**OS Project**

**Student Name:** Sumit kumar

**Student ID:** 11802714

**Section:** K18PA

**Group:** 2

**Roll Number:** 47

**Email Address:** sumitkumar17012000@gmail.com

**GitHub Link:** <https://tinyurl.com/OS-PROJECT>

**Question:** A uniprocessor system has n number of CPU intensive processes, each process has its own requirement of CPU burst. The process with lowest CPU burst is given the highest priority. A late-arriving higher priority process can preempt a currently running process with lower priority. Simulate a scheduler that is scheduling the processes in such a way that higher priority process is never starved due to the execution of lower priority process. What should be its average waiting time and average turnaround time if no two processes are arriving are arriving at same time .

**Code:**

1. #include<stdio.h>

2. int main()

3. {

4. int a[20],b[20],r[20],endTime,i,x;

5. int remain=0,n,time,sum\_wait=0,sum\_turnaround=0;

6. printf("Enter no of Processes : ");

7. scanf("%d",&n);

8. for(i=0;i<n;i++)

9. {

10. printf("\nEnter arrival time for Process P%d : ",i+1);

11. scanf("%d",&a[i]);

12. printf("Enter burst time for Process P%d : ",i+1);

13. scanf("%d",&b[i]);

14. r[i]=b[i];

15. }16. printf("\n\nProcess\t|Turnaround Time| Waiting Time\n\n");

17. r[9]=9999;

18. for(time=0;remain!=n;time++)

19. {

20. x=9;

21. for(i=0;i<n;i++)

22. {

23. if(a[i]<=time && r[i]<r[x] && r[i]>0)

24. {

25. x=i;

26. }

27. }

28. r[x]--;

29. if(r[x]==0)

30. {

31. remain++;

32. endTime=time+1;

33. printf("\nP%d\t|\t%d\t|\t%d",x+1,endTime-a[x],endTime-b[x]-a[x]);

34. sum\_wait=sum\_wait+endTime-b[x]-a[x];

35. sum\_turnaround=sum\_turnaround+endTime-a[x];

36. }

37. }

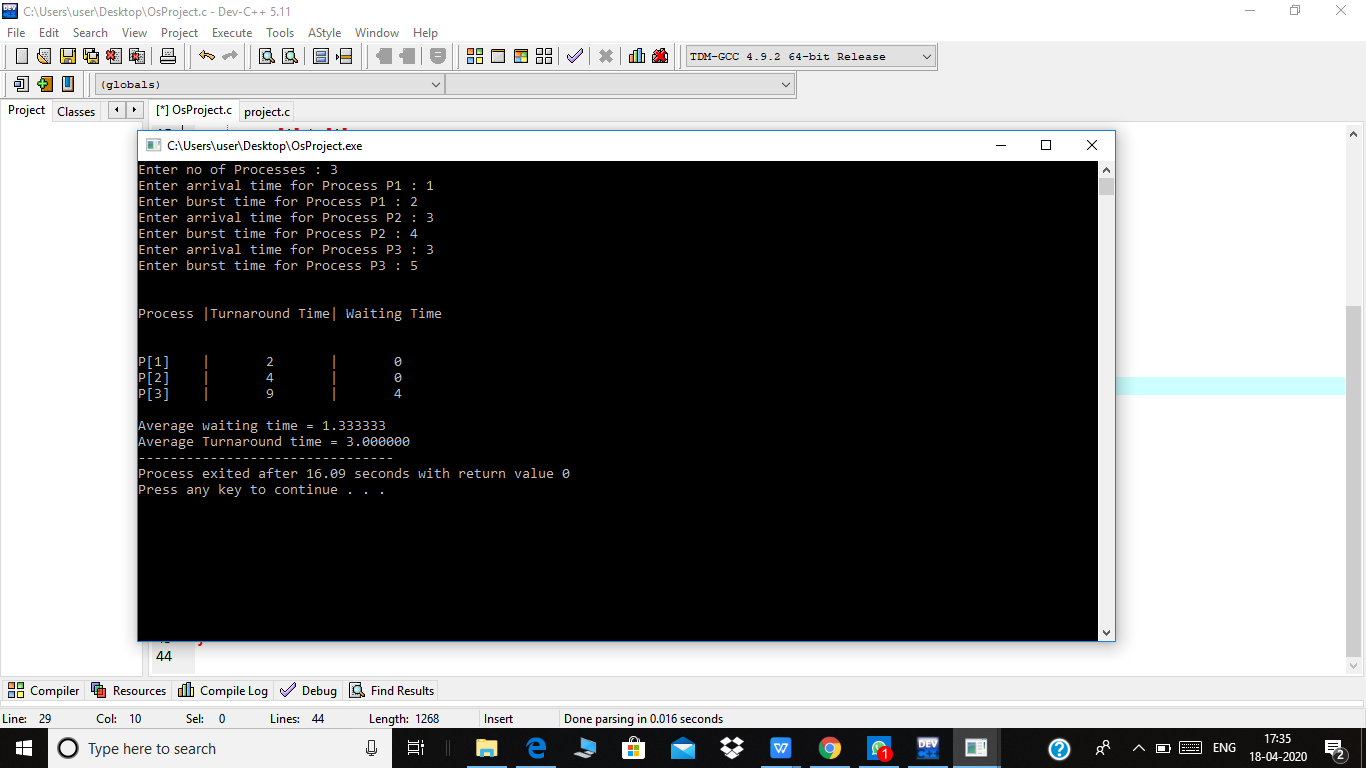
38. printf("\n\nAverage waiting time = %f\n",sum\_wait\*1.0/n);

39. printf("Average Turnaround time = %f",sum\_turnaround\*1.0/n);

40. return 0;

41. }

**Output:-**

****

**Description:** This hassle is solved by using Preemptive SJF, that is also referred to

**as** SRTF. Firstly Shortest Job First (SJF) is an algorithm in which the technique having the smallest execution time is selected for the subsequent execution. This

scheduling method may be preemptive or non-preemptive. We use Preemptive in this, so In Preemptive SJF Scheduling, jobs are put into the prepared queue as they come.

A method with shortest burst time begins execution. If a method with even a shorter burst time arrives, the present day technique is eliminated or preempted from execution, and the shorter activity is allotted CPU cycle.

This algorithm requires advanced concept and knowledge of CPU time required to

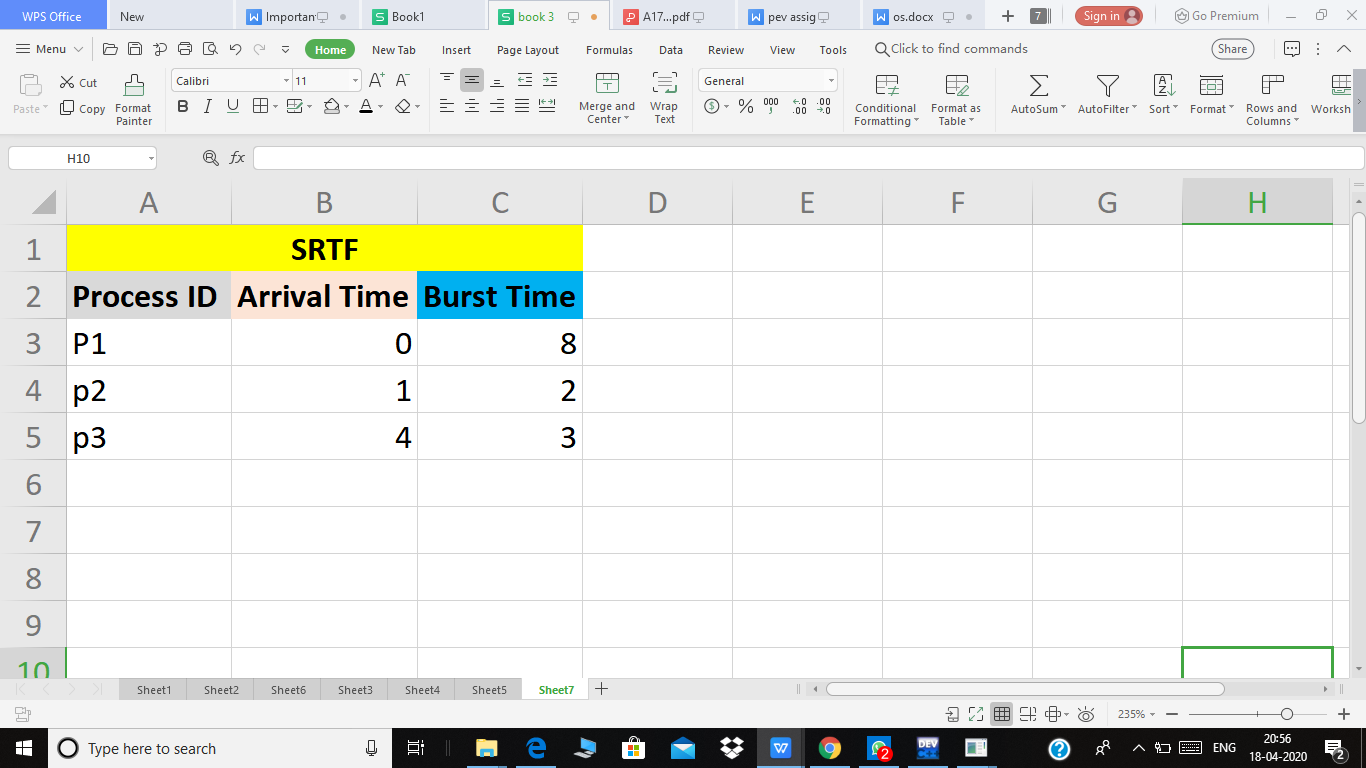
process the job in an interactive system, and hence can’t be implemented there. But, in a batch system where it is desirable to give preference to short jobs, SRT algorithm is used.

**Algorithm: SRTF**, Which Stands for **Shortest Remaining Time First** is a scheduling algorithm used in Operating Systems, which can also be called as the preemptive version of the SJF scheduling algorithm. The process which has the least processing time remaining is executed first. As it is a preemptive type of schedule, it is claimed to be better than SJF Scheduling Algorithm.

**Example:**

Let's understand this with the help of an example. Suppose we have the following 3

processes with process ID's **P1**, **P2**, and **P3** and they arrive into the CPU in the following manner.



**Gant Chart:**

IMG_256

**Explanation:**

1. At the 0th unit of the CPU, we have only process **P1**, so it gets executed for

the 1-time unit.

1. At the 1st unit of the CPU, the Process **P2** also arrives. Now, the **P1** needs 7

more units more to be executed, and **P2** needs only 2 units. So, **P2** is executed by preempting **P1**.• **P2** gets completed at time unit 3, and unit now no new process has arrived. So, after the completion of **P2**, again **P1** is sent for execution.

1. Now, **P1** has been executed for one unit only, and we have an arrival of

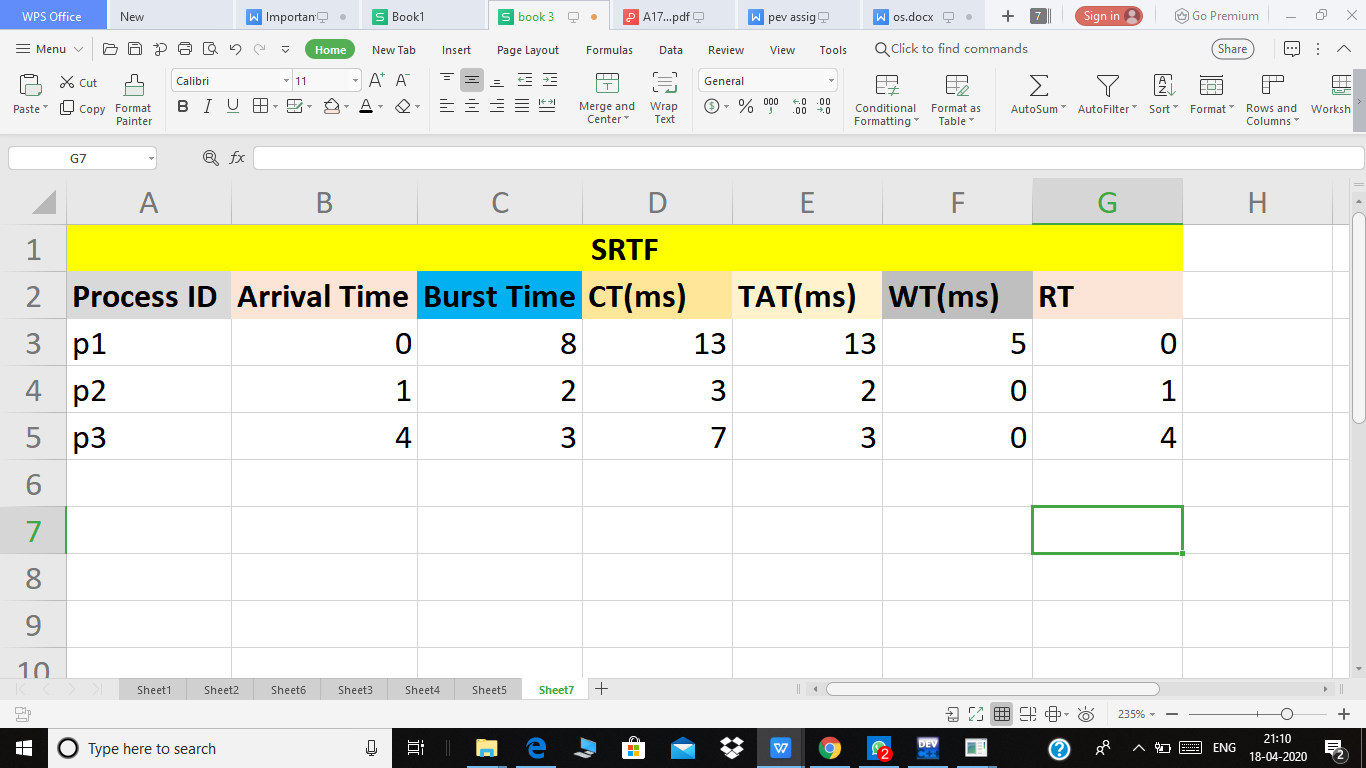
new process **P3** at time unit 4. Now, the **P1** needs 6-time units more

and **P3** needs only 3-time units. So, **P3** is executed by preempting **P1**.

1. **P1** gets completed at time unit 7, and after that, we have the arrival of no

other process. So again, **P1** is sent for execution, and it gets completed at

13th unit.



Turn Around time= 13+2+3 = 18 ms

Average Turn Around Time= Total Turn Around Time/ Total no of Processes

= 18/3

=6 ms

Total Waiting Time= 5+0+0 = 5 ms

Average Waiting time = Total Waiting time/ Total no of processes

=5/3

=1.67 ms

**Code Explanation:**

**Line no 8 to 15** - Are used for taking inputs. The array variables `a` and `b` stores time of arrival and burst time of processes. Array `r` stores the remaining time of the processes that at first are adequate to the burst time of the processes. this is often the time to complete execution of all the processes.

**Line no 18** starts a loop where time starts from zero.

**line no 20 to 27** are used to find a process which have the least remaining time and it’s arrival time is less than or equal to the current time. This condition is checked at every unit time, that is why time is incremented by 1 (time++).

**Line no 26** is used to print the arrival time and burst time for finished processes. At this moment, variable `time` gives the time of start of the process.

Therefore Turnaround time=(time+burst time-arrival time), Waiting time =(time – arrival time)

In **line no 28** remaining time for shortest process is decreased by one since time is increased by one. If a particular process have just finished/completed, then it satisfies the condition in **line no 29**.

Since the variable time stores the current time, and the process was running at this moment, therefore its ending time = time+1 (**line no 32**).

Since burst time+waiting time = end time - Start time

Therefore waiting time=end Time-burst time-arrival time

waiting time and turnaround time of all processes are summed up and divided by n to give their average.

1. Explain the boundary conditions of the implemented code.

**Description:**

1. Explain all the test cases applied on the solution of assigned problem**.**

**Description:**

**8.Have you made minimum 5 revisions of solution on GitHub?**

**GitHub Link:**

**Test Cases:-**

**Test Case 1:-**

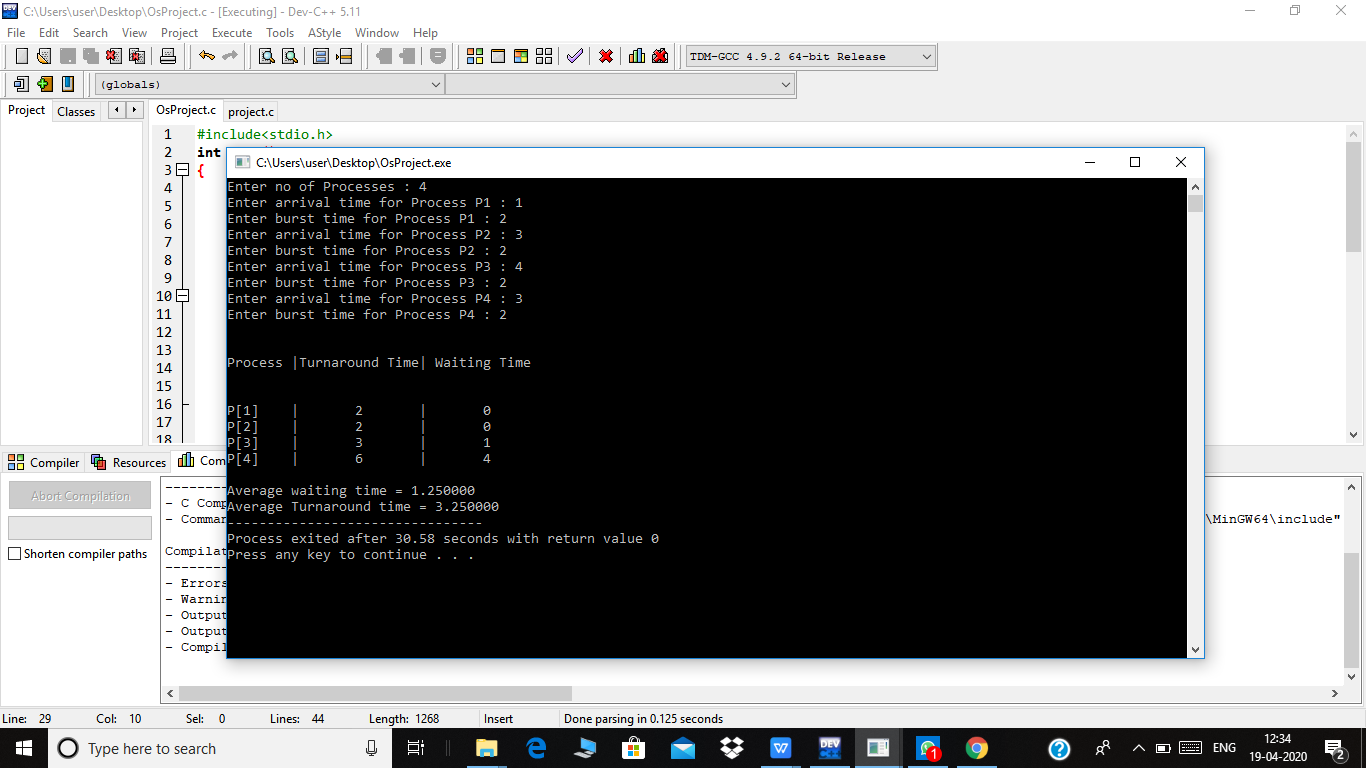
**Process ID Arrival Time Burst Time**

**P1 1 2**

**P2 3 2**

**P3 4 2**

**P4 3 2**

****

**Test Case 2:-**

**Process ID Arrival Time Burst Time**

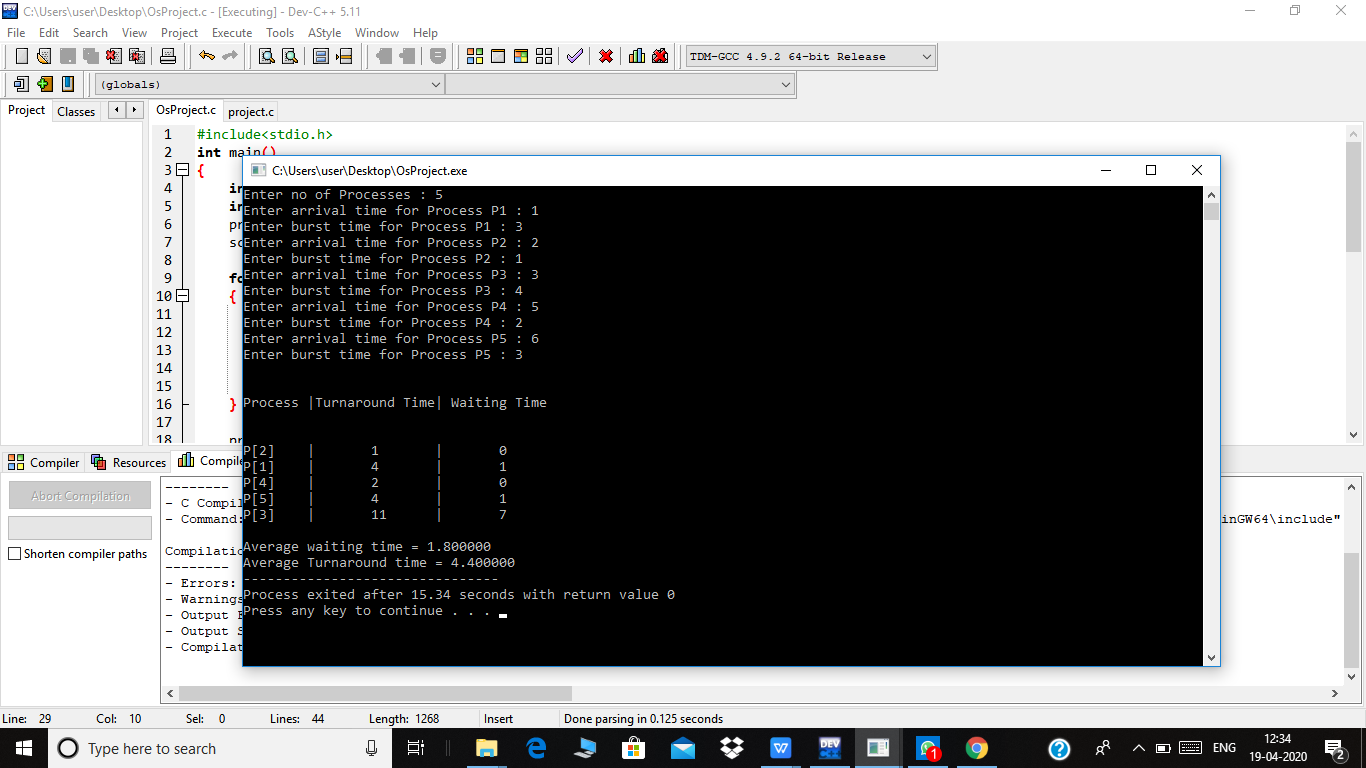
**P1 1 3**

**P2 2 1**

**P3 3 4**

**P4 5 2**

**P5 6 3**

****

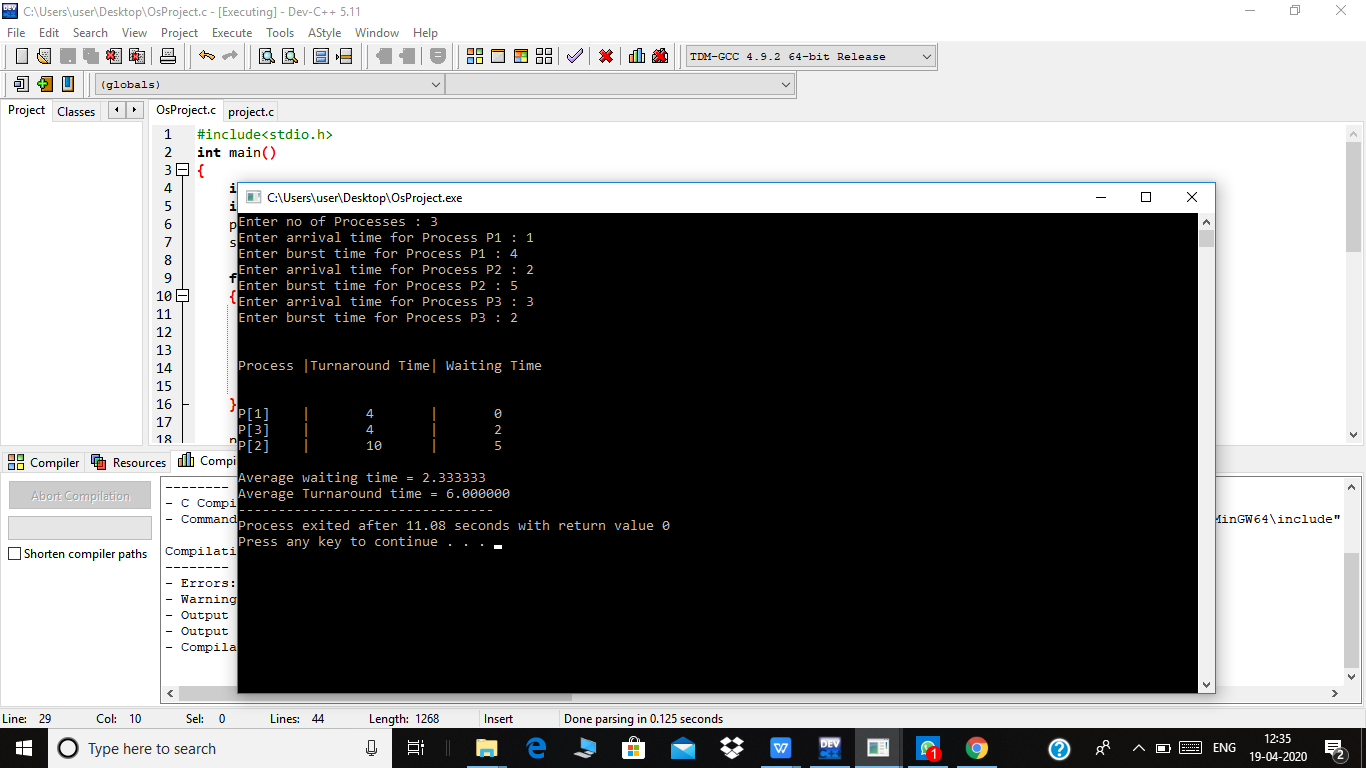
**Test Case 3:-**

**Process ID Arrival Time Burst Time**

**P1 1 4**

**P2 2 5**

**P3 3 2**

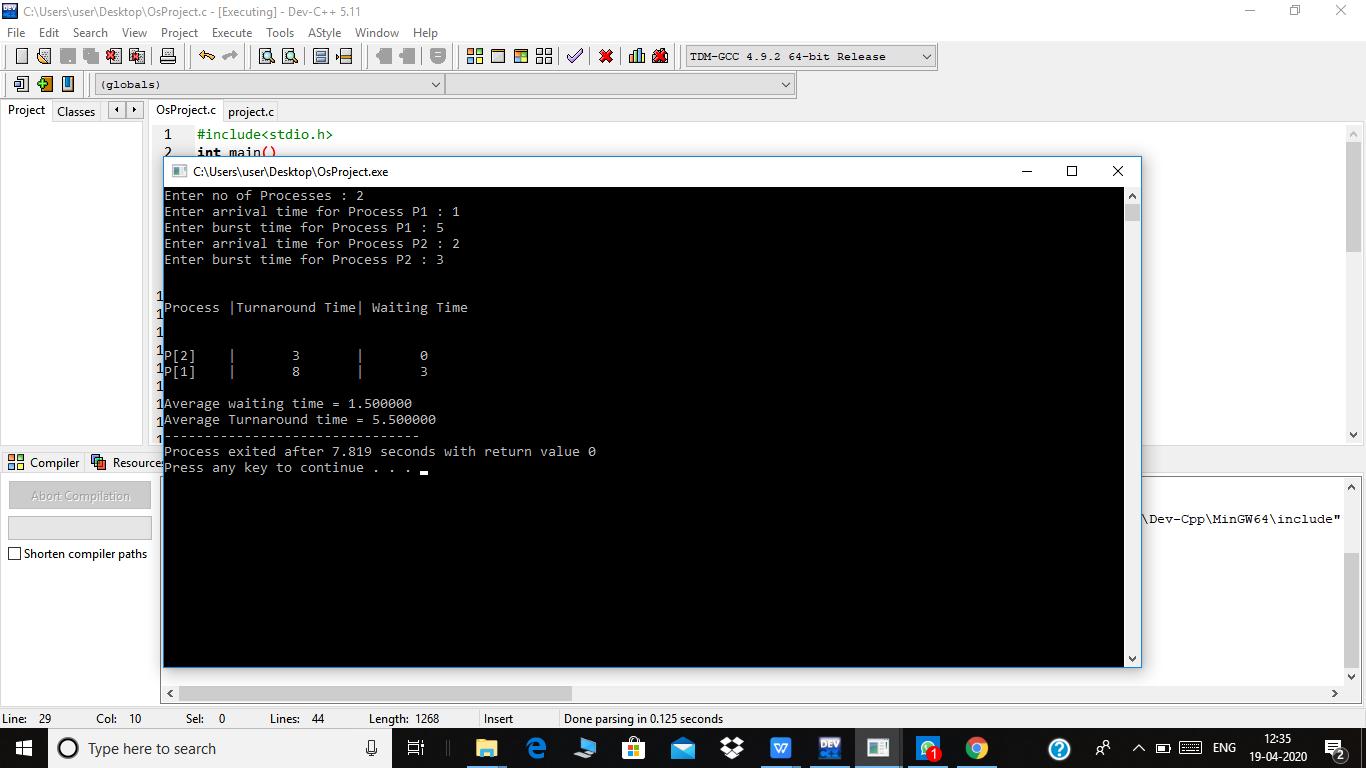
****

**Test Case 4:-**

**Process ID Arrival Time Burst Time**

**P1 1 5**

**P2 2 3**

****

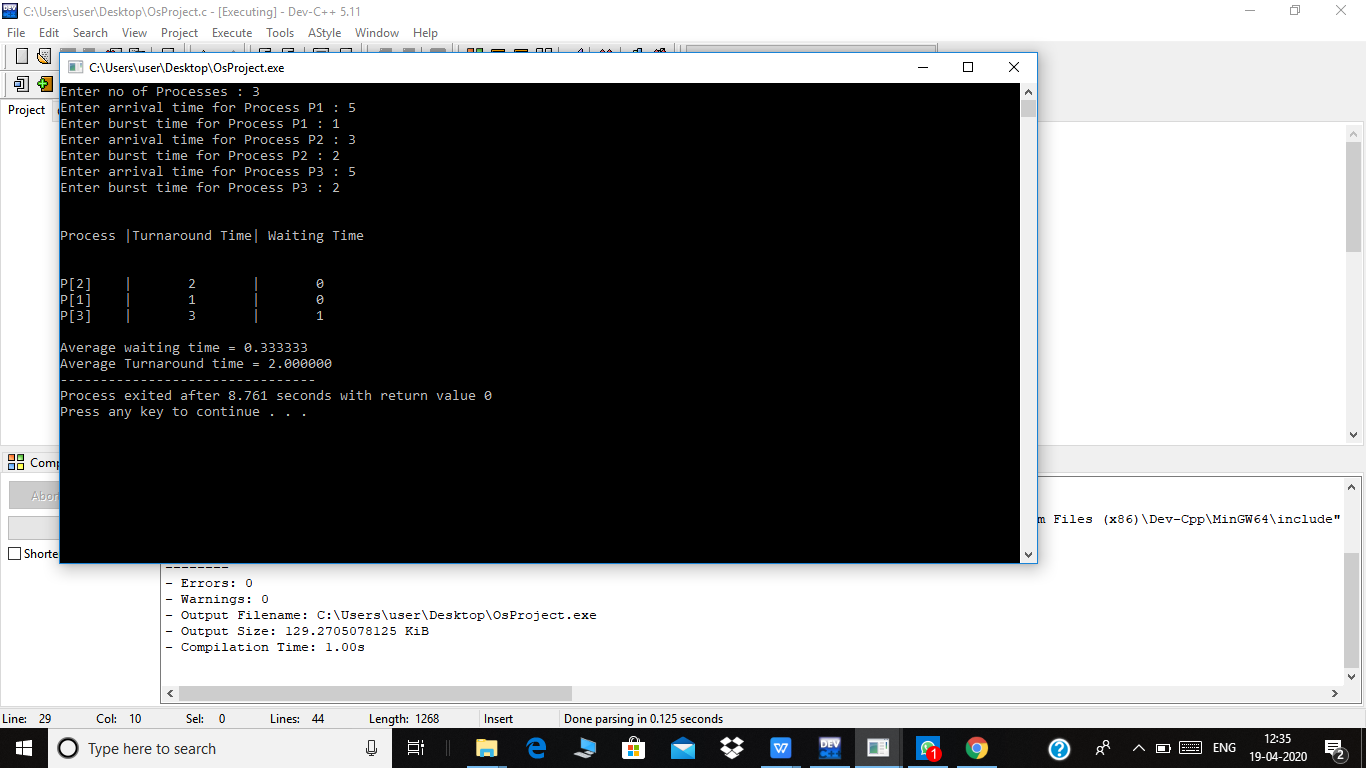
**Test Case 5:-**

**Process ID Arrival Time Burst Time**

**P1 5 1**

**P2 3 2**

**P3 5 2**

****

**Revisions:**

