Name : Anupam Kunwar
Reg : 19BCE1369

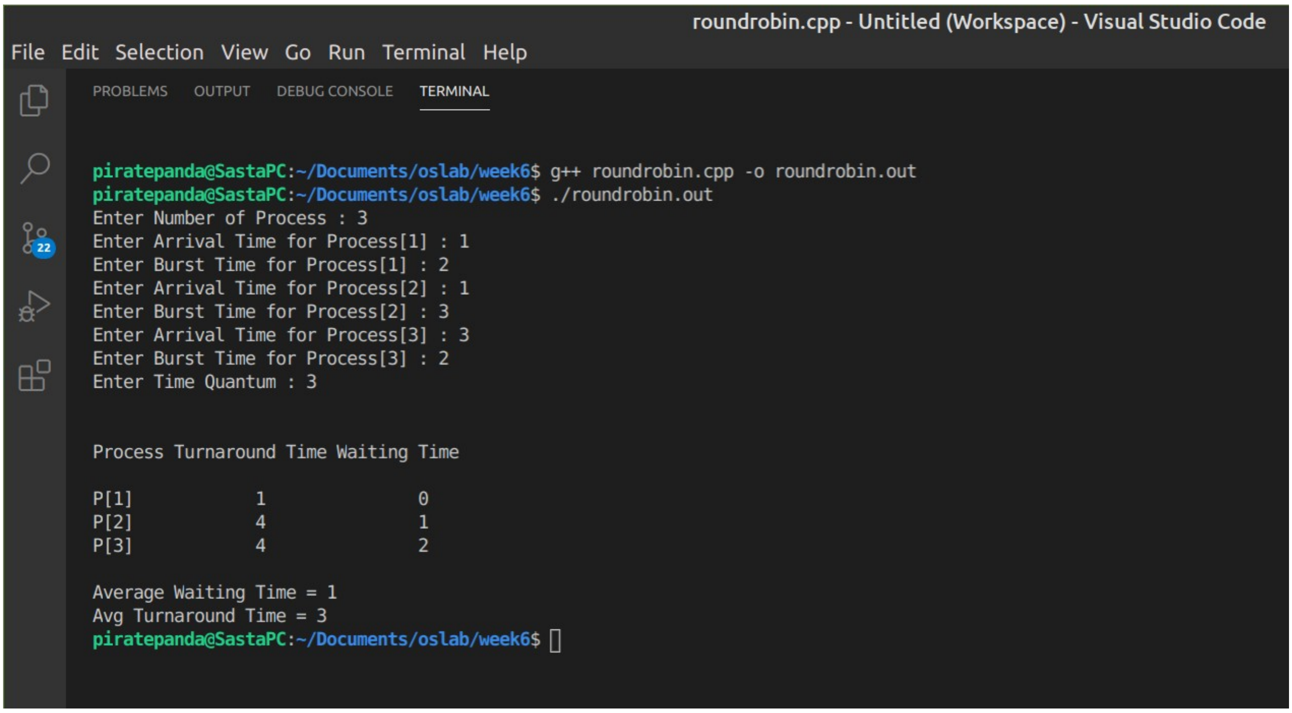
WEEK-6

Q1. <u>Round Robin Scheduling in c++.</u>

Solution :

```
#include<iostream>
using namespace std;
int main()
{
int processes, j, n, time, unfinished, flag = 0, quantum,wt,tat;
int wait_time = 0, turnaround_time = 0, at[10], bt[10], rt[10];
cout << "Enter Number of Process : ";
cin >> n;
unfinished = n;
for (processes = 0; processes < n; processes++)
{
cout << "Enter Arrival Time for Process[" << processes + 1 << "] : ";
cin >> at[processes];
cout << "Enter Burst Time for Process[" << processes + 1 << "] : ";
cin >> bt[processes];
rt[processes] = bt[processes];
}
cout << "Enter Time Quantum : ";
cin >> quantum;
cout << "\n\nProcess\tTurnaround Time\tWaiting Time\n\n";
for (time = 0, processes = 0; unfinished != 0;)
{
if (rt[processes] <= quantum && rt[processes] > 0)
{
time += rt[processes];
rt[processes] = 0;
flag = 1;
}
else if (rt[processes] > 0)
{
rt[processes] -= quantum;
time += quantum;
}
if (rt[processes] == 0 && flag == 1)
{
unfinished--;
wt = time - at[processes] - bt[processes];
if(wt<0)
wt=0;
tat = time-at[processes];
cout << "P[" << processes+1 << "]" << "\t\t" << tat << "\t\t" << wt << endl;
wait_time += wt;
turnaround_time += tat;
flag = 0;
}
if (processes == n - 1)
processes = 0;
else if (at[processes + 1] <= time)
processes++;
else
processes = 0;
}
cout << "\nAverage Waiting Time = " << (wait_time * 1.0 / n) << endl;
cout << "Avg Turnaround Time = " << (turnaround_time * 1.0 / n) << endl;
return 0;
}
```

Output :


roundrobin.cpp - Untitled (Workspace) - Visual Studio Code
File Edit Selection View Go Run Terminal Help
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
22

```
piratepanda@SastaPC:~/Documents/oslab/week6$ g++ roundrobin.cpp -o roundrobin.out
piratepanda@SastaPC:~/Documents/oslab/week6$ ./roundrobin.out
Enter Number of Process : 3
Enter Arrival Time for Process[1] : 1
Enter Burst Time for Process[1] : 2
Enter Arrival Time for Process[2] : 1
Enter Burst Time for Process[2] : 3
Enter Arrival Time for Process[3] : 3
Enter Burst Time for Process[3] : 2
Enter Time Quantum : 3


Process Turnaround Time Waiting Time

P[1]            1               0
P[2]            4               1
P[3]            4               2

Average Waiting Time = 1
Avg Turnaround Time = 3
piratepanda@SastaPC:~/Documents/oslab/week6$ 
```

.

Q2 : Queue Process Scheduling program in c :

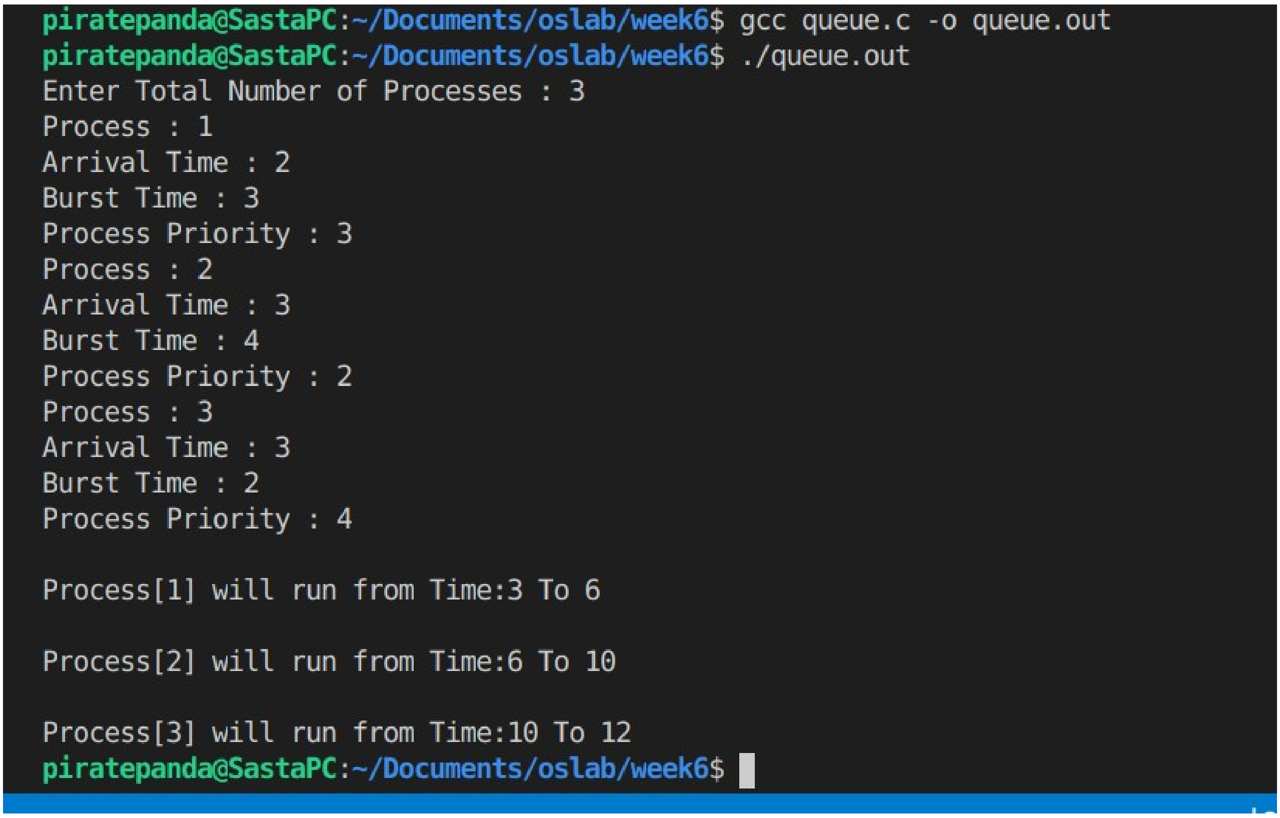
Program :

```
#include <stdio.h>

#define N 10

typedef struct
{
int process_id, arrival_time, burst_time, priority;
int q, ready;
} process_structure;

int Queue(int t1)
{
if (t1 == 0 || t1 == 1 || t1 == 2 || t1 == 3)
{
return 1;
}
else
{
return 2;
}
}

int main()
{
int limit, count, temp_process, time, j, y;
process_structure temp;
printf("Enter Total Number of Processes : ");
scanf("%d", &limit);
process_structure process[limit];
for (count = 0; count < limit; count++)
{
process[count].process_id = count+1;
printf("Process : %d\n",count+1);
printf("Arrival Time : ");
scanf("%d", &process[count].arrival_time);
printf("Burst Time : ");
```

```
scanf("%d", &process[count].burst_time);
printf("Process Priority : ");
scanf("%d", &process[count].priority);
temp_process = process[count].priority;
process[count].q = Queue(temp_process);
process[count].ready = 0;
}
time = process[0].burst_time;
for (y = 0; y < limit; y++)
{
for (count = y; count < limit; count++)
{
if (process[count].arrival_time < time)
{
process[count].ready = 1;
}
}
for (count = y; count < limit - 1; count++)
{
for (j = count + 1; j < limit; j++)
{
if (process[count].ready == 1 && process[j].ready == 1)
{
if (process[count].q == 2 && process[j].q == 1)
{
temp = process[count];
process[count] = process[j];
process[j] = temp;
}
}
}
}
for (count = y; count < limit - 1; count++)
{
for (j = count + 1; j < limit; j++)
{
if (process[count].ready == 1 && process[j].ready == 1)
{
if (process[count].q == 1 && process[j].q == 1)
{
if (process[count].burst_time > process[j].burst_time)
{
```

```
temp = process[count];
process[count] = process[j];
process[j] = temp;
}
else
{
break;
}
}
}
}
}
printf("\nProcess[%d] will run from Time %d To %d\n", process[y].process_id,
time, time + process[y].burst_time);
time = time + process[y].burst_time;
for (count = y; count < limit; count++)
{
if (process[count].ready == 1)
{
process[count].ready = 0;
}
}
}
return 0;
}
```

Output :



```
piratepanda@SastaPC:~/Documents/oslab/week6$ gcc queue.c -o queue.out
piratepanda@SastaPC:~/Documents/oslab/week6$ ./queue.out
Enter Total Number of Processes : 3
Process : 1
Arrival Time : 2
Burst Time : 3
Process Priority : 3
Process : 2
Arrival Time : 3
Burst Time : 4
Process Priority : 2
Process : 3
Arrival Time : 3
Burst Time : 2
Process Priority : 4

Process[1] will run from Time:3 To 6

Process[2] will run from Time:6 To 10

Process[3] will run from Time:10 To 12
piratepanda@SastaPC:~/Documents/oslab/week6$
```

.

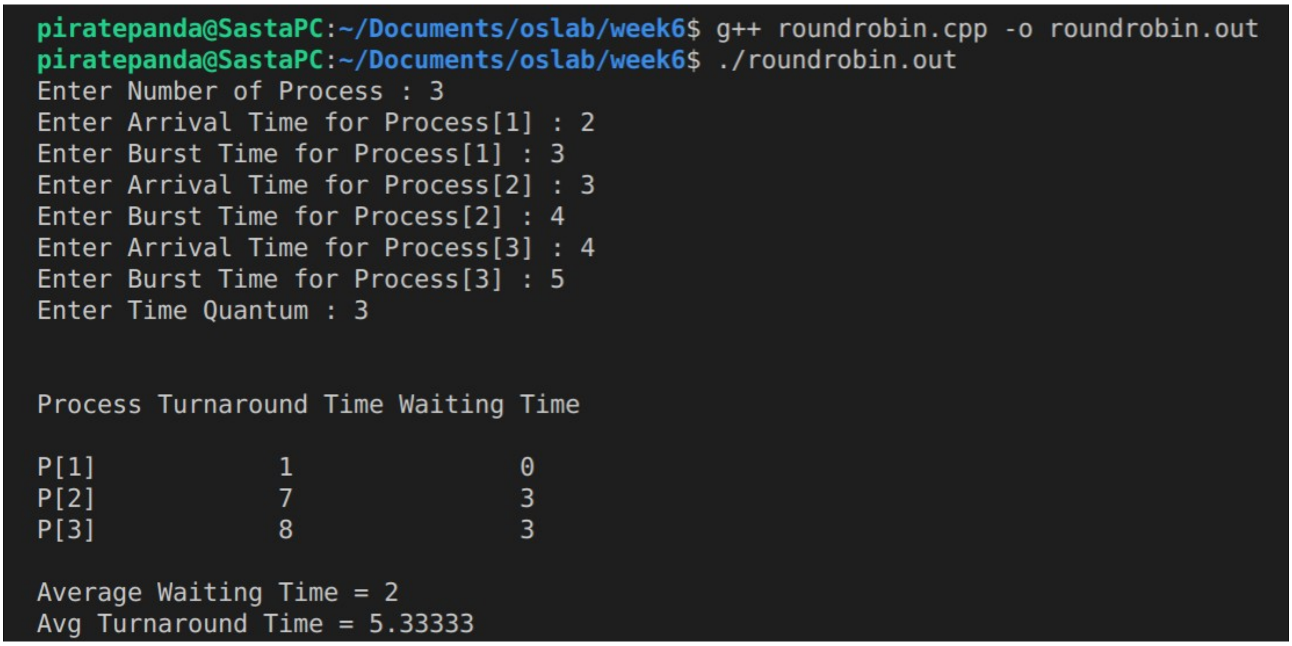
Q3 . Round Robin Scheduling with dynamic factor program in c++

```
#include<iostream>
using namespace std;
int main()
{
int i, processes, j, n, time, unfinished, flag = 0, quantum,wt,tat;
int wait_time = 0, turnaround_time = 0, at[10], bt[10], rt[10];
cout << "Enter Number of Process : ";
cin >> n;
unfinished = n;
for (processes = 0; processes < n; processes++)
{
cout << "Enter Arrival Time for Process[" << processes + 1 << "] : ";
cin >> at[processes];
cout << "Enter Burst Time for Process[" << processes + 1 << "] : ";
cin >> bt[processes];
rt[processes] = bt[processes];
}
```

```
cout << "Enter Time Quantum : ";
cin >> quantum;
cout << "\n\nProcess\tTurnaround Time\tWaiting Time\n\n";
for (time = 0, processes = 0; unfinished != 0;)
{
if (rt[processes] <= quantum && rt[processes] > 0)
{
time += rt[processes];
rt[processes] = 0;
flag = 1;
}
else if (rt[processes] > 0)
{
rt[processes] -= quantum;
time += quantum;
}
if (rt[processes] == 0 && flag == 1)
{
unfinished--;
wt = time - at[processes] - bt[processes];
if(wt<0)
wt=0;
tat = time-at[processes];
cout << "P[" << processes+1 << "]" << "\t\t" << tat << "\t\t" << wt << endl;
wait_time += wt;
turnaround_time += tat;
flag = 0;
}
if (processes == n - 1)
processes = 0;
else if (at[processes + 1] <= time)
processes++;
else
processes = 0;
for(i=processes;i<n;i++){
quantum+=bt[i];
}
quantum = quantum*1.0/unfinished;
}
cout << "\nAverage Waiting Time = " << (wait_time * 1.0 / n) << endl;
cout << "Avg Turnaround Time = " << (turnaround_time * 1.0 / n) << endl;
return 0;
```

}

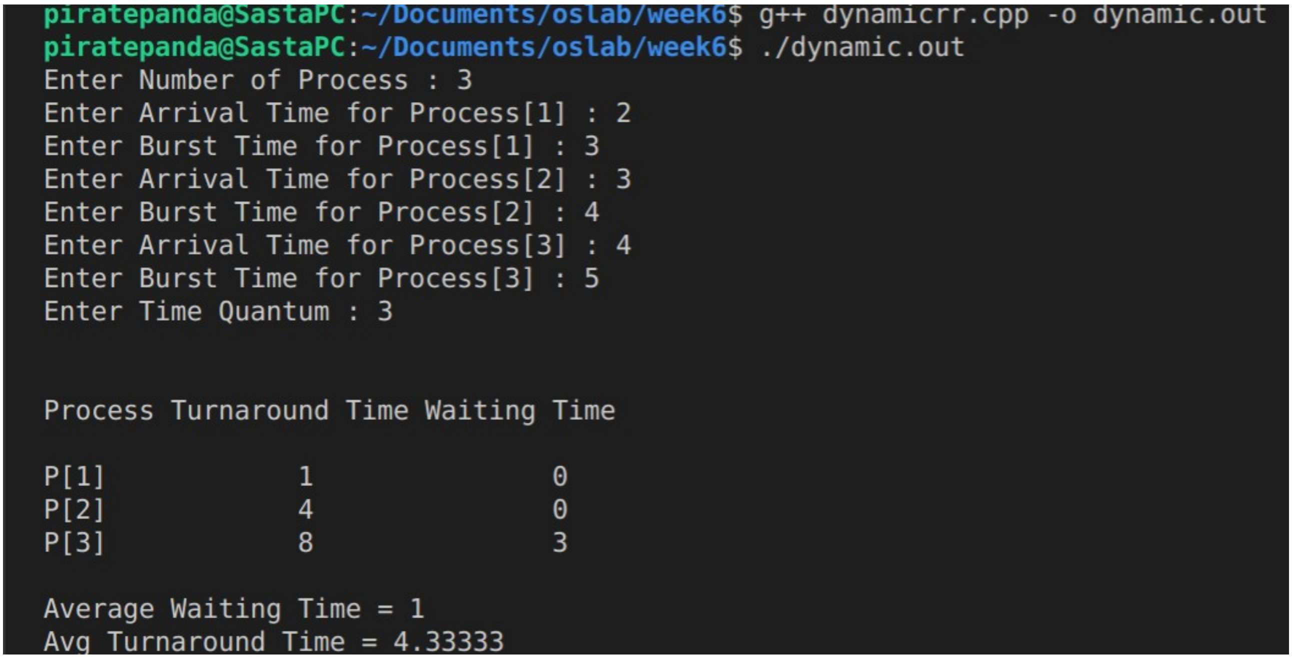
Output with standard Round Robin approach :



```
piratepanda@SastaPC:~/Documents/oslab/week6$ g++ roundrobin.cpp -o roundrobin.out
piratepanda@SastaPC:~/Documents/oslab/week6$ ./roundrobin.out
Enter Number of Process : 3
Enter Arrival Time for Process[1] : 2
Enter Burst Time for Process[1] : 3
Enter Arrival Time for Process[2] : 3
Enter Burst Time for Process[2] : 4
Enter Arrival Time for Process[3] : 4
Enter Burst Time for Process[3] : 5
Enter Time Quantum : 3


Process Turnaround Time Waiting Time

P[1]            1               0
P[2]            7               3
P[3]            8               3

Average Waiting Time = 2
Avg Turnaround Time = 5.33333
```

.

Output with Dynamic Round Robin approach :



```
piratepanda@SastaPC:~/Documents/oslab/week6$ g++ dynamicrr.cpp -o dynamic.out
piratepanda@SastaPC:~/Documents/oslab/week6$ ./dynamic.out
Enter Number of Process : 3
Enter Arrival Time for Process[1] : 2
Enter Burst Time for Process[1] : 3
Enter Arrival Time for Process[2] : 3
Enter Burst Time for Process[2] : 4
Enter Arrival Time for Process[3] : 4
Enter Burst Time for Process[3] : 5
Enter Time Quantum : 3


Process Turnaround Time Waiting Time

P[1]            1               0
P[2]            4               0
P[3]            8               3

Average Waiting Time = 1
Avg Turnaround Time = 4.33333
```

.

Conclusion :

As seen above for identical processes dynamic round robin improved average wait time and average turn around time