**Name : Harsh Sinha**

**Reg. No. : 11708322**

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

**GITHUB Link :** [**https://github.com/harshputu/LPU-OS-Project**](https://github.com/harshputu/LPU-OS-Project)

**Code : 3**

**Description:**

* This is a scheduling program that is capable of scheduling many processes that comes in at some time interval and is allocated the CPU.
* This scheduler which schedules the job by considering the arrival time of the processes where arrival time if given as 0 is discarded or displayed as error.
* The scheduler implements the shortest job first scheduling policy, but checks the queue of the processes after the every process terminates and time taken for checking and arranging the process according to the shortest job is 2 time unit.
* At last it computes the waiting time for process, turnaround time for process and average waiting time and turnaround time and the total time taken by the processor to compute all the jobs.

**SJF(Shortest Job First Scheduling)**

* Shortest job first is a scheduling algorithm in which the process with the smallest execution time is selected for execution next.
* The scheduling in which a running process cannot be interrupted by any other process is called non-preemptive scheduling.
* Any other process which enters the queue has to wait until the current process finishes its CPU cycle.
* It is a Greedy Algorithm
* It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of aging.
* It is practically infeasible as Operating System may not know burst time and therefore may not sort them. While it is not possible to predict execution time, several methods can be used to estimate the execution time for a job, such as a weighted average of previous execution times. SJF can be used in specialized environments where accurate estimates of running time are available.
* The throughput is increased because more processes can be executed in less amount of time.

**Algorithm: (SJF Non Preemptive)**

**Step 1:** Start

**Step 2:** Declare variable : int i, n ,process[10];

int burstTime[10],temp,j,arrivalTime[10],waitTime[10],turnAroundTime[10],completionTime[10],ta=0,csum=1,real=1,turnArountTime1[10],completionTime1[10],large;

float waitAvg=0,turnAvg=0,turnSum=0,waitSum=0;

**Step 3:** Read values using for loop.

**Step 4**: Apply bubble sort to sort the array of arrival time and burst time.

[ example :

1. The first element(index = 0), compare the current element with the next element of the array.

2. If the current element is greater than the next element of the array, swap them.

3. If the current element is less than the next element, move to the next element.

And repeat…………………….. until the loop ends

]

if(arrivalTime[i]<arrivalTime[j]) sort processes according to arrival time.

if (burstTime[i]>=arrivalTime[i] && burstTime[i]>burstTime[j] ) sort process according to the burst time and compare the processes according to there completion time as the arrival time of the next process should be less than the burst time of previous process.

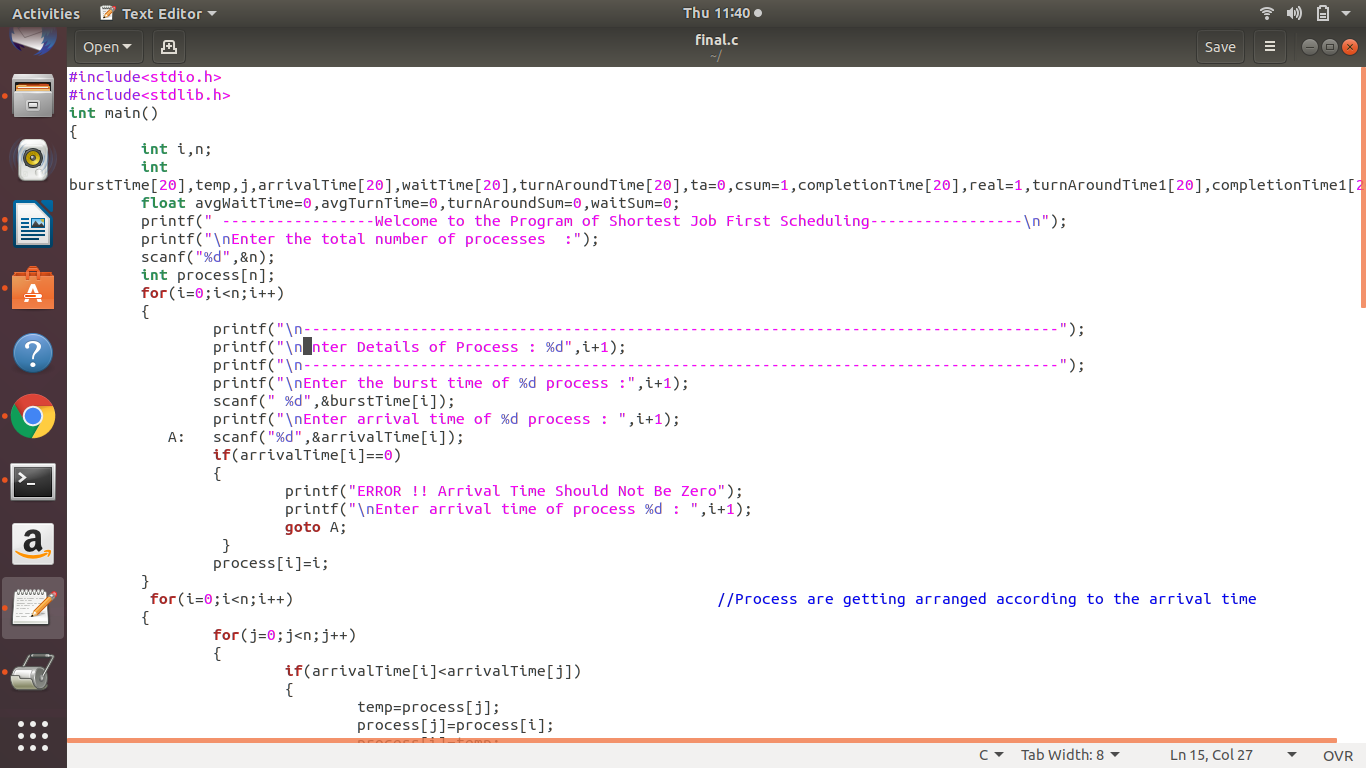
**Step 5:** Now calculate turn around time , waiting time, completion time and after completion of one loop give 2 unit rest to the cpu (increases the completion time of last process by 2 unit ) turn around time = completion time – arrival time , waiting time = turn around time – burst time , completion time = turn around time + arrival time.

**Step 6:** Calculate average waiting time(total waiting time / total no. of processes) , average turn around time(total turn around time / total no. of processes) and total completion time(total completion time / total no. of processes) to complete all the jobs.

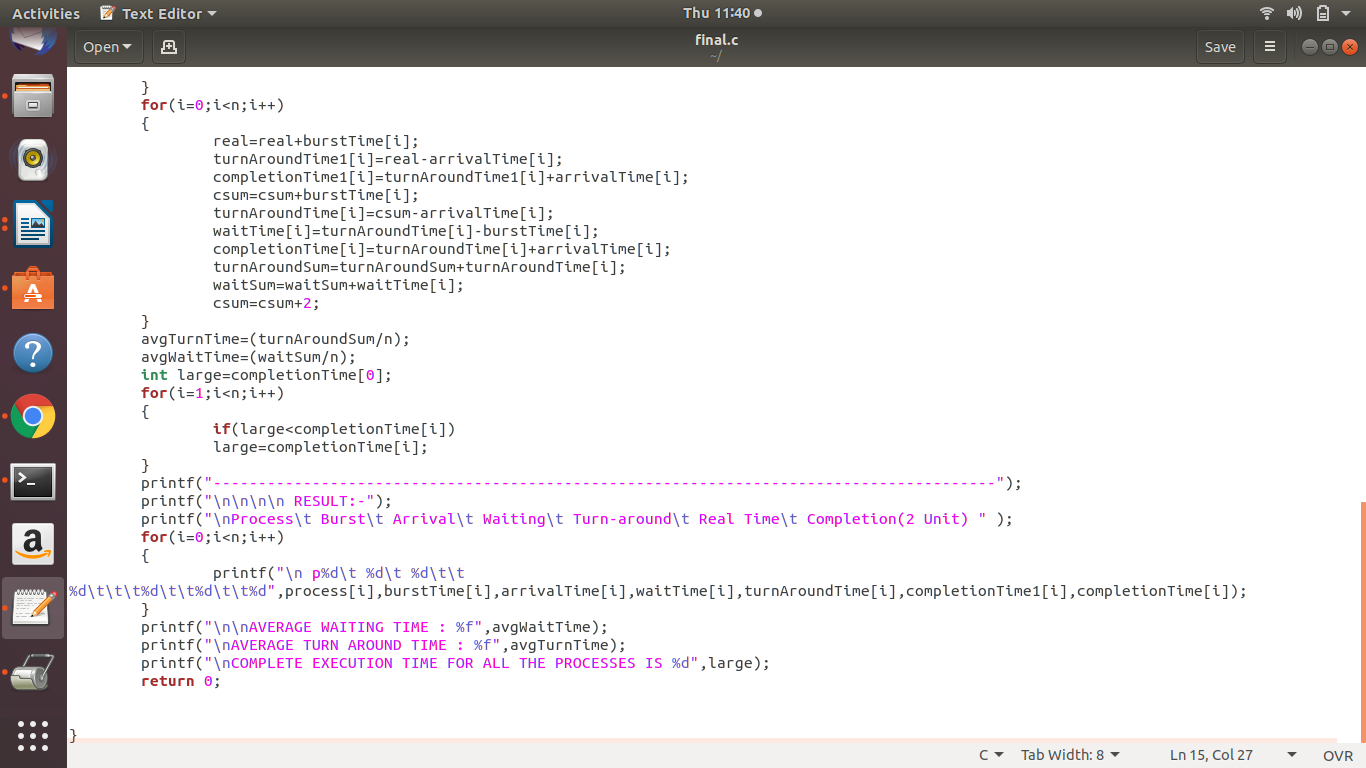
**Step 7:** Stop

**Constraints :**

1. Arrival time should not be zero if given zero print “ERROR”

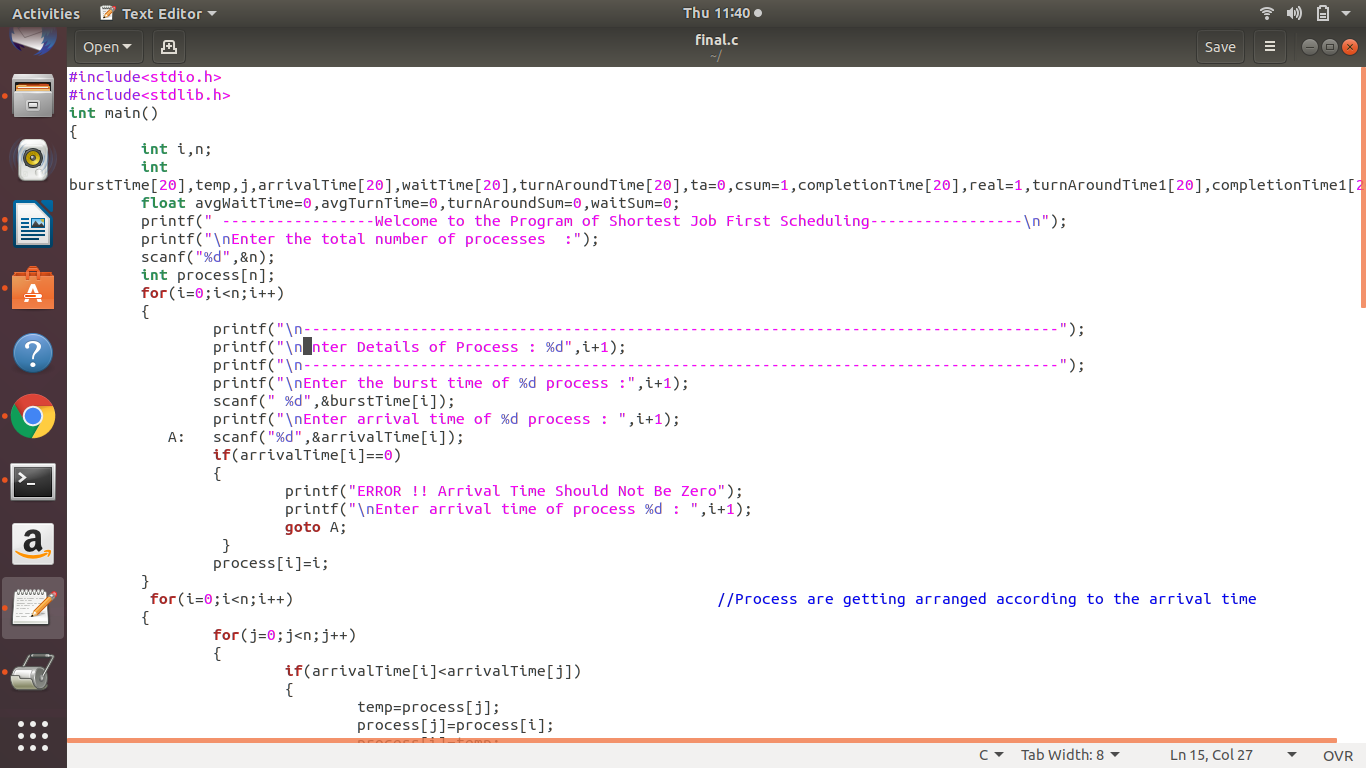


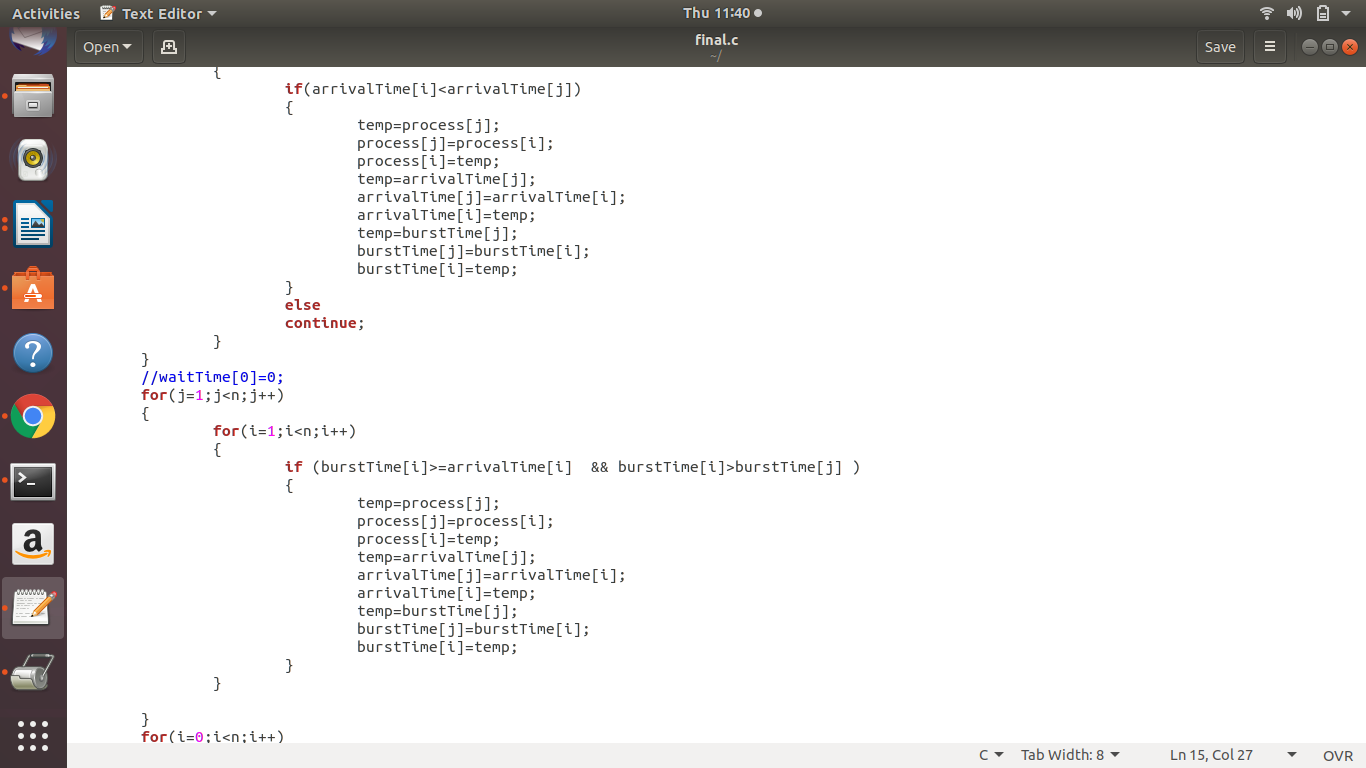
1. The scheduler checks the queue of the processes after every process terminates and time taken for checking and arranging the process according to the shortest job is 2 time

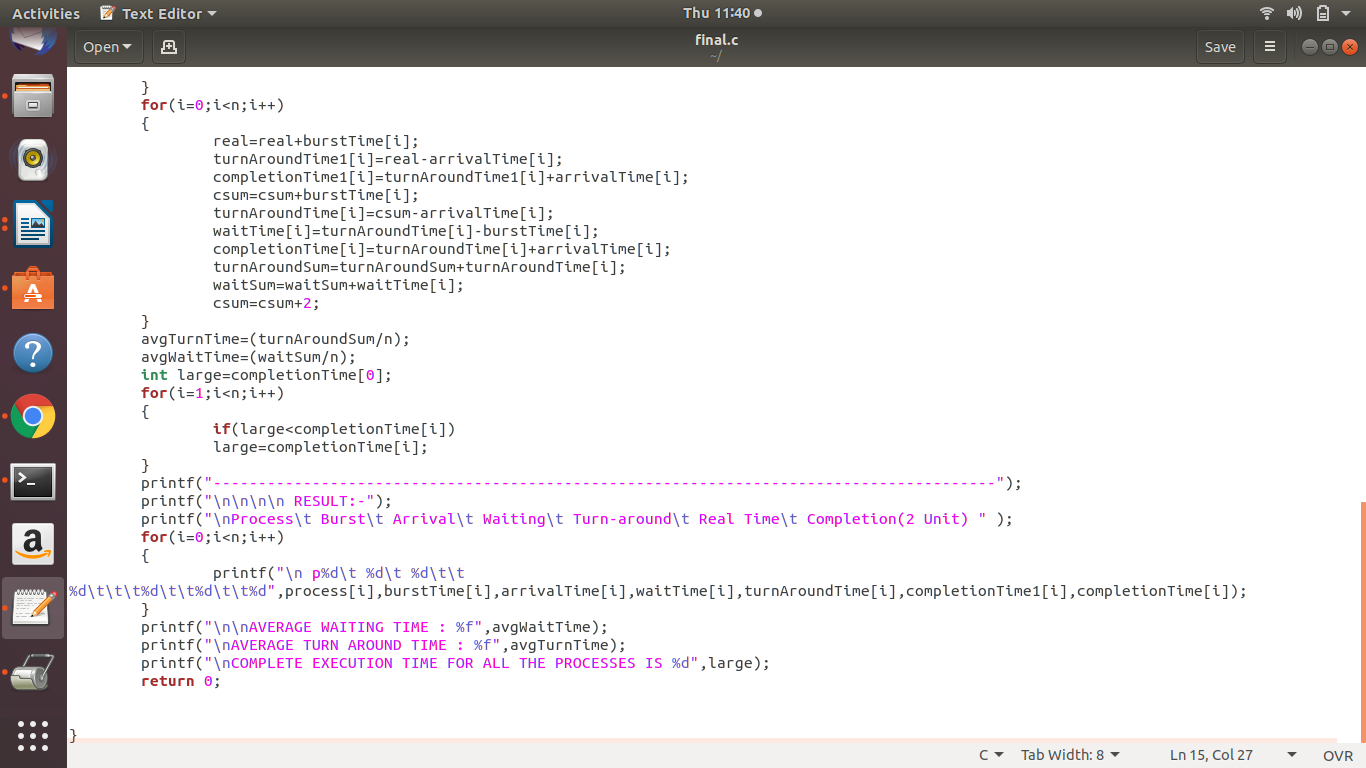


// logic so that every loop traverse and increment the burst time by 2 unit so that due to which it will change the completion time of each processes.

**Code snippet:**







**Complexity :**

For loop user entry: O(n);

Bubble sort for arrival time: O(n^2);

Bubble sort for burst time: O(n^2);

For loop in calculating arrival, waiting time and turn around time: O(n);

For loop for calculating total completion time of all jobs :O(n);

For loop to display the values : O(n);

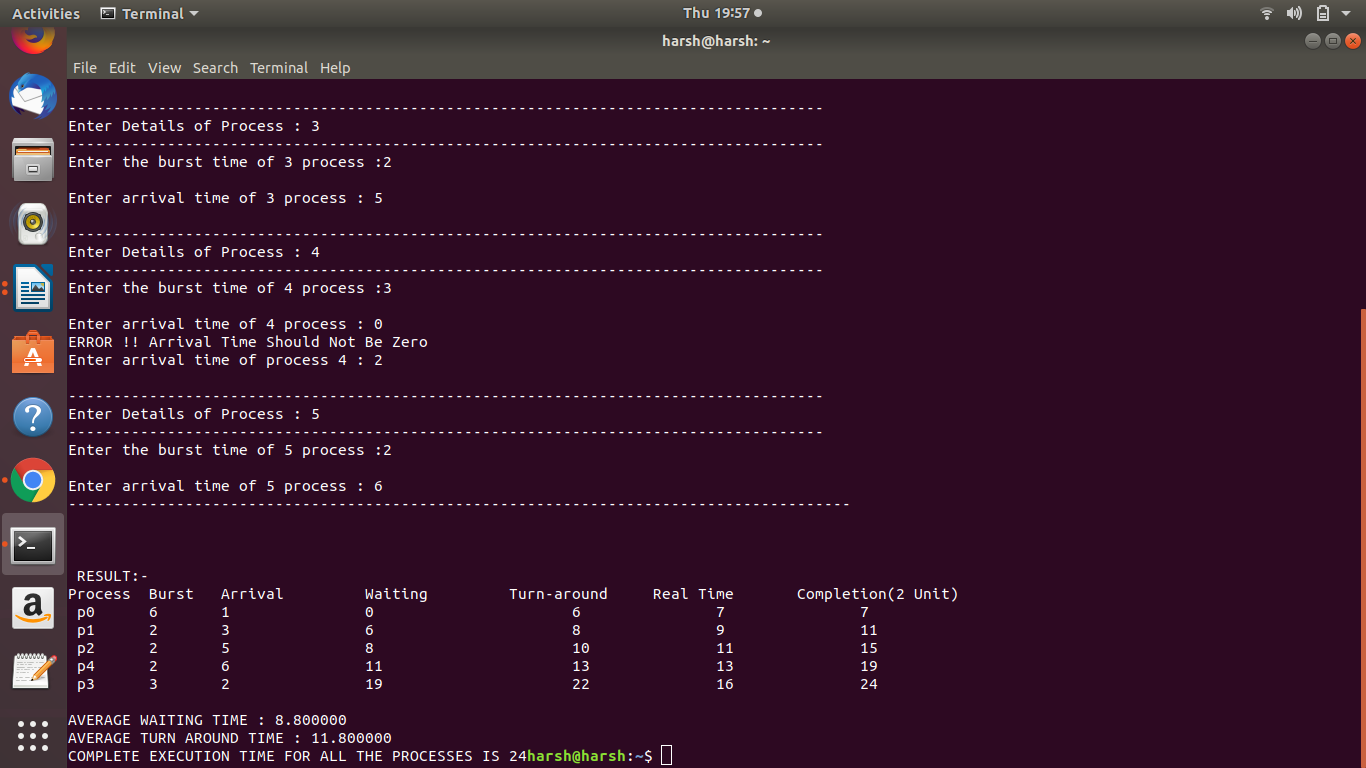
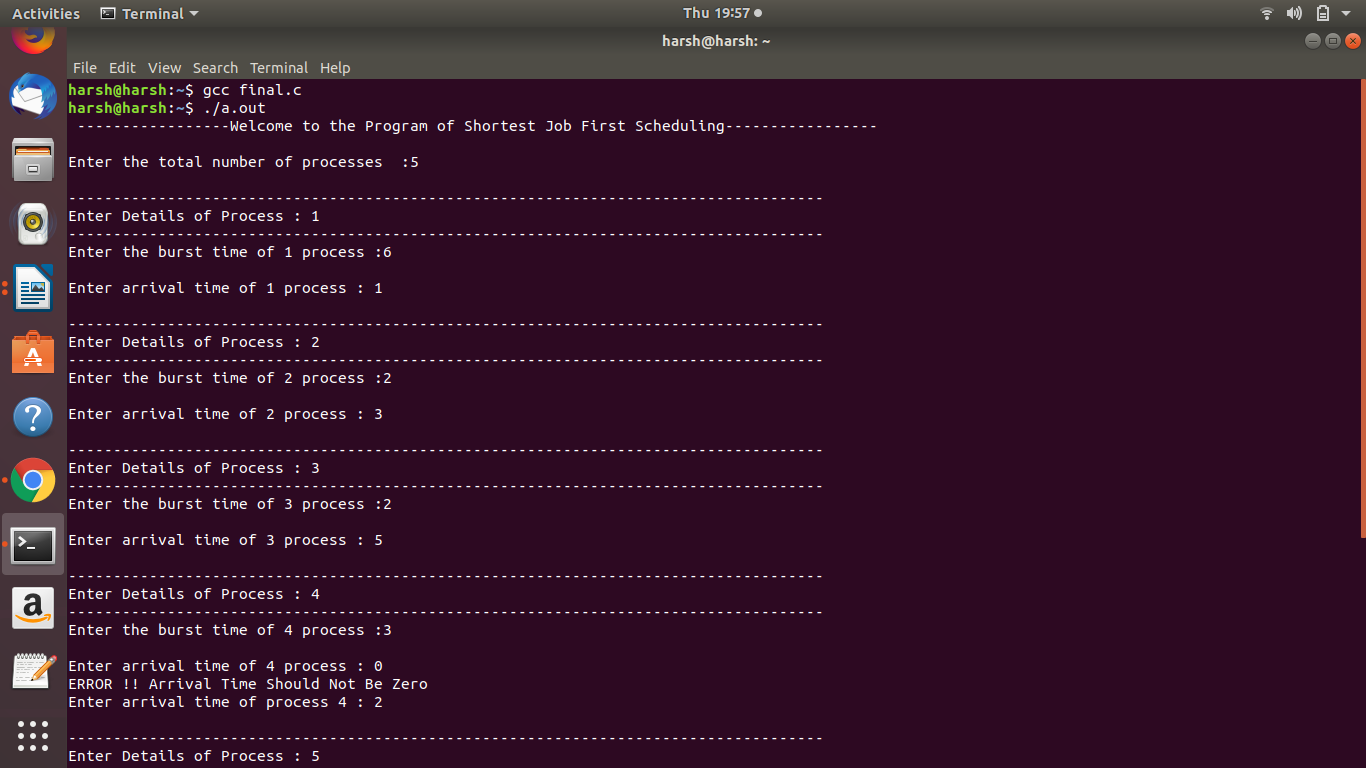
**Overall Complexity:**

O(n)+O(n^2)+O(n^2)+O(n)+O(n)+O(n) = O(n^2).

**Boundary conditions:**

* CPU will remain idle for 2 unit of time after every process is terminated .
* Arrival time of exactly one process should be ‘1’ and no process should have arrival time ‘0’

**Test Cases :**



In the above program when arrival time is given ‘0’ it prints error message and ask user to re enter the other arrival time than it is calculating waiting , turn around time , actual time taken by normal SJF .When CPU is no ideal for any unit of time and the completion time of processes when CPU is ideal for 2 unit of time after the execution of every process and at last it calculates the average waiting time , average turn around time and total completion time of all the process according to the 2 unit time (CPU ideal condition).

**GitHub Link**: [**https://github.com/harshputu/LPU-OS-Project**](https://github.com/harshputu/LPU-OS-Project)

The following code has been uploaded to the GitHub by making a repository named’ LPU-OS-Project ‘ and the five revisions were completed.

1st revision : <https://github.com/harshputu/LPU-OS-Project/blob/master/revision1.c>

2nd revision: <https://github.com/harshputu/LPU-OS-Project/blob/master/revision2.c>

3rd revision: [<https://github.com/harshputu/LPU-OS-Project/blob/master/revision3.c>](https://github.com/anushri0/OS/blob/master/3.txt)

4th revision: <https://github.com/harshputu/LPU-OS-Project/blob/master/revision4.c>

5th revision: <https://github.com/harshputu/LPU-OS-Project/blob/master/final.c>

=====================================================================