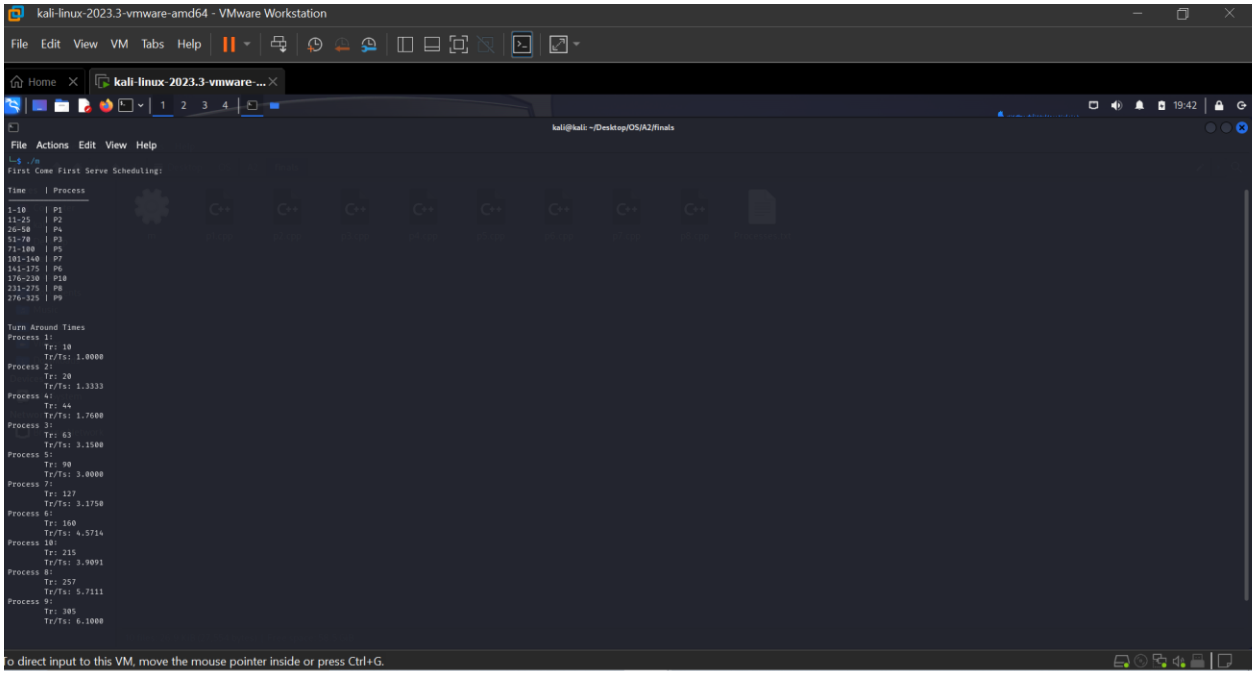
OPERATING SYSTEMS
FALL 2023
ASSIGNMENT 2: Scheduling codes

Group members:
Minam Faisal (21i-1901)
Momenah Saif (21i-1909)

Part -1 (FCFS)

Output: -


kali-linux-2023.3-vmware-amd64 - VMware Workstation
File Edit View VM Tabs Help
Home
kali-linux-2023.3-vmware-...
19:42
kali@kali: ~/Desktop/OS/A2/finals
File Actions Edit View Help
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

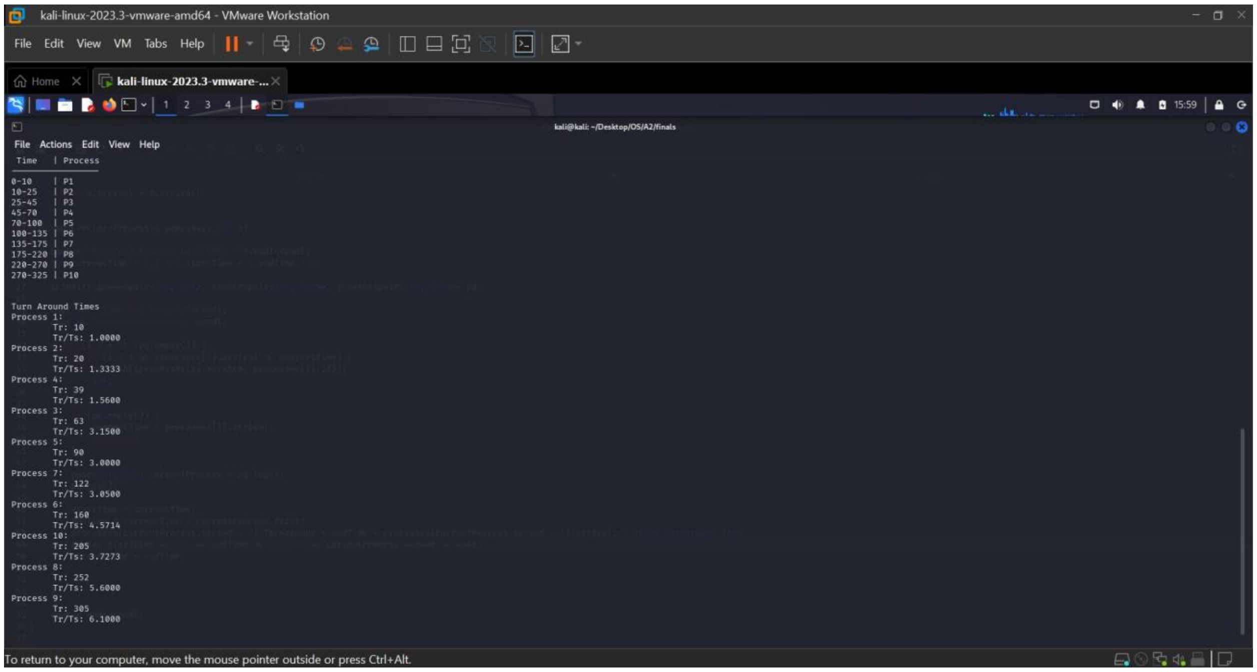
```
└─$ ./m
First Come First Serve Scheduling:

Time    | Process
_________________
1-10    | P1
11-25   | P2
26-50   | P4
51-70   | P3
71-100  | P5
101-140 | P7
141-175 | P6
176-230 | P10
231-275 | P8
276-325 | P9


Turn Around Times
Process 1:
        Tr: 10
        Tr/Ts: 1.0000
Process 2:
        Tr: 20
        Tr/Ts: 1.3333
Process 4:
        Tr: 44
        Tr/Ts: 1.7600
Process 3:
        Tr: 63
        Tr/Ts: 3.1500
Process 5:
        Tr: 90
        Tr/Ts: 3.0000
Process 7:
        Tr: 127
        Tr/Ts: 3.1750
Process 6:
        Tr: 160
        Tr/Ts: 4.5714
Process 10:
        Tr: 215
        Tr/Ts: 3.9091
Process 8:
        Tr: 257
        Tr/Ts: 5.7111
Process 9:
        Tr: 305
        Tr/Ts: 6.1000
```

Explanation:

The algorithm works by iterating over the vector named as processes and starting each process at the current time. The end time of the process is calculated by adding its service time to the current time. The start and end time of the process are then printed in table form. The current time is then updated to the end time of the process. To calculate Turnaround time, at the end of every iteration the end time is subtracted by arrival time and stored in that process turnaround time. The loop runs like this till all processes have ended.

Part -2 (SPN)

Output: -


kali-linux-2023.3-vmware-amd64 - VMware Workstation
File Edit View VM Tabs Help
Home
kali-linux-2023.3-vmware-...
kali@kali: ~/Desktop/OS/A2/finals
File Actions Edit View Help
To return to your computer, move the mouse pointer outside or press Ctrl+Alt.

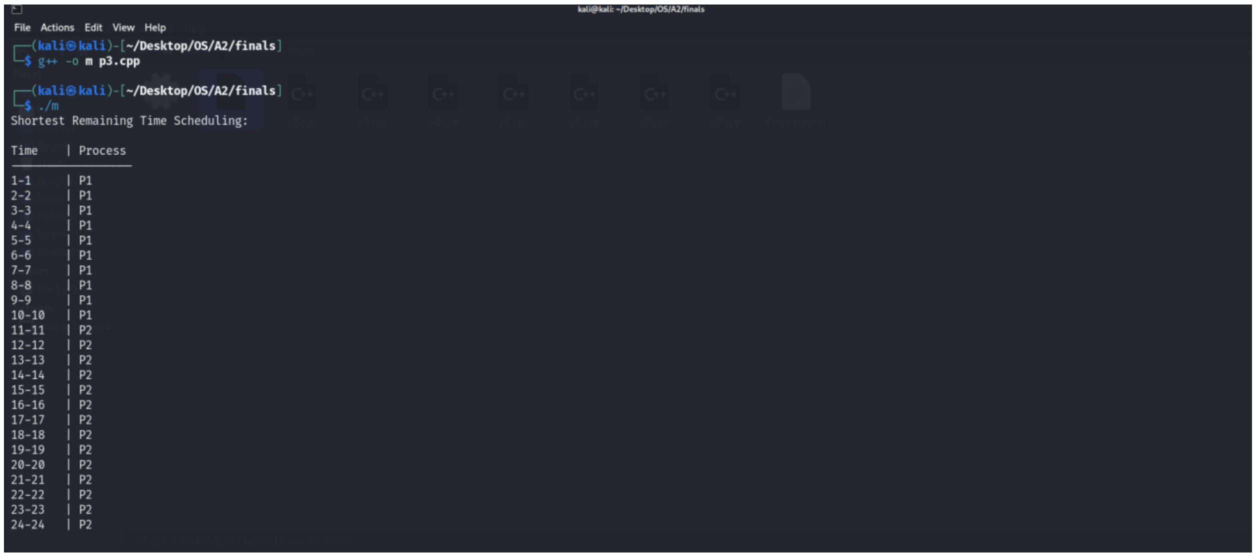
```
 Time    | Process
─────────────────
0-10     | P1
10-25    | P2
25-45    | P3
45-70    | P4
70-100   | P5
100-135  | P6
135-175  | P7
175-220  | P8
220-270  | P9
270-325  | P10

Turn Around Times
Process 1:
        Tr: 10
        Tr/Ts: 1.0000
Process 2:
        Tr: 20
        Tr/Ts: 1.3333
Process 4:
        Tr: 39
        Tr/Ts: 1.5600
Process 3:
        Tr: 63
        Tr/Ts: 3.1500
Process 5:
        Tr: 90
        Tr/Ts: 3.0000
Process 7:
        Tr: 122
        Tr/Ts: 3.0500
Process 6:
        Tr: 160
        Tr/Ts: 4.5714
Process 10:
        Tr: 205
        Tr/Ts: 3.7273
Process 8:
        Tr: 252
        Tr/Ts: 5.6000
Process 9:
        Tr: 305
        Tr/Ts: 6.1000
```

Explanation:

The way the algorithm operates is by keeping track of processes in a priority queue according to their service times. At the front of the queue is always the procedure with the shortest service time. The algorithm plans for the process at the head of the queue to execute at each time step. The procedure continues until it is finished or until a new one with a shorter process time shows up. The new process is then scheduled to execute and the existing one is preempted.

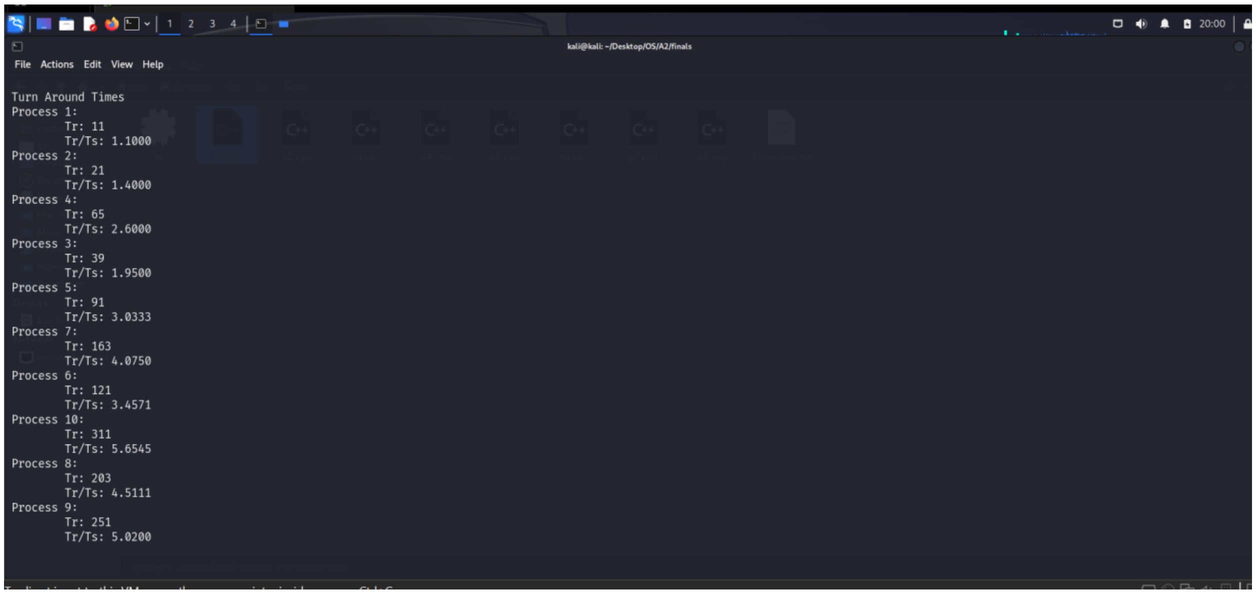
Part -3 (SRT)

Output: -



kali@kali: ~/Desktop/OS/A2/finals
File Actions Edit View Help

```
┌──(kali㉿kali)-[~/Desktop/OS/A2/finals]
└─$ g++ -o m p3.cpp

┌──(kali㉿kali)-[~/Desktop/OS/A2/finals]
└─$ ./m
Shortest Remaining Time Scheduling:

Time    | Process
─────────────────
1-1     | P1
2-2     | P1
3-3     | P1
4-4     | P1
5-5     | P1
6-6     | P1
7-7     | P1
8-8     | P1
9-9     | P1
10-10   | P1
11-11   | P2
12-12   | P2
13-13   | P2
14-14   | P2
15-15   | P2
16-16   | P2
17-17   | P2
18-18   | P2
19-19   | P2
20-20   | P2
21-21   | P2
22-22   | P2
23-23   | P2
24-24   | P2
```



1 2 3 4
20:00
kali@kali: ~/Desktop/OS/A2/finals
File Actions Edit View Help

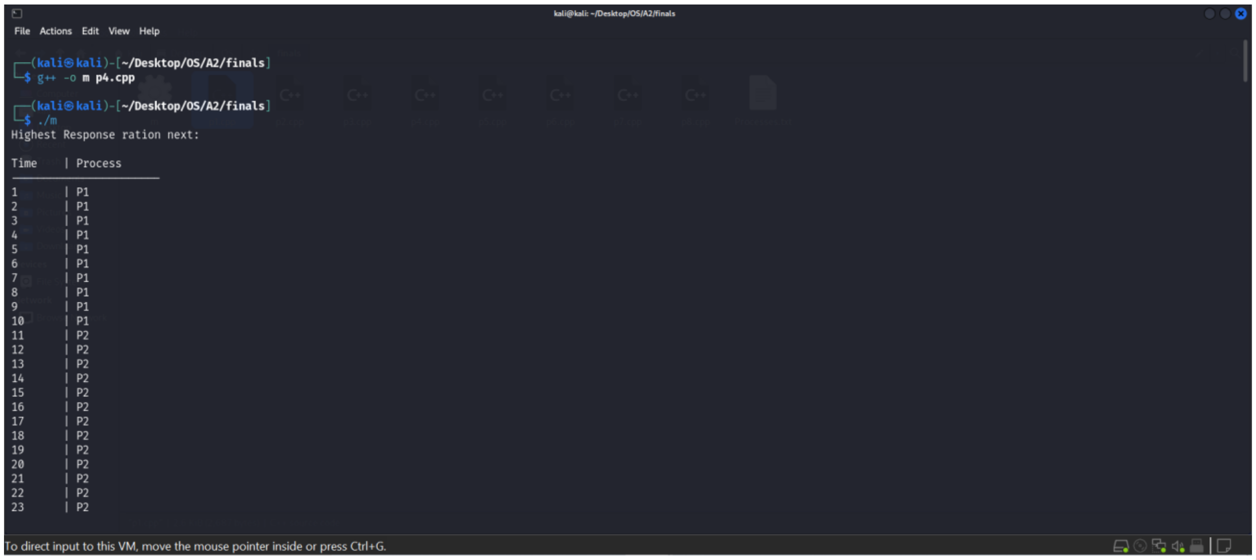
```
Turn Around Times
Process 1:
        Tr: 11
        Tr/Ts: 1.1000
Process 2:
        Tr: 21
        Tr/Ts: 1.4000
Process 4:
        Tr: 65
        Tr/Ts: 2.6000
Process 3:
        Tr: 39
        Tr/Ts: 1.9500
Process 5:
        Tr: 91
        Tr/Ts: 3.0333
Process 7:
        Tr: 163
        Tr/Ts: 4.0750
Process 6:
        Tr: 121
        Tr/Ts: 3.4571
Process 10:
        Tr: 311
        Tr/Ts: 5.6545
Process 8:
        Tr: 203
        Tr/Ts: 4.5111
Process 9:
        Tr: 251
        Tr/Ts: 5.0200
```

Explanation:

To run SRT, the code must keep track of every process that has arrived and has service time left. The algorithm chooses the process from the list that has the least amount of time left and schedules it to execute at each time step. The process continues to run until either another process comes with a lesser remaining time, or until its remaining service time is decreased to 0 or 1.

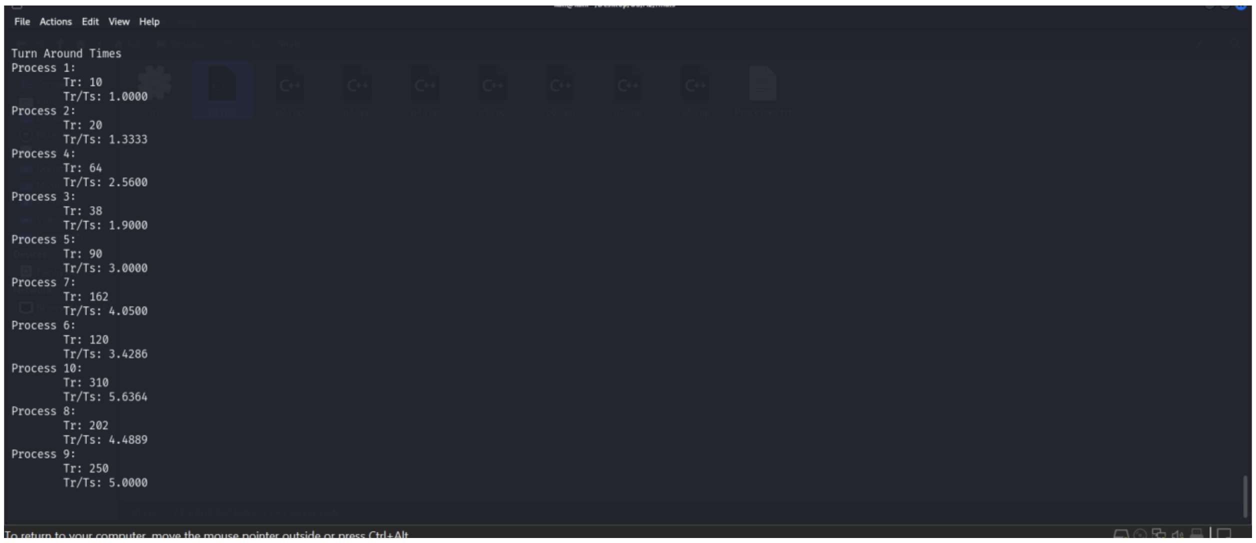
Part -4 (HRRN)

Output: -



```
┌──(kali㉿kali)-[~/Desktop/OS/A2/finals]
└─$ g++ -o m p4.cpp

┌──(kali㉿kali)-[~/Desktop/OS/A2/finals]
└─$ ./m
Highest Response ration next:

Time    | Process
────────────────────
1       | P1
2       | P1
3       | P1
4       | P1
5       | P1
6       | P1
7       | P1
8       | P1
9       | P1
10      | P1
11      | P2
12      | P2
13      | P2
14      | P2
15      | P2
16      | P2
17      | P2
18      | P2
19      | P2
20      | P2
21      | P2
22      | P2
23      | P2
```

kali@kali: ~/Desktop/OS/A2/finals
File Actions Edit View Help
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.



```
Turn Around Times
Process 1:
        Tr: 10
        Tr/Ts: 1.0000
Process 2:
        Tr: 20
        Tr/Ts: 1.3333
Process 4:
        Tr: 64
        Tr/Ts: 2.5600
Process 3:
        Tr: 38
        Tr/Ts: 1.9000
Process 5:
        Tr: 90
        Tr/Ts: 3.0000
Process 7:
        Tr: 162
        Tr/Ts: 4.0500
Process 6:
        Tr: 120
        Tr/Ts: 3.4286
Process 10:
        Tr: 310
        Tr/Ts: 5.6364
Process 8:
        Tr: 202
        Tr/Ts: 4.4889
Process 9:
        Tr: 250
        Tr/Ts: 5.0000
```

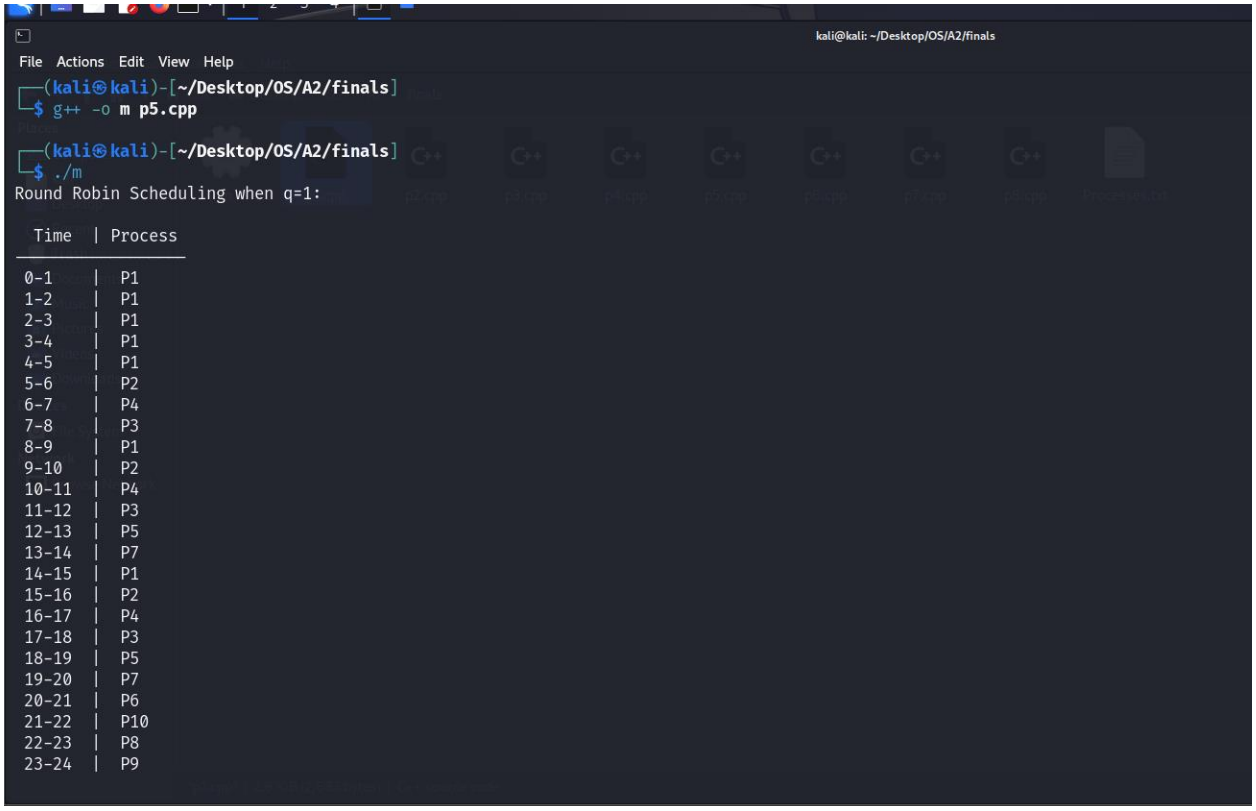
To return to your computer, move the mouse pointer outside or press Ctrl+Alt

Explanation:

The algorithm computes the response ratios of each process as iterates over them. After that, it chooses the process with the best response ratio and sets a time for it to execute. The procedure continues until either all of the processes have finished or its service time drops to zero.

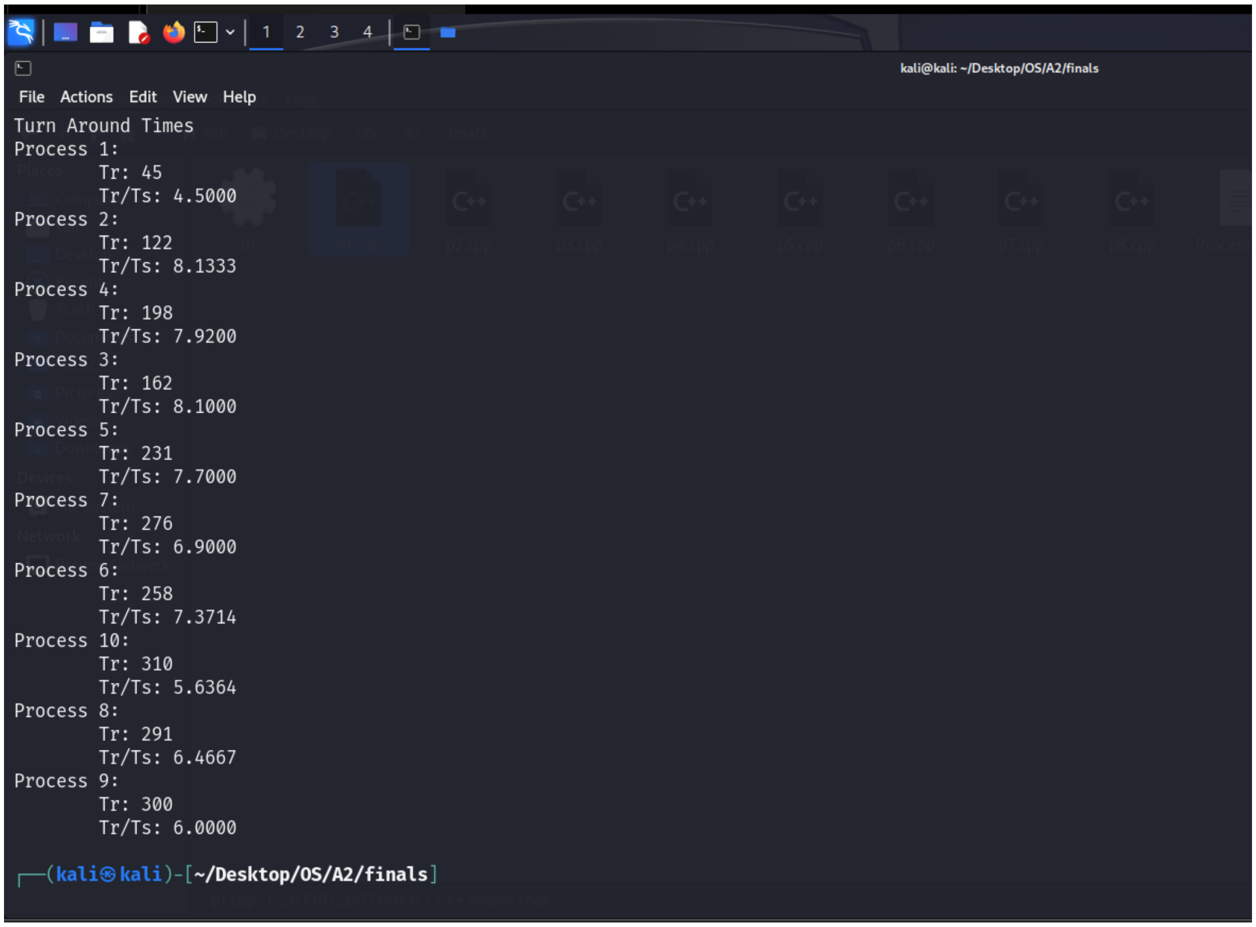
Part -5 (Round Robin (q=1))

Output: -



kali@kali: ~/Desktop/OS/A2/finals
File Actions Edit View Help

```
┌──(kali㉿kali)-[~/Desktop/OS/A2/finals]
└─$ g++ -o m p5.cpp

┌──(kali㉿kali)-[~/Desktop/OS/A2/finals]
└─$ ./m
Round Robin Scheduling when q=1:

  Time  | Process
 ─────────────────
 0-1    |  P1
 1-2    |  P1
 2-3    |  P1
 3-4    |  P1
 4-5    |  P1
 5-6    |  P2
 6-7    |  P4
 7-8    |  P3
 8-9    |  P1
 9-10   |  P2
 10-11  |  P4
 11-12  |  P3
 12-13  |  P5
 13-14  |  P7
 14-15  |  P1
 15-16  |  P2
 16-17  |  P4
 17-18  |  P3
 18-19  |  P5
 19-20  |  P7
 20-21  |  P6
 21-22  |  P10
 22-23  |  P8
 23-24  |  P9
```


kali@kali: ~/Desktop/OS/A2/finals
File Actions Edit View Help

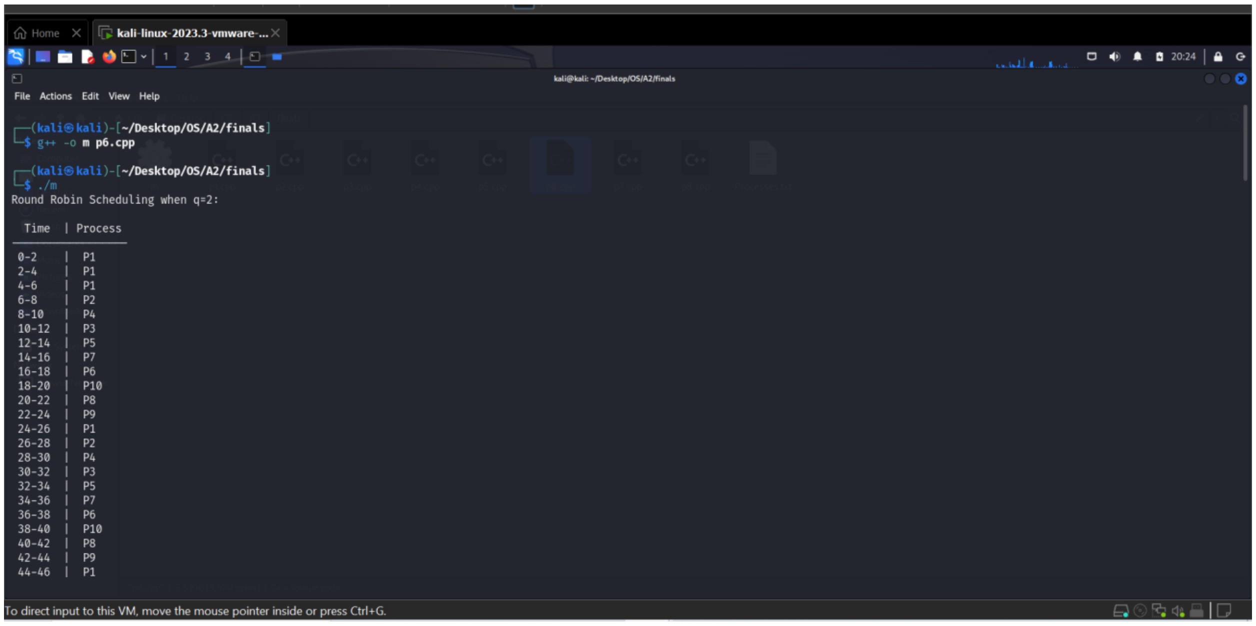
```
Turn Around Times
Process 1:
        Tr: 45
        Tr/Ts: 4.5000
Process 2:
        Tr: 122
        Tr/Ts: 8.1333
Process 4:
        Tr: 198
        Tr/Ts: 7.9200
Process 3:
        Tr: 162
        Tr/Ts: 8.1000
Process 5:
        Tr: 231
        Tr/Ts: 7.7000
Process 7:
        Tr: 276
        Tr/Ts: 6.9000
Process 6:
        Tr: 258
        Tr/Ts: 7.3714
Process 10:
        Tr: 310
        Tr/Ts: 5.6364
Process 8:
        Tr: 291
        Tr/Ts: 6.4667
Process 9:
        Tr: 300
        Tr/Ts: 6.0000

┌──(kali㉿kali)-[~/Desktop/OS/A2/finals]
```

Explanation:

To start, all of the processes that have arrived and are prepared to run are gathered into a queue by the algorithm. After that, it initializes a vector of booleans to track which processes have finished and sets the current time to 0. After that, the algorithm loops through every process in the ready queue. It verifies whether each process has arrived and hasn't been finished. In that case, the algorithm determines whether the operation can be finished inside the time quantum. The algorithm finishes the operation and removes it from the ready queue if it is possible. If not, the algorithm reduces the process's service time by the time quantum and preempts the process. After that, the algorithm adds the preempted process to the back of the ready queue and updates the current time.

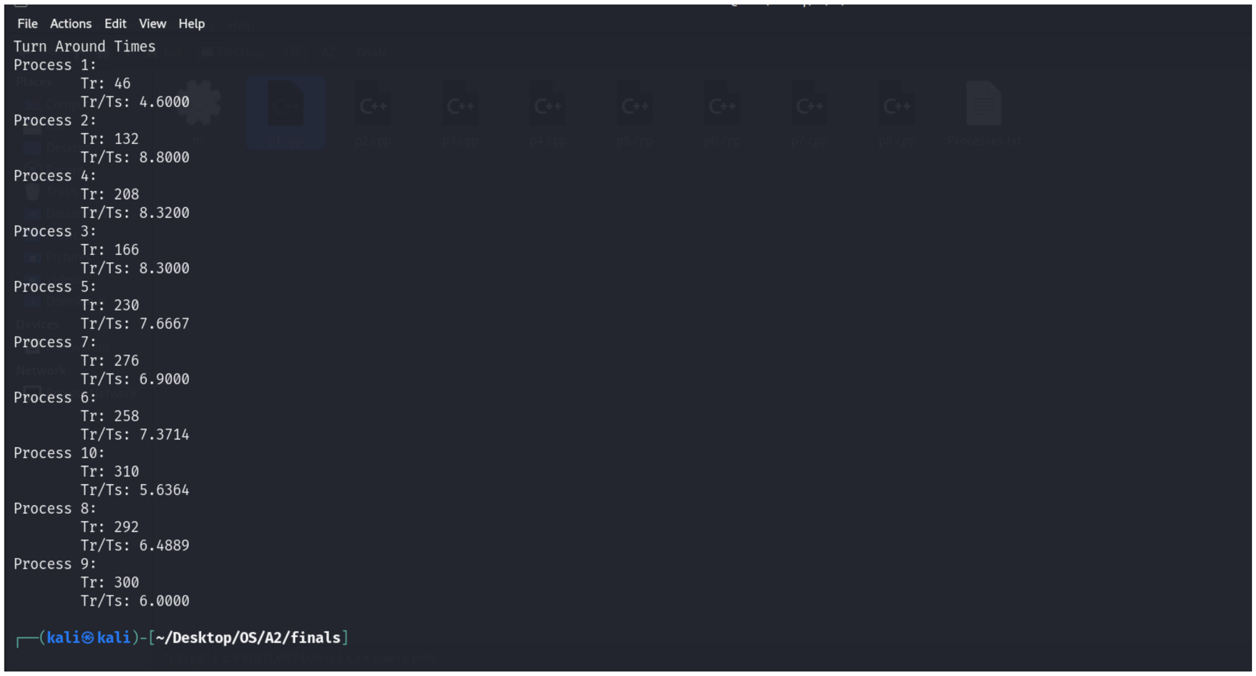
Part -6 (Round Robin(q=2))

Output: -



Home
kali-linux-2023.3-vmware-...
20:24
kali@kali: ~/Desktop/OS/A2/finals
File Actions Edit View Help
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

```
┌──(kali㉿kali)-[~/Desktop/OS/A2/finals]
└─$ g++ -o m p6.cpp

┌──(kali㉿kali)-[~/Desktop/OS/A2/finals]
└─$ ./m
Round Robin Scheduling when q=2:

 Time  | Process
 ────────────────
 0-2   |  P1
 2-4   |  P1
 4-6   |  P1
 6-8   |  P2
 8-10  |  P4
 10-12 |  P3
 12-14 |  P5
 14-16 |  P7
 16-18 |  P6
 18-20 |  P10
 20-22 |  P8
 22-24 |  P9
 24-26 |  P1
 26-28 |  P2
 28-30 |  P4
 30-32 |  P3
 32-34 |  P5
 34-36 |  P7
 36-38 |  P6
 38-40 |  P10
 40-42 |  P8
 42-44 |  P9
 44-46 |  P1
```



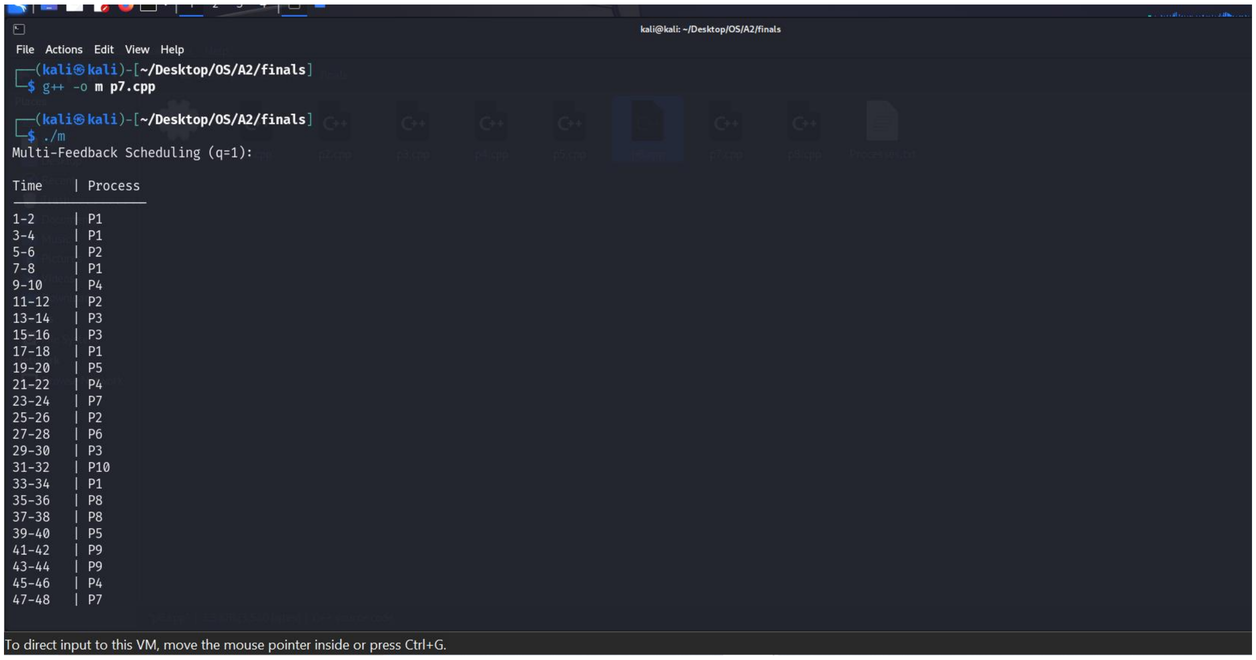
```
Turn Around Times
Process 1:
          Tr: 46
          Tr/Ts: 4.6000
Process 2:
          Tr: 132
          Tr/Ts: 8.8000
Process 4:
          Tr: 208
          Tr/Ts: 8.3200
Process 3:
          Tr: 166
          Tr/Ts: 8.3000
Process 5:
          Tr: 230
          Tr/Ts: 7.6667
Process 7:
          Tr: 276
          Tr/Ts: 6.9000
Process 6:
          Tr: 258
          Tr/Ts: 7.3714
Process 10:
          Tr: 310
          Tr/Ts: 5.6364
Process 8:
          Tr: 292
          Tr/Ts: 6.4889
Process 9:
          Tr: 300
          Tr/Ts: 6.0000

┌──(kali㉿kali)-[~/Desktop/OS/A2/finals]
```

Explanation:

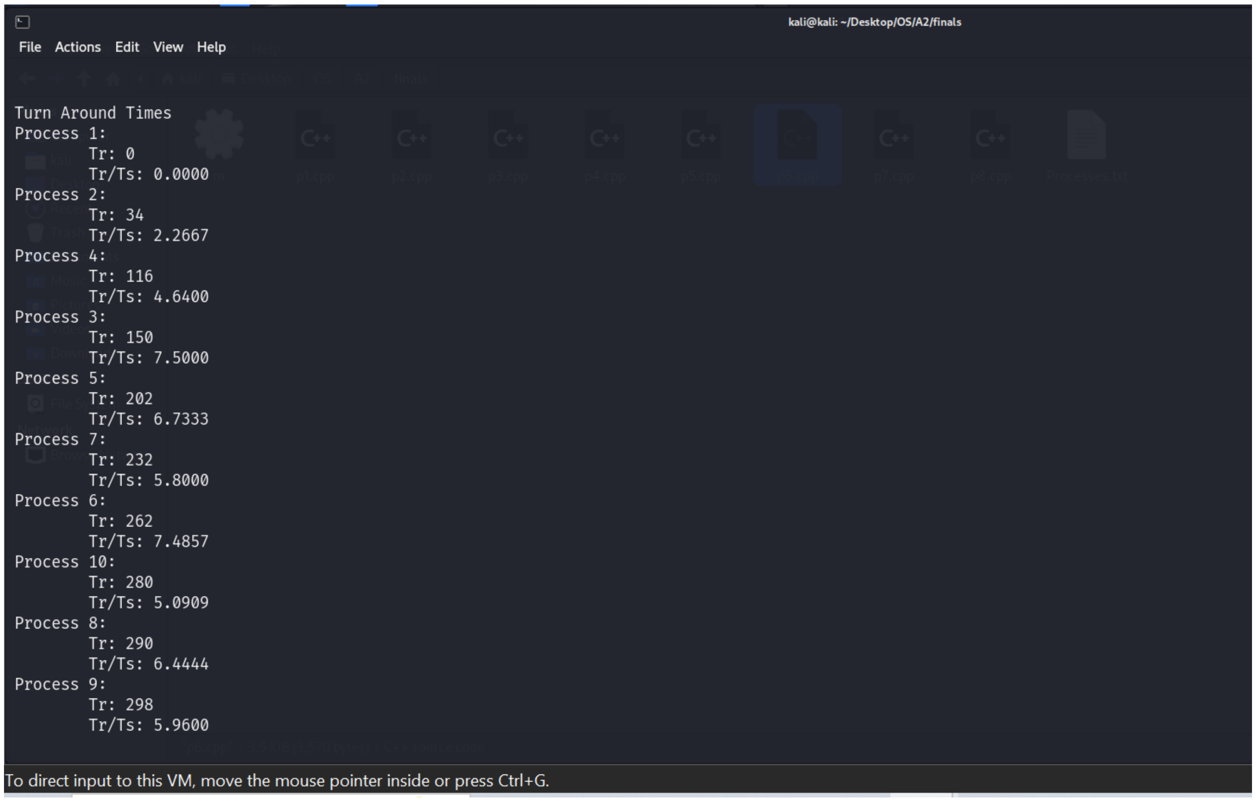
This algorithm works the same as above just the difference is that in main function the time quantum changed to 2 so the output also changed.

Part -7 (Multi level Feedback(q=1))

Output: -



```
kali@kali: ~/Desktop/OS/A2/finals
File  Actions  Edit  View  Help
┌──(kali㉿kali)-[~/Desktop/OS/A2/finals]
└─$ g++ -o m p7.cpp

┌──(kali㉿kali)-[~/Desktop/OS/A2/finals]
└─$ ./m
Multi-Feedback Scheduling (q=1):

Time    | Process
──────────────────
1-2     | P1
3-4     | P1
5-6     | P2
7-8     | P1
9-10    | P4
11-12   | P2
13-14   | P3
15-16   | P3
17-18   | P1
19-20   | P5
21-22   | P4
23-24   | P7
25-26   | P2
27-28   | P6
29-30   | P3
31-32   | P10
33-34   | P1
35-36   | P8
37-38   | P8
39-40   | P5
41-42   | P9
43-44   | P9
45-46   | P4
47-48   | P7
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.
```



kali@kali: ~/Desktop/OS/A2/finals
File Actions Edit View Help

```
Turn Around Times
Process 1:
        Tr: 0
        Tr/Ts: 0.0000
Process 2:
        Tr: 34
        Tr/Ts: 2.2667
Process 4:
        Tr: 116
        Tr/Ts: 4.6400
Process 3:
        Tr: 150
        Tr/Ts: 7.5000
Process 5:
        Tr: 202
        Tr/Ts: 6.7333
Process 7:
        Tr: 232
        Tr/Ts: 5.8000
Process 6:
        Tr: 262
        Tr/Ts: 7.4857
Process 10:
        Tr: 280
        Tr/Ts: 5.0909
Process 8:
        Tr: 290
        Tr/Ts: 6.4444
Process 9:
        Tr: 298
        Tr/Ts: 5.9600
```

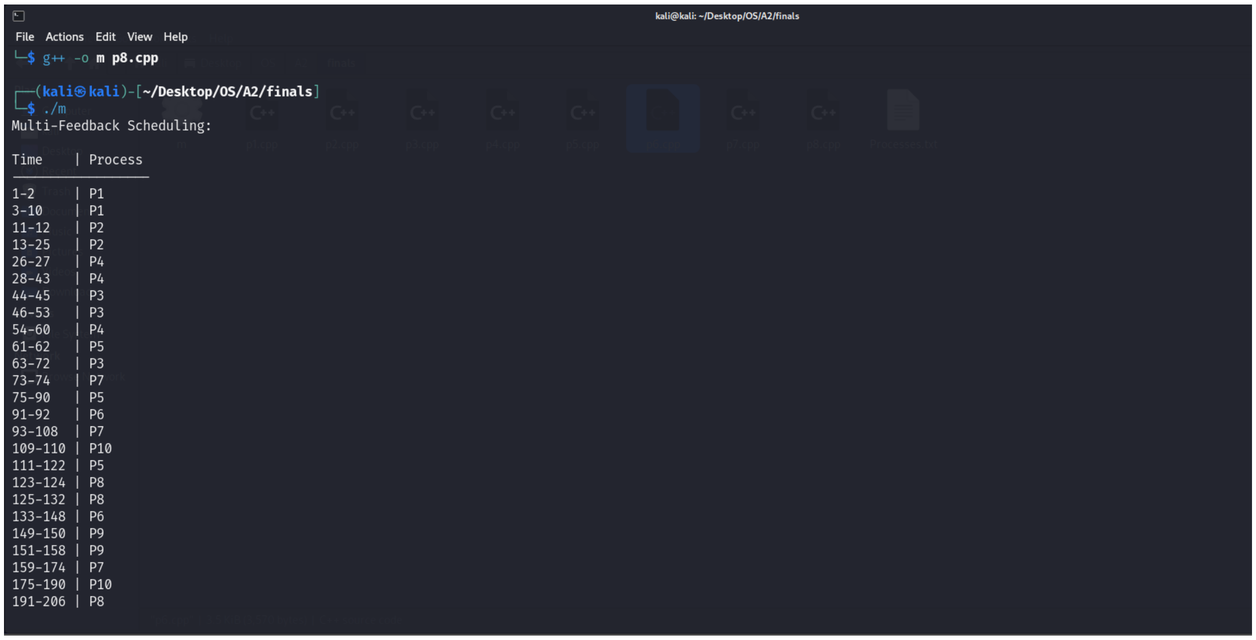
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.

Explanation:

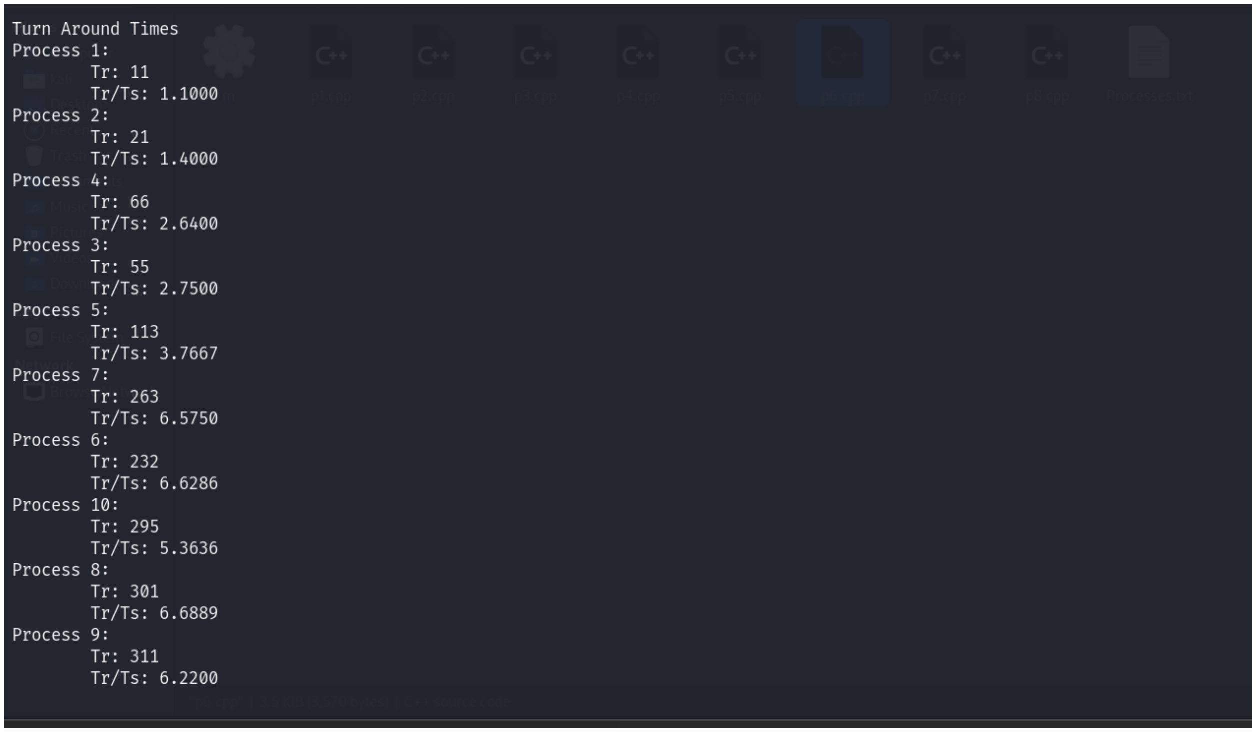
In this algorithm, processes are sorted into several queues according to priority. As they execute, processes are transferred from the highest priority queue to lower priority queues. This makes it more likely that all processes will ultimately be finished, even if some require more computation than others. Initializing a vector of four priority queues is how the code operates. The first queue has the highest priority, while the fourth queue has the lowest priority. Each queue denotes a distinct priority level. After that, the code loops through every process, placing it in the queue according to its priority. After that, a loop that iterates until every queue is empty is entered by the code. The code determines whether the current priority queue is empty at each iteration of the loop. The code moves the process to the front of the queue and schedules it to run if it is not empty. The process continues to run until either its time slice expires or it is finished. The process is advanced to the next lower priority queue if its time slice expires. This guarantees that all processes, regardless of how computationally demanding they are, get an opportunity to execute.

Part -8 (Multi level Feedback(q=2^i))

Output: -


kali@kali: ~/Desktop/OS/A2/finals
File Actions Edit View Help

```
└─$ g++ -o m p8.cpp

┌──(kali㉿kali)-[~/Desktop/OS/A2/finals]
└─$ ./m
Multi-Feedback Scheduling:

Time    | Process
──────────────────
1-2     | P1
3-10    | P1
11-12   | P2
13-25   | P2
26-27   | P4
28-43   | P4
44-45   | P3
46-53   | P3
54-60   | P4
61-62   | P5
63-72   | P3
73-74   | P7
75-90   | P5
91-92   | P6
93-108  | P7
109-110 | P10
111-122 | P5
123-124 | P8
125-132 | P8
133-148 | P6
149-150 | P9
151-158 | P9
159-174 | P7
175-190 | P10
191-206 | P8
```



```
Turn Around Times
Process 1:
          Tr: 11
          Tr/Ts: 1.1000
Process 2:
          Tr: 21
          Tr/Ts: 1.4000
Process 4:
          Tr: 66
          Tr/Ts: 2.6400
Process 3:
          Tr: 55
          Tr/Ts: 2.7500
Process 5:
          Tr: 113
          Tr/Ts: 3.7667
Process 7:
          Tr: 263
          Tr/Ts: 6.5750
Process 6:
          Tr: 232
          Tr/Ts: 6.6286
Process 10:
          Tr: 295
          Tr/Ts: 5.3636
Process 8:
          Tr: 301
          Tr/Ts: 6.6889
Process 9:
          Tr: 311
          Tr/Ts: 6.2200
```

Explanation:

This algorithm is same as above code of q=1. The difference is that a queue named timequantum with priorities 2,4,8,16 are used and code runs according to it.

Common parts of all parts

In all codes the function with scheduling algorithm is changed. Some parts of struct also is changed. Other than that many funtions are same in all codes:

Main:

In main function all the functions in order are called. First of all the the name of the file is stored in the string. Then the function to read file is called and its output is stored in struct vector named processes that is used throughout the code. Next a built in function that comes by adding algorithm library is called with a function named sortByArrivalTime is called. In feedback the processes are also sorted by priority. After the processes are sorted in come codes the service time is stored in remaining time for future use. Next the function with scheduling code is called in which after every run of a process a turnaround time is stored in processes.turnaround. Next in main function the turnaroundTime function is called in which the turnaround time of every process is displayed and Tr/Ts is also calculated and displayed.

readFromFile:

In this function it has basic file reading code but with some amendments. The txt file given has first line that tells what each column represents. The first line is skipped in reading. Next the code takes care of commas and spaces that are present in the file. The service time is written first which is also tackled and the values are put in their respective parts of struct. Other than the feedback codes the priority from the files are not read as they are not needed.

sortByArrivalTime:

In this function the arrival time of each process is simply checked and returned the process with smaller arrival time. The vector is sort by built in function.

sortByPriority

In feedback codes the process are also sort according to priority.

TurnaroundTime:

Turnaround time is stored in scheduling functions so in this function the turnarounds are just displayed and then divided by service time of the process.

--------------------------The End------------------------------