NAME- SUBHAJIT PAYRA   REG_NO- 11801422

EMAIL- SUBHAPAYRA.SP@GMAIL.COM

GITHUB LINK-
https://github.com/subhajitpayra/OS_task3_11801422

OPERATING SYSTEM

SUBMITTED BY

SUBHAJIT PAYRA

REG_NO- 11801422

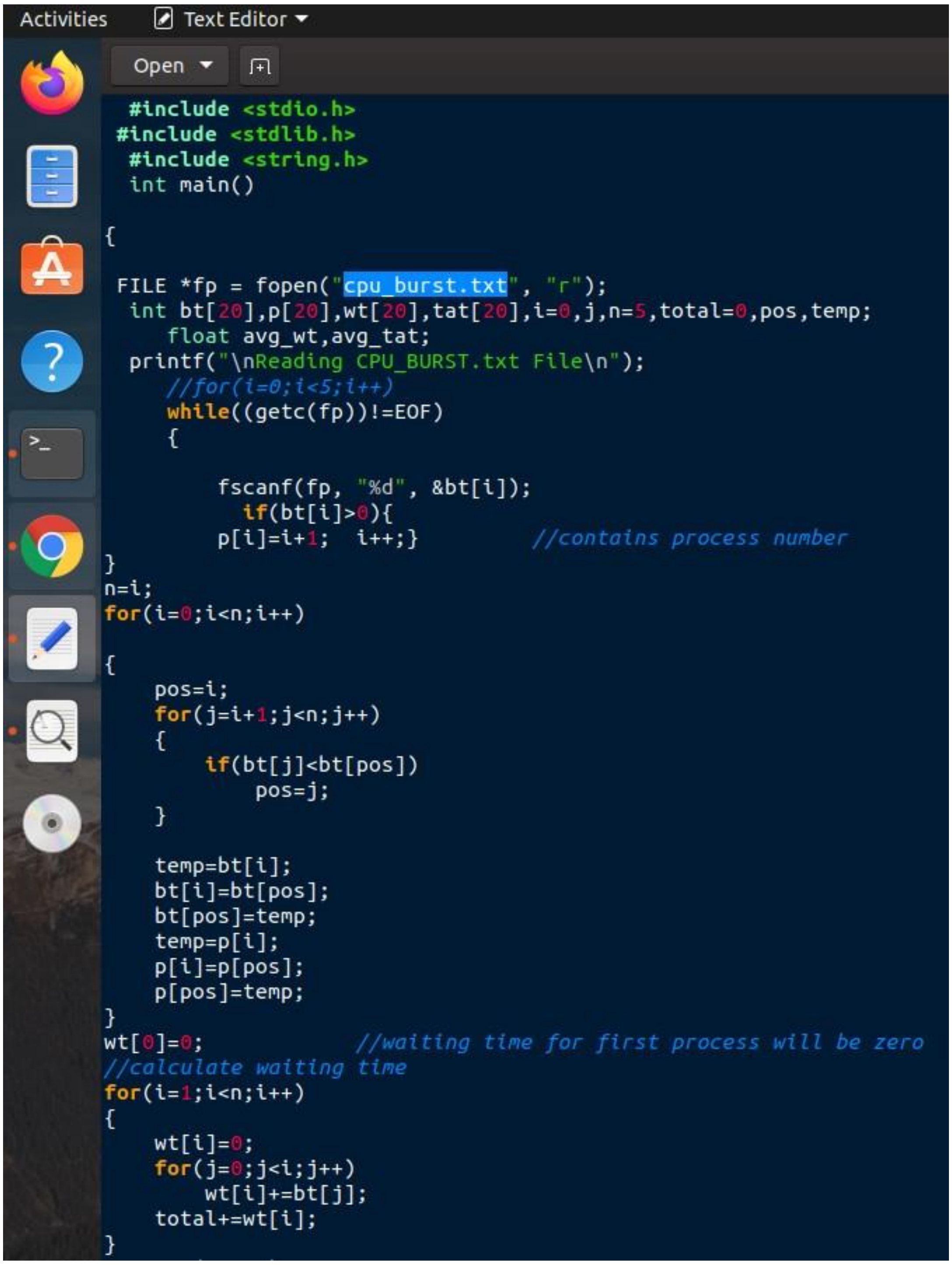
SECTION – K18SB

SUBMITTED TO

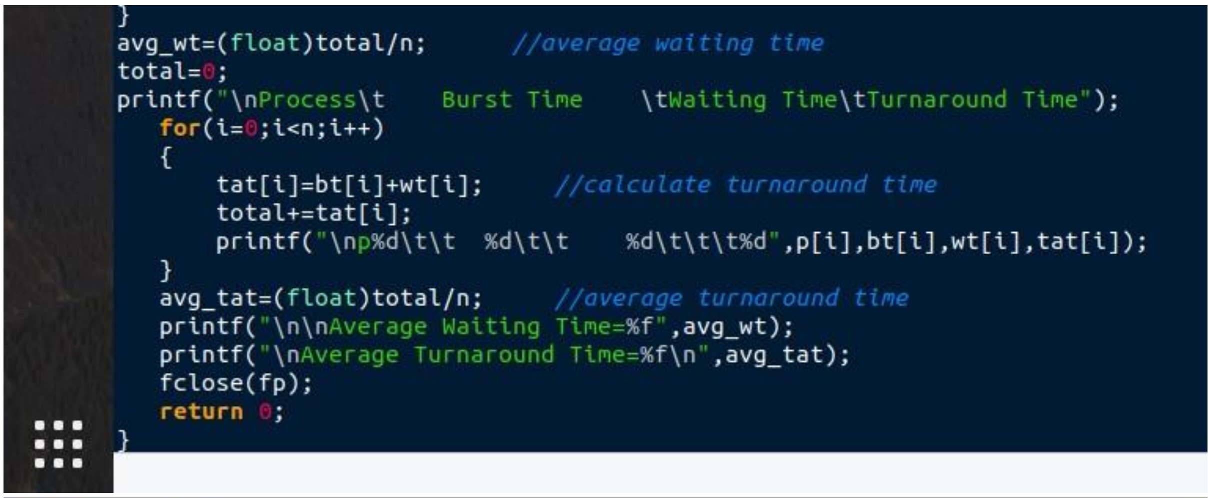
Mr. MANPREET SINGH

Q1. Write a program in C which reads input CPU bursts from a the first line of a text file named as CPU_BURST.txt. Validate the input numbers whether the numbers are positive intergers or not. Consider the numbers as CPU burst.If there are 5 positive integers in the first line of the text file then the program treat those argument as required CPU bust for P1, P2, P3, P4, and P5 process and calculate average waiting time and average turn around time. Consider used scheduling algorithm as SJF and same arrival time for all the processes.

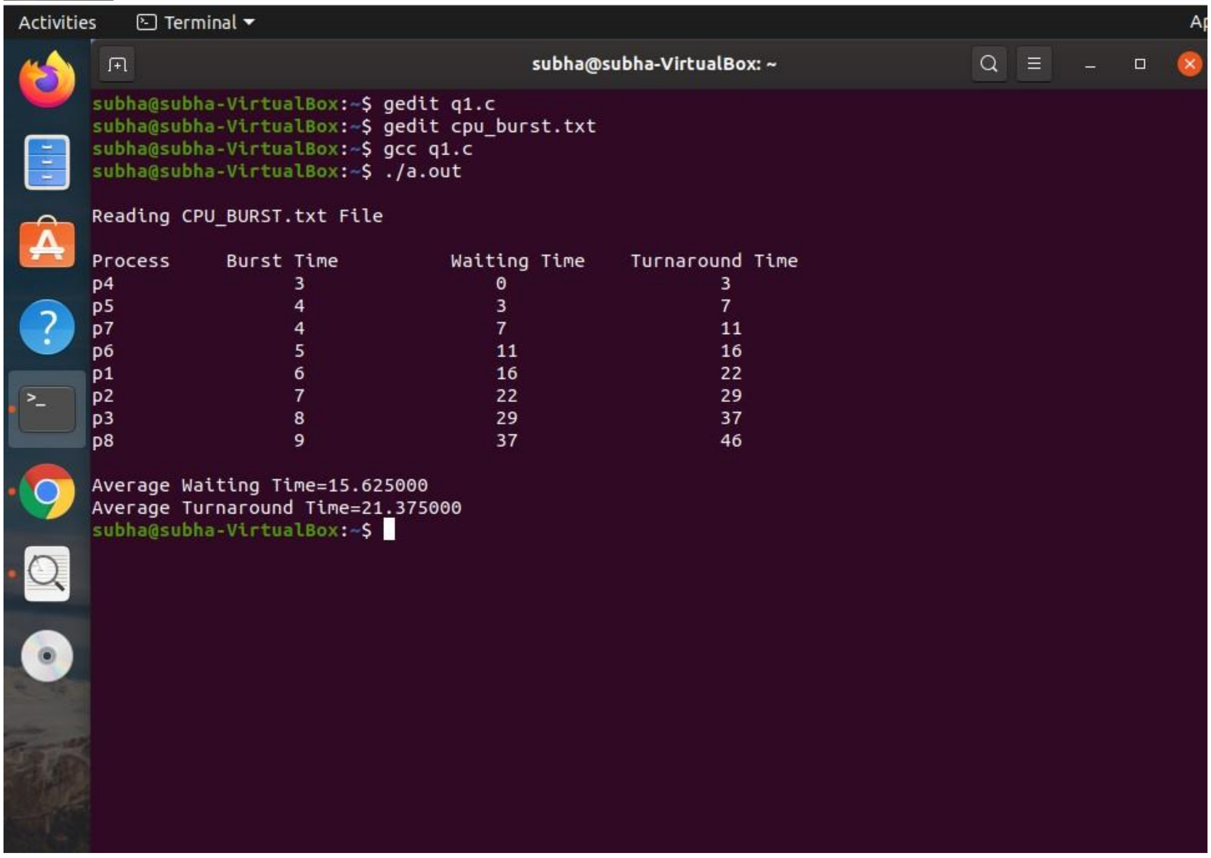
CODE:



```
  #include <stdio.h>
 #include <stdlib.h>
  #include <string.h>
  int main()

{

 FILE *fp = fopen("cpu_burst.txt", "r");
  int bt[20],p[20],wt[20],tat[20],i=0,j,n=5,total=0,pos,temp;
     float avg_wt,avg_tat;
  printf("\nReading CPU_BURST.txt File\n");
     //for(i=0;i<5;i++)
     while((getc(fp))!=EOF)
     {

         fscanf(fp, "%d", &bt[i]);
           if(bt[i]>0){
         p[i]=i+1;  i++;}         //contains process number
}
n=i;
for(i=0;i<n;i++)

{
    pos=i;
    for(j=i+1;j<n;j++)
    {
        if(bt[j]<bt[pos])
            pos=j;
    }

    temp=bt[i];
    bt[i]=bt[pos];
    bt[pos]=temp;
    temp=p[i];
    p[i]=p[pos];
    p[pos]=temp;
}
wt[0]=0;            //waiting time for first process will be zero
//calculate waiting time
for(i=1;i<n;i++)
{
    wt[i]=0;
    for(j=0;j<i;j++)
        wt[i]+=bt[j];
    total+=wt[i];
}
```



```
}
avg_wt=(float)total/n;      //average waiting time
total=0;
printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
   for(i=0;i<n;i++)
   {
       tat[i]=bt[i]+wt[i];     //calculate turnaround time
       total+=tat[i];
       printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
   }
   avg_tat=(float)total/n;     //average turnaround time
   printf("\n\nAverage Waiting Time=%f",avg_wt);
   printf("\nAverage Turnaround Time=%f\n",avg_tat);
   fclose(fp);
   return 0;
}
```



```
subha@subha-VirtualBox:~$ gedit q1.c
subha@subha-VirtualBox:~$ gedit cpu_burst.txt
subha@subha-VirtualBox:~$ gcc q1.c
subha@subha-VirtualBox:~$ ./a.out

Reading CPU_BURST.txt File

Process      Burst Time          Waiting Time    Turnaround Time
p4                3                   0                3
p5                4                   3                7
p7                4                   7                11
p6                5                   11               16
p1                6                   16               22
p2                7                   22               29
p3                8                   29               37
p8                9                   37               46

Average Waiting Time=15.625000
Average Turnaround Time=21.375000
subha@subha-VirtualBox:~$
```

Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN is a non-preemptive algorithm.

- Shortest Job first has the advantage of having a minimum average waiting time among all scheduling algorithms.
- It is a Greedy Algorithm.
- It may cause starvation if shorter processes keep coming. This problem can be solved using the concept of ageing.
- 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.

Algorithm:

1. Sort all the process according to the arrival time.
2. Then select that process which has minimum arrival time and minimum Burst time.
3. After completion of process make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum Burst time.

```
FILE *fp = fopen("cpu_burst.txt", "r"); // complexity is O(1)
  int bt[20],p[20],wt[20],tat[20],i=0,j,n=5,total=0,pos,temp; // complexity is O(1)
     float avg_wt,avg_tat;   // complexity is O(1)
  printf("\nReading CPU_BURST.txt File\n");   // complexity is O(1)
     //for(i=0;i<5;i++)
     while((getc(fp))!=EOF)   // complexity is O(f)  f=file size
     {

         fscanf(fp, "%d", &bt[i]);
           if(bt[i]>0){
         p[i]=i+1;  i++;}         //contains process number
}
n=i;
for(i=0;i<n;i++) // complexity is O(n)

{
    pos=i;
    for(j=i+1;j<n;j++)   // complexity is O(n*n)
    {
        if(bt[j]<bt[pos])
            pos=j;
    }

    temp=bt[i];
    bt[i]=bt[pos];
    bt[pos]=temp;
    temp=p[i];
    p[i]=p[pos];
    p[pos]=temp;
}
wt[0]=0;            //waiting time for first process will be zero
//calculate waiting time
for(i=1;i<n;i++)   // complexity is O(n)
{
    wt[i]=0;
    for(j=0;j<i;j++)
        wt[i]+=bt[j];
    total+=wt[i];
}
avg_wt=(float)total/n;      //average waiting time // complexity is O(1)
total=0;
printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
   for(i=0;i<n;i++) // complexity is O(1)
   {
       tat[i]=bt[i]+wt[i];     //calculate turnaround time
       total+=tat[i];
       printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
   }
   avg_tat=(float)total/n;     //average turnaround time
   printf("\n\nAverage Waiting Time=%f",avg_wt);
   printf("\nAverage Turnaround Time=%f\n",avg_tat); // complexity is O(1)
   fclose(fp);  // complexity is O(1)
   return 0;
}
```

Overall complexity is = O(n*n)

Description (constraints):

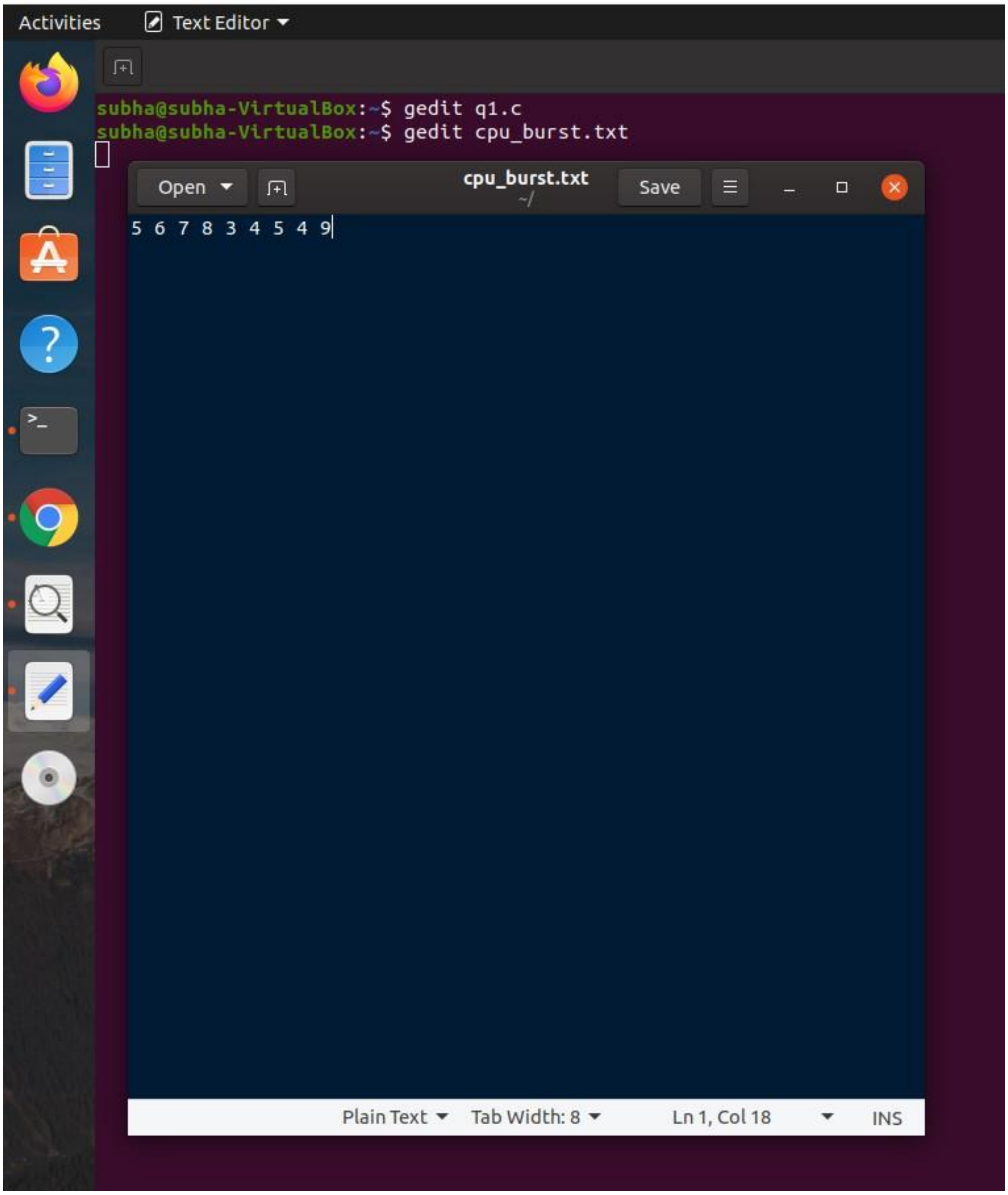
a program in C which reads input CPU bursts from a the first line of a text file named as CPU_BURST.txt.

```
  FILE *fp = fopen("cpu_burst.txt", "r");
    int bt[20],p[20],wt[20],tat[20],i=0,j,n=5,total=0,pos,temp;
       float avg_wt,avg_tat;
    printf("\nReading CPU_BURST.txt File\n");
```

Validate the input numbers whether the numbers are positive intergers or not.

```
       while((getc(fp))!=EOF)
       {

           fscanf(fp, "%d", &bt[i]);
             if(bt[i]>0){
           p[i]=i+1;  i++;}         //contains process number
  }
```

If there are 5 positive integers in the first line of the text file then the program treat those argument as required CPU bust for P1, P2, P3, P4, and P5 process



```
subha@subha-VirtualBox:~$ gedit q1.c
subha@subha-VirtualBox:~$ gedit cpu_burst.txt
```

```
5 6 7 8 3 4 5 4 9
```

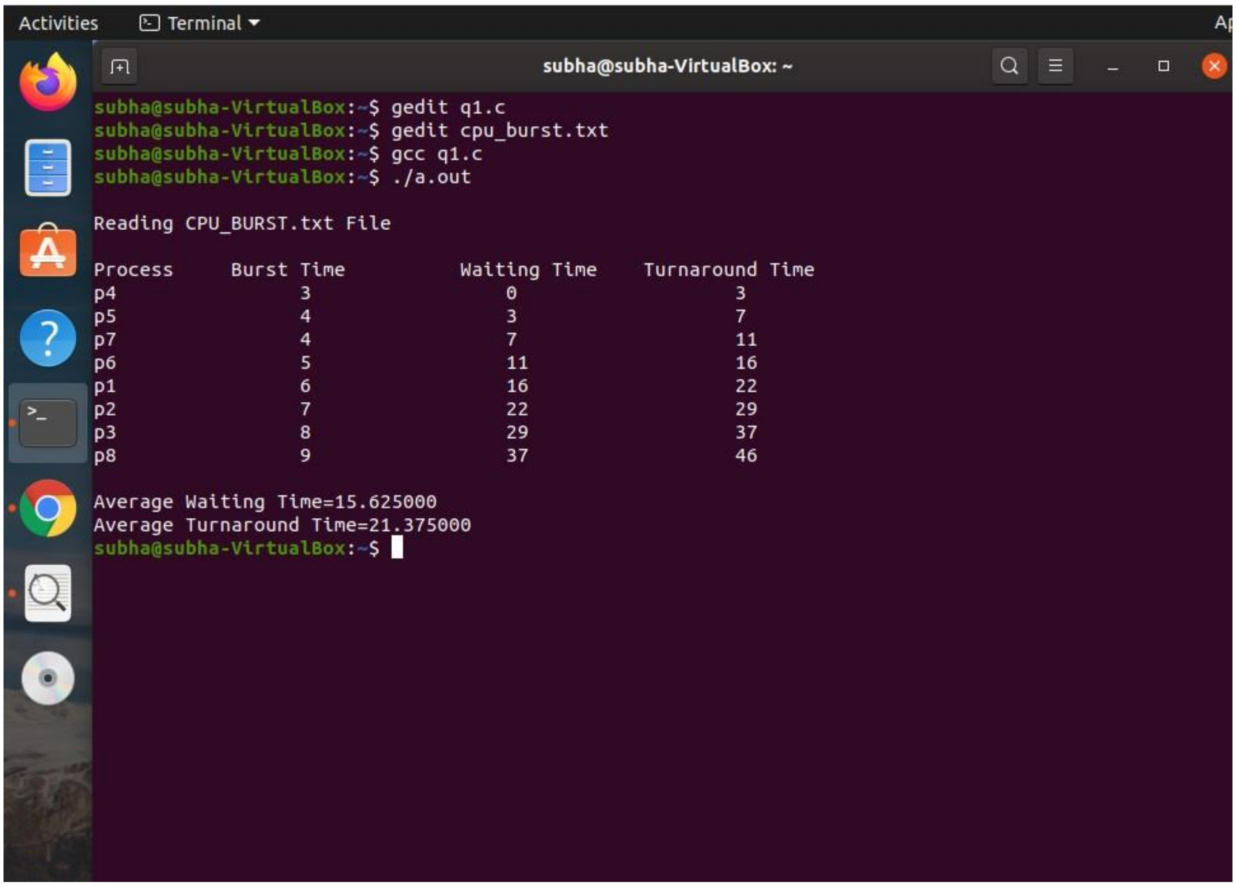
calculate average waiting time

```
wt[0]=0;            //waiting time for first process will be zero
//calculate waiting time
for(i=1;i<n;i++)
{
    wt[i]=0;
    for(j=0;j<i;j++)
        wt[i]+=bt[j];
    total+=wt[i];
}
avg_wt=(float)total/n;      //average waiting time
```

and average turn around time.

```
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];     //calculate turnaround time
    total+=tat[i];
    printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
}
avg_tat=(float)total/n;     //average turnaround time
```

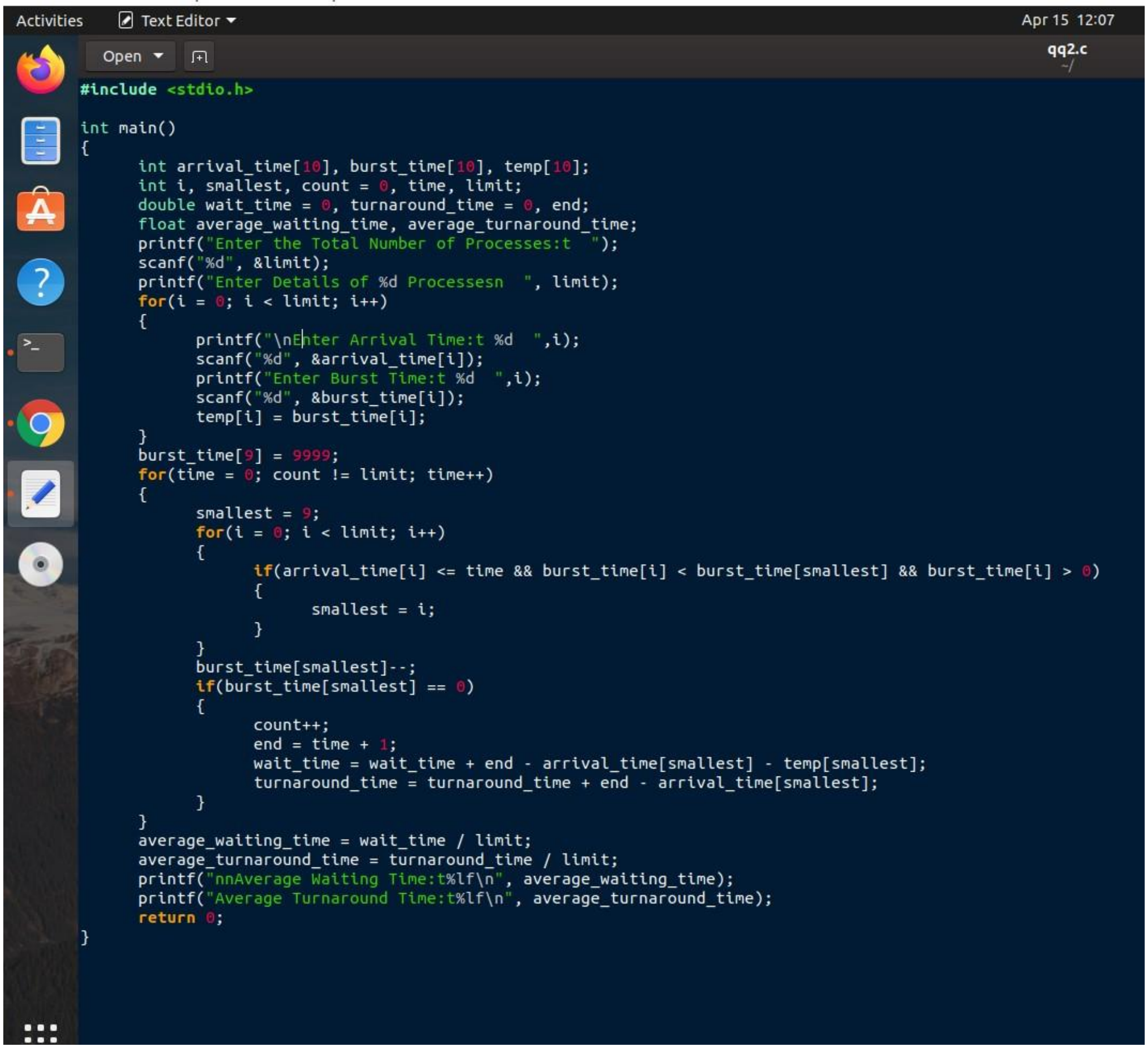
Code snippet:



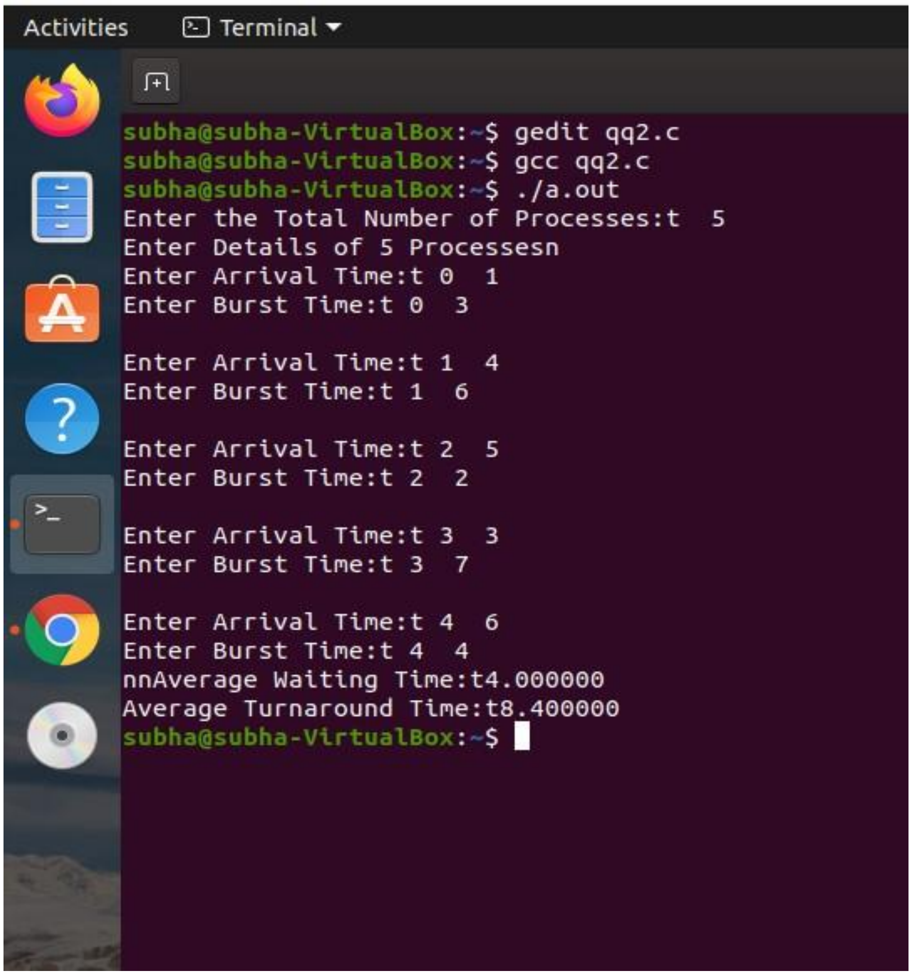
```
subha@subha-VirtualBox:~$ gedit q1.c
subha@subha-VirtualBox:~$ gedit cpu_burst.txt
subha@subha-VirtualBox:~$ gcc q1.c
subha@subha-VirtualBox:~$ ./a.out

Reading CPU_BURST.txt File

Process      Burst Time          Waiting Time     Turnaround Time
p4                3                   0                   3
p5                4                   3                   7
p7                4                   7                   11
p6                5                   11                  16
p1                6                   16                  22
p2                7                   22                  29
p3                8                   29                  37
p8                9                   37                  46

Average Waiting Time=15.625000
Average Turnaround Time=21.375000
subha@subha-VirtualBox:~$
```

Q2. 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 at same time.

CODE:



```c
#include <stdio.h>

int main()
{
        int arrival_time[10], burst_time[10], temp[10];
        int i, smallest, count = 0, time, limit;
        double wait_time = 0, turnaround_time = 0, end;
        float average_waiting_time, average_turnaround_time;
        printf("Enter the Total Number of Processes:t  ");
        scanf("%d", &limit);
        printf("Enter Details of %d Processesn  ", limit);
        for(i = 0; i < limit; i++)
        {
                printf("\nEnter Arrival Time:t %d  ",i);
                scanf("%d", &arrival_time[i]);
                printf("Enter Burst Time:t %d  ",i);
                scanf("%d", &burst_time[i]);
                temp[i] = burst_time[i];
        }
        burst_time[9] = 9999;
        for(time = 0; count != limit; time++)
        {
                smallest = 9;
                for(i = 0; i < limit; i++)
                {
                        if(arrival_time[i] <= time && burst_time[i] < burst_time[smallest] && burst_time[i] > 0)
                        {
                                smallest = i;
                        }
                }
                burst_time[smallest]--;
                if(burst_time[smallest] == 0)
                {
                        count++;
                        end = time + 1;
                        wait_time = wait_time + end - arrival_time[smallest] - temp[smallest];
                        turnaround_time = turnaround_time + end - arrival_time[smallest];
                }
        }
        average_waiting_time = wait_time / limit;
        average_turnaround_time = turnaround_time / limit;
        printf("nnAverage Waiting Time:t%lf\n", average_waiting_time);
        printf("Average Turnaround Time:t%lf\n", average_turnaround_time);
        return 0;
}
```



```
subha@subha-VirtualBox:~$ gedit qq2.c
subha@subha-VirtualBox:~$ gcc qq2.c
subha@subha-VirtualBox:~$ ./a.out
Enter the Total Number of Processes:t  5
Enter Details of 5 Processesn
Enter Arrival Time:t 0  1
Enter Burst Time:t 0  3

Enter Arrival Time:t 1  4
Enter Burst Time:t 1  6

Enter Arrival Time:t 2  5
Enter Burst Time:t 2  2

Enter Arrival Time:t 3  3
Enter Burst Time:t 3  7

Enter Arrival Time:t 4  6
Enter Burst Time:t 4  4
nnAverage Waiting Time:t4.000000
Average Turnaround Time:t8.400000
subha@subha-VirtualBox:~$ 
```

The Solution of this problem can be done by primitive SJF. In the Shortest Remaining Time First (SRTF) scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time.

1- Short processes are handled very quickly.
2- The system also requires very little overhead since it only makes a decision when a process completes or a new process is added.
3- When a new process is added the algorithm only needs to compare the currently executing process with the new process, ignoring all other processes currently waiting to execute.

Algorithm:

```
1- Traverse until all process gets completely
   executed.
   a) Find process with minimum remaining time at
     every single time lap.
   b) Reduce its time by 1.
   c) Check if its remaining time becomes 0
```

```
 d) Increment the counter of process completion.
 e) Completion time of current process =
   current_time +1;
 e) Calculate waiting time for each completed
   process.
 wt[i]= Completion time - arrival_time-burst_time
 f)Increment time lap by one.
2- Find turnaround time (waiting_time+burst_time).
```

```
#include <stdio.h>

int main()
{
    int arrival_time[10], burst_time[10], temp[10]; // Complexity is O(1)
    int i, smallest, count = 0, time, limit; // Complexity is O(1)
    double wait_time = 0, turnaround_time = 0, end; // Complexity is O(1)
    float average_waiting_time, average_turnaround_time; // Complexity is O(1)
    printf("Enter the Total Number of Processes:t  ");// Complexity is O(1)
    scanf("%d", &limit); // Complexity is O(1)
    printf("Enter Details of %d Processesn  ", limit); // Complexity is O(1)
    for(i = 0; i < limit; i++) // Complexity is O(n)
    {
        printf("\nEnter Arrival Time:t %d  ",i);
        scanf("%d", &arrival_time[i]);
        printf("Enter Burst Time:t %d  ",i);
        scanf("%d", &burst_time[i]);
        temp[i] = burst_time[i];
    }
    burst_time[9] = 9999;
    for(time = 0; count != limit; time++) // Complexity is O(n)
    {
        smallest = 9;
        for(i = 0; i < limit; i++)// Complexity is O(n*n)
        {
            if(arrival_time[i] <= time && burst_time[i] < burst_time[smallest] &&
burst_time[i] > 0) // Complexity is O(n*n)
            {
                smallest = i;
            }
        }
        burst_time[smallest]--;
        if(burst_time[smallest] == 0)
        {
            count++;
```

```
                    end = time + 1;
                    wait_time = wait_time + end - arrival_time[smallest] - temp[smallest];
                    turnaround_time = turnaround_time + end - arrival_time[smallest];
                }
            }
            average_waiting_time = wait_time / limit;  // Complexity is O(1)
            average_turnaround_time = turnaround_time / limit; // Complexity is O(1)
            printf("nnAverage Waiting Time:t%lf\n", average_waiting_time);
        // Complexity is O(1)
            printf("Average Turnaround Time:t%lf\n", average_turnaround_time);
        // Complexity is O(1)
            return 0;
        }
```

Overall Complexity O(n*n)

Description (constraints):

A uniprocessor system has n number of CPU intensive processes, each process has its own requirement of CPU burst.

```
int arrival_time[10], burst_time[10], temp[10];

    int i, smallest, count = 0, time, limit;

    double wait_time = 0, turnaround_time = 0, end;

    float average_waiting_time, average_turnaround_time;

    printf("Enter the Total Number of Processes:t  ");

    scanf("%d", &limit);

    printf("Enter Details of %d Processesn  ", limit);

    for(i = 0; i < limit; i++)

    {

        printf("\nEnter Arrival Time:t %d  ",i);

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

        printf("Enter Burst Time:t %d  ",i);

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

        temp[i] = burst_time[i];

    }
```

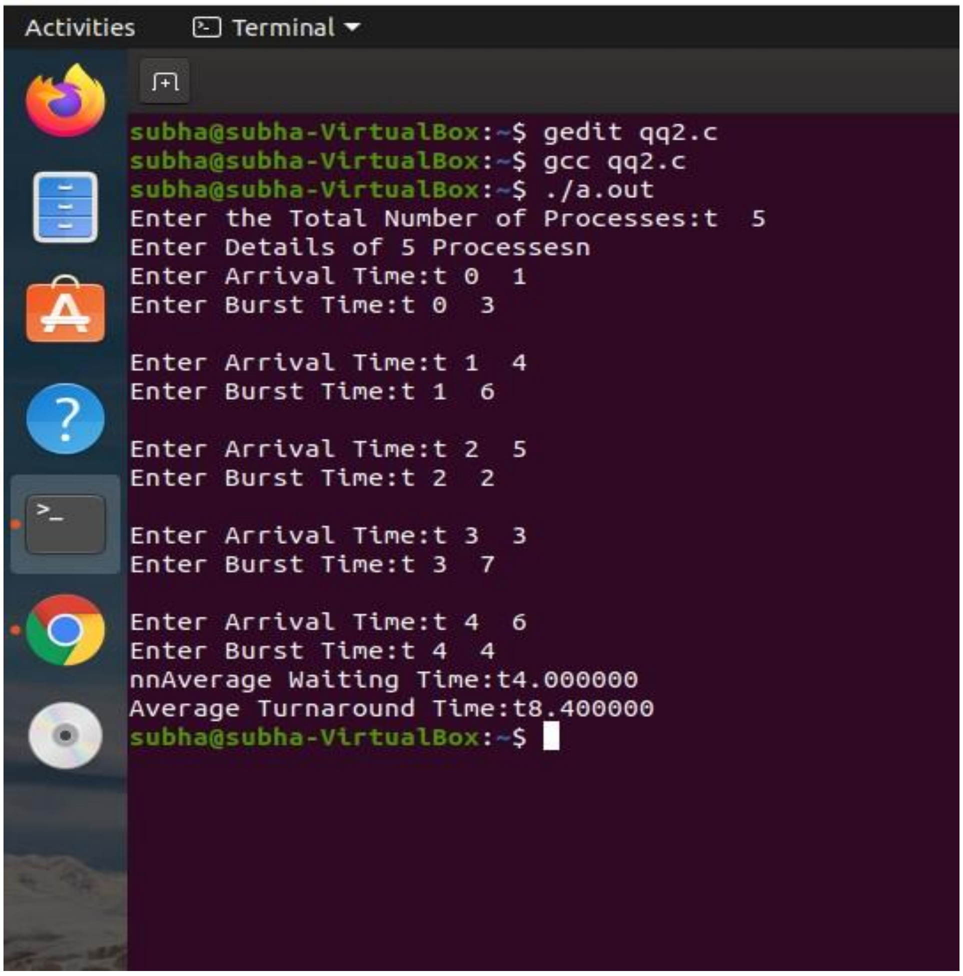
The process with lowest CPU burst is given the highest priority. A late arriving higher priority process can preempt a currently running process with lower priority. Simulate a scheduler that is scheduling the processes in such a way that higher priority process is never starved due to the execution of lower priority process.

```
for(time = 0; count != limit; time++)
    {
        smallest = 9;
        for(i = 0; i < limit; i++)
        {
            if(arrival_time[i] <= time && burst_time[i] < burst_time[smallest] && burst_time[i] > 0)
            {
                smallest = i;
            }
        }
```

its average waiting timeand average turnaround time if no two processes are arriving at same time.

```
    burst_time[smallest]--;
            if(burst_time[smallest] == 0)
            {
                count++;
                end = time + 1;
                wait_time = wait_time + end - arrival_time[smallest] - temp[smallest];
                turnaround_time = turnaround_time + end - arrival_time[smallest];
            }
        }
        average_waiting_time = wait_time / limit;
        average_turnaround_time = turnaround_time / limit;
        printf("nnAverage Waiting Time:t%lf\n", average_waiting_time);
        printf("Average Turnaround Time:t%lf\n", average_turnaround_time);
        return 0;
```

Code snippet(Output):



```
subha@subha-VirtualBox:~$ gedit qq2.c
subha@subha-VirtualBox:~$ gcc qq2.c
subha@subha-VirtualBox:~$ ./a.out
Enter the Total Number of Processes:t  5
Enter Details of 5 Processesn
Enter Arrival Time:t 0  1
Enter Burst Time:t 0  3

Enter Arrival Time:t 1  4
Enter Burst Time:t 1  6

Enter Arrival Time:t 2  5
Enter Burst Time:t 2  2

Enter Arrival Time:t 3  3
Enter Burst Time:t 3  7

Enter Arrival Time:t 4  6
Enter Burst Time:t 4  4
nnAverage Waiting Time:t4.000000
Average Turnaround Time:t8.400000
subha@subha-VirtualBox:~$ 
```