LOVELY PROFESSIONAL UNIVERSITY

Transforming Education Transforming India

OPERATING SYSTEM PROJECT

Department of Computer Science Engineering (CSE) And technology

NAME: K. SAI VINESH

EMAIL:vineshvictorious@gmail.com

REG NUMBER: 11803034

ROLL.NO: 62

SECTION: K18JC

Submitted to: Isha Mam.

GITHUB LINK:

QUESTION :

Ques.16. Design a scheduler that can schedule the processes arriving system at periodical intervals. Every process is assigned with a fixed time slice t milliseconds. If it is not able to complete its execution within the assigned time quantum, then automated timer generates an interrupt. The scheduler will select the next process in the queue and dispatcher dispatches the process to processor for execution. Compute the total time for which processes were in the queue waiting for the processor. Take the input for CPU burst, arrival time and time quantum from the user.

CODE:

```
#include<stdio.h>

int main()
{
    int i, limit, total = 0, x, counter = 0, time_quantum;

    int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10], temp[10];

    float average_wait_time, average_turnaround_time;

   printf("\nEnter Total Number of Processes:\t");

    scanf("%d", &limit);

    x = limit;

    for(i = 0; i < limit; i++)
    {
        printf("\nEnter Details of Process[%d]\n", i + 1);

        printf("Arrival Time:\t");

        scanf("%d", &arrival_time[i]);

        printf("Burst Time:\t");

        scanf("%d", &burst_time[i]);
        temp[i] = burst_time[i];
    }
    printf("\nEnter Time Quantum:\t");

    scanf("%d", &time_quantum);

    printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");

    for(total = 0, i = 0; x != 0;)
    {
        if(temp[i] <= time_quantum && temp[i] > 0)
        {
            total = total + temp[i];
            temp[i] = 0;
            counter = 1;
        }
        else if(temp[i] > 0)
        {
            temp[i] = temp[i] - time_quantum;
            total = total + time_quantum;
        }
        if(temp[i] == 0 && counter == 1)
        {
            x--;
            printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burst_time[i], total -
arrival_time[i], total - arrival_time[i] - burst_time[i]);
            wait_time = wait_time + total - arrival_time[i] - burst_time[i];
            turnaround_time = turnaround_time + total - arrival_time[i];
            counter = 0;
        }
    }
    average_wait_time = wait_time * 1.0 / limit;

    average_turnaround_time = turnaround_time * 1.0 / limit;

    printf("\n\nTotal Waiting Time:\t%d", wait_time);

    printf("\n\nAverage Waiting Time:\t%f", average_wait_time);

    printf("\nAvg Turnaround Time:\t%f\n", average_turnaround_time);
    return 0;
}
```

Description:

Design a scheduler that can schedule the processes arriving system at periodical intervals. Every process is assigned with a fixed time slice t milliseconds. If it is not able to complete its execution within the assigned time quantum, then automated timer generates an interrupt. The scheduler will select the next process in the queue and Dispatcher dispatches the process to processor for execution. Compute the total time for which processes were in the queue waiting for the processor. Take the input for CPU burst, arrival time and time quantum from the user. The Algorithm focuses on Time Sharing. In this algorithm, every process gets executed in a cyclic way. A certain time slice is defined in the system which is called time quantum. Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will terminate else the process will go back to the ready queue and waits for the next turn to complete the execution.

Algorithm:

1. Create an array arrival_time[], burst_time[] to keep track of arrival and bust time of processes.
2. Create another array temp[] to store waiting times of processes temporarily in between execution.
3. Initialize time: total = 0, counter = 0, wait time = 0, turnaround time = 0.
4. Ask user for no of process and store it in limit.
5. Repetitively ask user to give input for - arrival time, bust time up to limit.
6. Ask user to enter time quantum and store it into time_quantum.
7. Keep traversing the all processes while all processes are not done. Do

following for i'th process if it is not done yet. a. If temp[i] <= time_quantum && temp[i] > 0 (i) total = total + temp[i]; (ii) temp[i] = 0; (iii) counter = 1; b. Else if temp[i] > 0 (i) temp[i] = temp[i] - time_quantum; (ii) total = total + time_quantum; c. If temp[i] == 0 && counter == 1 (i) x--; (ii) print burst_time[i], total, arrival_time[i], total - arrival_time[i] - burst_time[i]); (iii) wait_time = wait_time + total - arrival_time[i] - burst_time[i]; (iv) turnaround_time = turnaround_time + total - arrival_time[i]; (v) counter = 0; d. If i == limit – 1 (i) i = 0; e. Else if arrival_time[i + 1] <= total (i) i++; c. Else (i) i = 0;

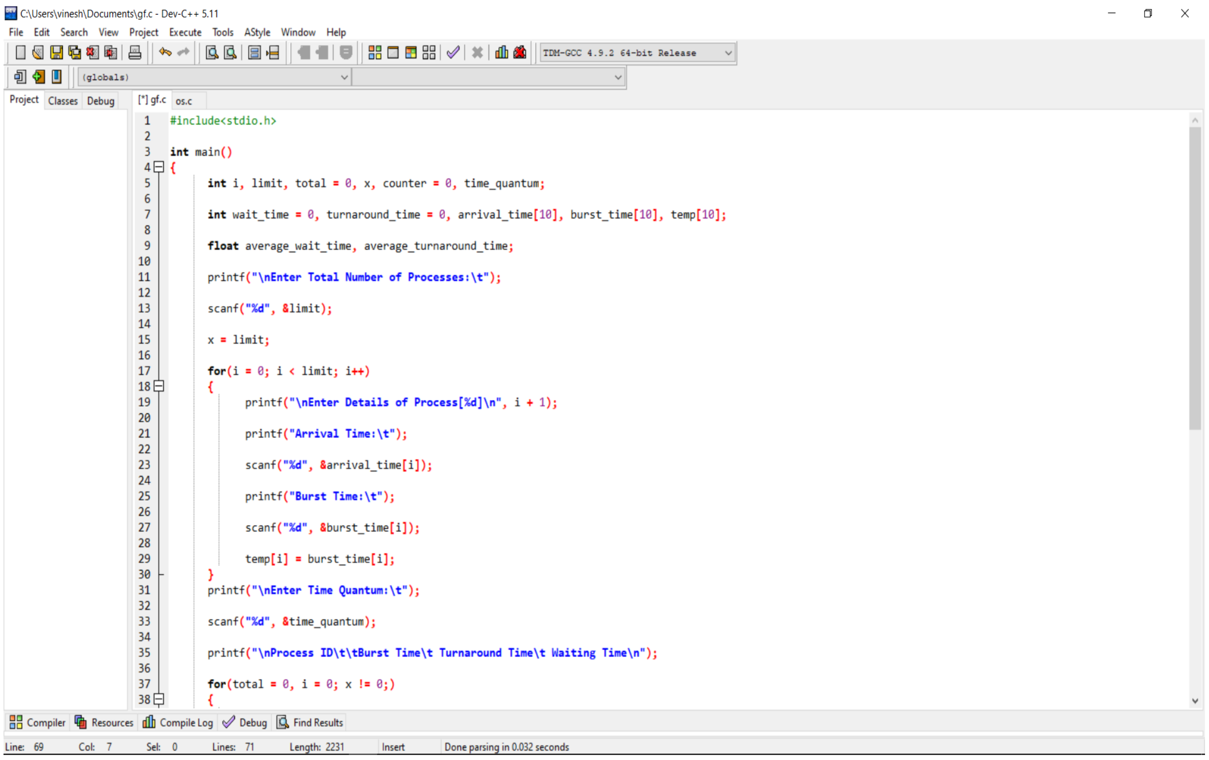
8. Print total waiting time - wait_time, average waiting time - wait_time/limit, average turnaround time - average_turnaround_time/limit.

Constraints:

1. If we have large number of processes and a process with very less burst time. According to round robin it will have to wait for very long time if burst time of other processes is large and its turn comes late.

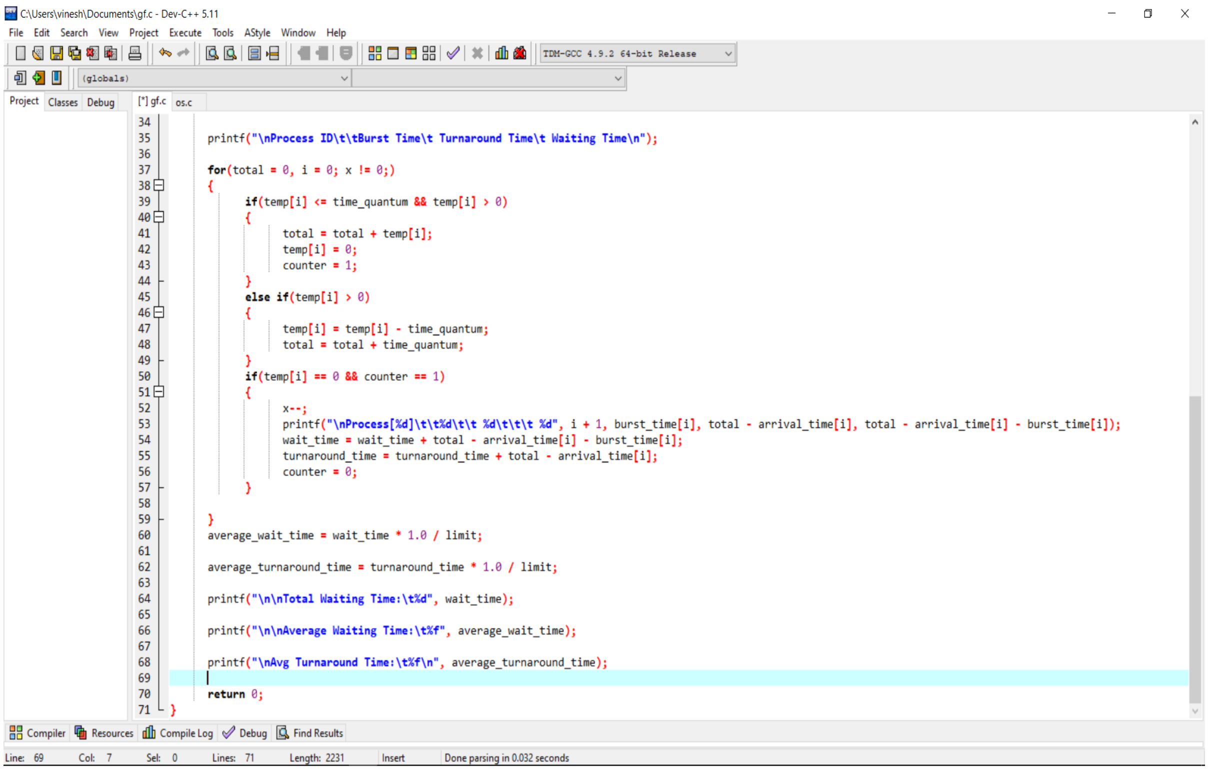
2. If we have some processes running according to round robin and we have a process whose arrival time is after the completion of those processes, then that process will not execute. If we want that all the processes should execute successfully then the arrival time of other process must be less than or equal to the completion time of the running processes.

Code snippet:

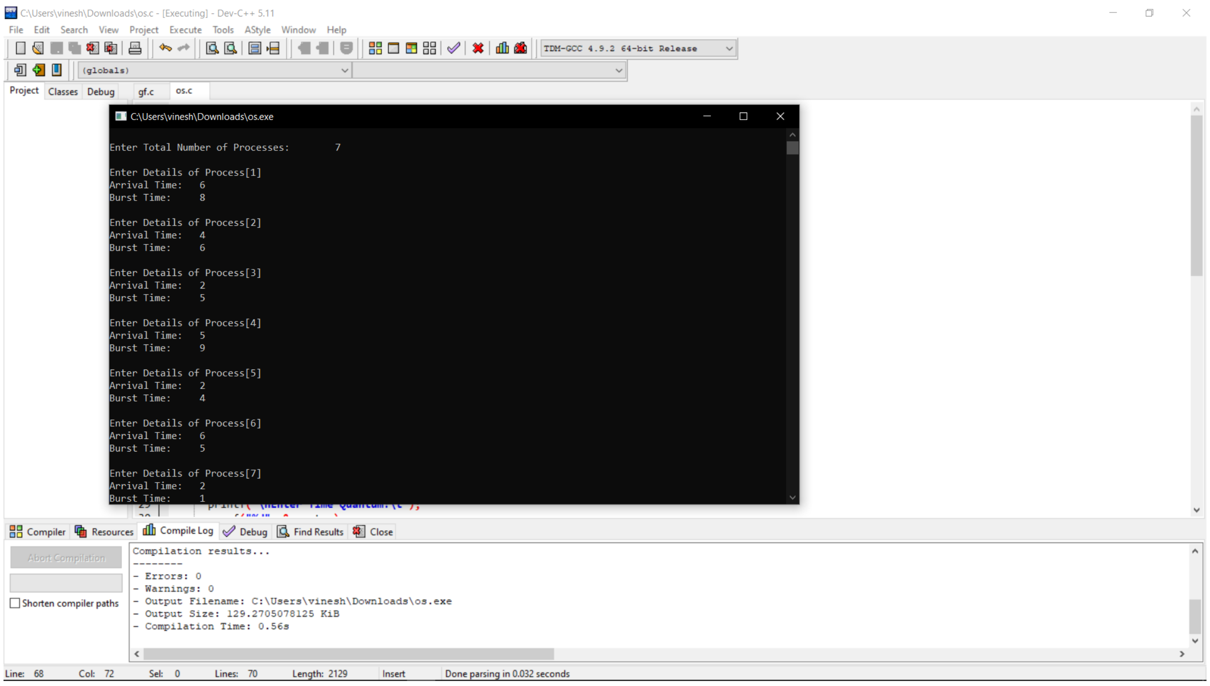


```c
#include<stdio.h>

int main()
{
	int i, limit, total = 0, x, counter = 0, time_quantum;

	int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10], temp[10];

	float average_wait_time, average_turnaround_time;

	printf("\nEnter Total Number of Processes:\t");

	scanf("%d", &limit);

	x = limit;

	for(i = 0; i < limit; i++)
	{
		printf("\nEnter Details of Process[%d]\n", i + 1);

		printf("Arrival Time:\t");

		scanf("%d", &arrival_time[i]);

		printf("Burst Time:\t");

		scanf("%d", &burst_time[i]);

		temp[i] = burst_time[i];
	}
	printf("\nEnter Time Quantum:\t");

	scanf("%d", &time_quantum);

	printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");

	for(total = 0, i = 0; x != 0;)
	{
```

C:\Users\vinesh\Documents\gf.c - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
[*] gf.c os.c
Compiler Resources Compile Log Debug Find Results
Line: 69 Col: 7 Sel: 0 Lines: 71 Length: 2231 Insert Done parsing in 0.032 seconds

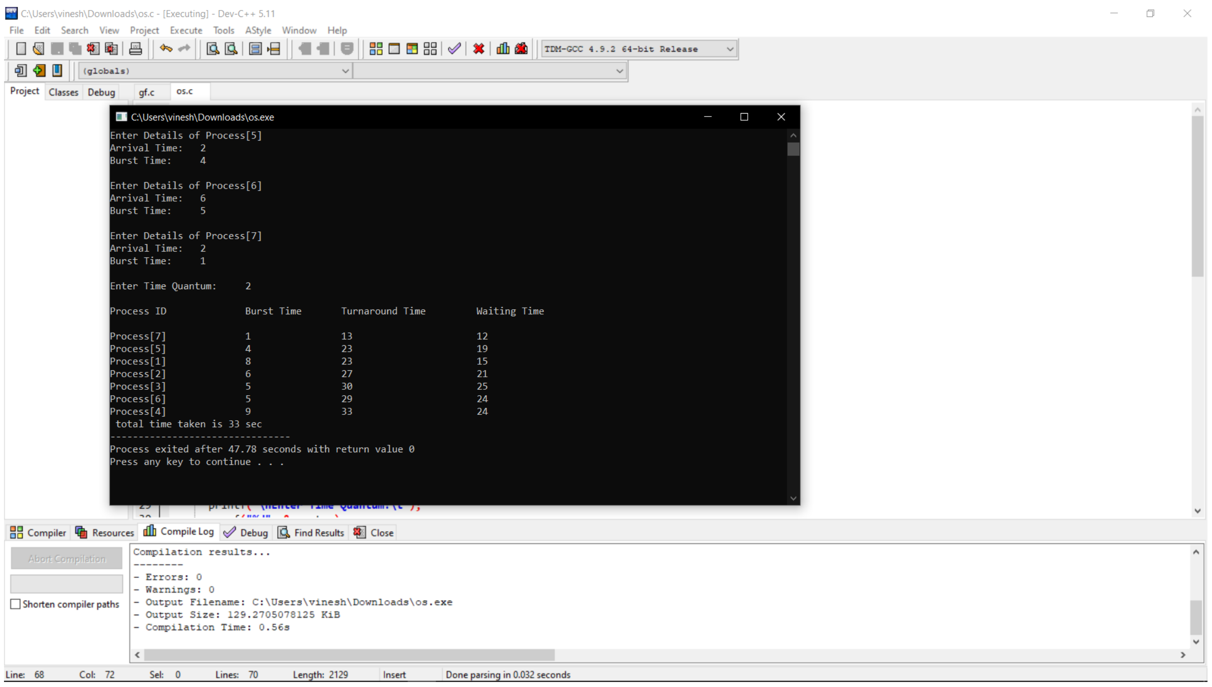


```c
    printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");

    for(total = 0, i = 0; x != 0;)
    {
        if(temp[i] <= time_quantum && temp[i] > 0)
        {
            total = total + temp[i];
            temp[i] = 0;
            counter = 1;
        }
        else if(temp[i] > 0)
        {
            temp[i] = temp[i] - time_quantum;
            total = total + time_quantum;
        }
        if(temp[i] == 0 && counter == 1)
        {
            x--;
            printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d", i + 1, burst_time[i], total - arrival_time[i], total - arrival_time[i] - burst_time[i]);
            wait_time = wait_time + total - arrival_time[i] - burst_time[i];
            turnaround_time = turnaround_time + total - arrival_time[i];
            counter = 0;
        }

    }
    average_wait_time = wait_time * 1.0 / limit;

    average_turnaround_time = turnaround_time * 1.0 / limit;

    printf("\n\nTotal Waiting Time:\t%d", wait_time);

    printf("\n\nAverage Waiting Time:\t%f", average_wait_time);

    printf("\nAvg Turnaround Time:\t%f\n", average_turnaround_time);

    return 0;
}
```



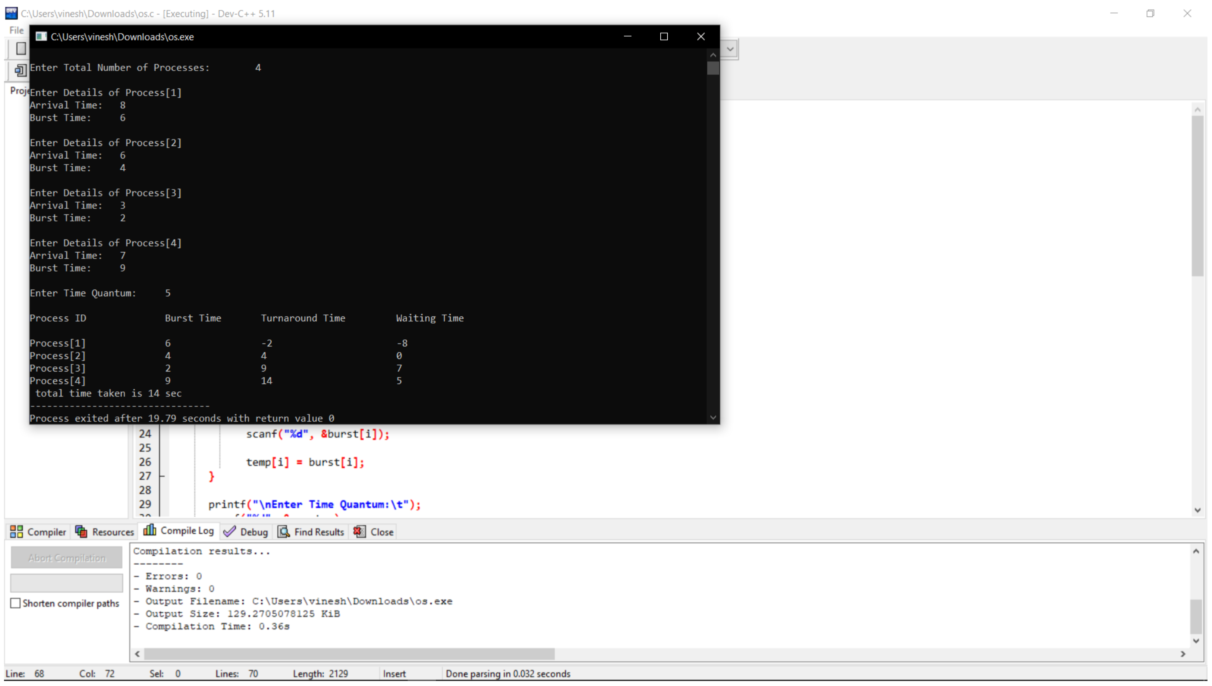
```
Enter Total Number of Processes:        7

Enter Details of Process[1]
Arrival Time:   6
Burst Time:     8

Enter Details of Process[2]
Arrival Time:   4
Burst Time:     6

Enter Details of Process[3]
Arrival Time:   2
Burst Time:     5

Enter Details of Process[4]
Arrival Time:   5
Burst Time:     9

Enter Details of Process[5]
Arrival Time:   2
Burst Time:     4

Enter Details of Process[6]
Arrival Time:   6
Burst Time:     5

Enter Details of Process[7]
Arrival Time:   2
Burst Time:     1
```

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\vinesh\Downloads\os.exe
- Output Size: 129.2705078125 KiB
- Compilation Time: 0.56s
```



```
Enter Details of Process[5]
Arrival Time:   2
Burst Time:     4

Enter Details of Process[6]
Arrival Time:   6
Burst Time:     5

Enter Details of Process[7]
Arrival Time:   2
Burst Time:     1

Enter Time Quantum:     2

Process ID              Burst Time      Turnaround Time         Waiting Time

Process[7]              1               13                      12
Process[5]              4               23                      19
Process[1]              8               23                      15
Process[2]              6               27                      21
Process[3]              5               30                      25
Process[6]              5               29                      24
Process[4]              9               33                      24
 total time taken is 33 sec
--------------------------------
Process exited after 47.78 seconds with return value 0
Press any key to continue . . .
```

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\vinesh\Downloads\os.exe
- Output Size: 129.2705078125 KiB
- Compilation Time: 0.56s
```



```
Enter Total Number of Processes:        4

Enter Details of Process[1]
Arrival Time:   8
Burst Time:     6

Enter Details of Process[2]
Arrival Time:   6
Burst Time:     4

Enter Details of Process[3]
Arrival Time:   3
Burst Time:     2

Enter Details of Process[4]
Arrival Time:   7
Burst Time:     9

Enter Time Quantum:     5

Process ID              Burst Time      Turnaround Time         Waiting Time

Process[1]              6               -2                      -8
Process[2]              4               4                       0
Process[3]              2               9                       7
Process[4]              9               14                      5
 total time taken is 14 sec
--------------------------------
Process exited after 19.79 seconds with return value 0
```

```
            scanf("%d", &burst[i]);

            temp[i] = burst[i];
    }

    printf("\nEnter Time Quantum:\t");
```

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\vinesh\Downloads\os.exe
- Output Size: 129.2705078125 KiB
- Compilation Time: 0.36s
```

TEST CASES:

Test case number 1:

Time Quantum: 2

Process	Arrival Time	Burst Time	Turnaround Time	Waiting Time
P1	6	8	23	15
P2	4	6	27	21
P3	2	5	30	25

P4	5	9	33	24
P5	2	4	23	19
P6	6	5	29	24
P7	2	1	13	12

Total time taken: 33 sec

Test case number 2:

Time Quantum: 5

Process	Arrival Time	Burst Time	Turnaround Time	Waiting Time
P1	8	6	-2	15
P2	6	4	4	21
P3	3	2	9	25
P4	7	9	14	24

Total time taken: 14 sec