OPERATING SYSTEMS

ASSESSMENT-5

SIDDHI SINGH
17BIT0028

QUESTION:

a) Consider a corporate hospital where we have n number of patients waiting for consultation. The amount of time required to serve a patient may vary, say 10 to 30 minutes. If a patient arrives with an emergency,he /she should be attended immediately before other patients, which may increase the waiting time of other patients. If you are given this problem with the following algorithms how would you devise an effective scheduling so that it optimizes the overall performance such as minimizing the waiting time of
all patients. [Single queue or multi-level queue can be used].
• Consider the availability of single and multiple doctors
• Assign top priority for patients with emergency case, women, children, elders, and youngsters.
• Patients coming for review may take less time than others. This can be taken into account while using SJF.
a. FCFS
b. SJF (primitive and non-pre-emptive)

SOLUTION:

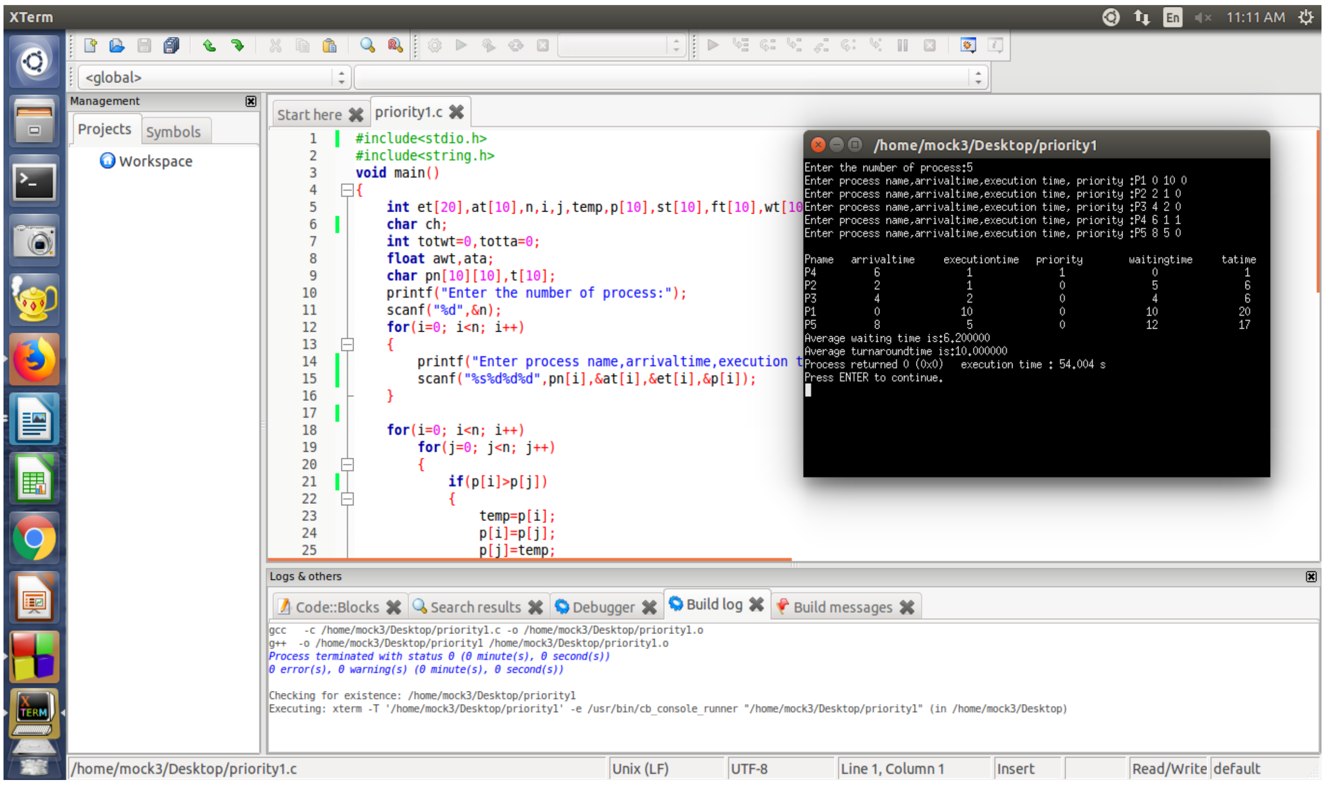
For the given problem I'd choose priority preemptive scheduling algorithm. If there is no emergency the patients will be having same priority and would be executed by FCFS algorithm and if there's any emergency the patient will be given more priority and will be executed first i.e., for normal patient **priority=0** and in emergency **priority=1**

CODE:

```
#include<stdio.h>
#include<string.h>
void main()
{
   int et[20],at[10],n,i,j,temp,p[10],st[10],ft[10],wt[10],ta[10];
   char ch;
   int totwt=0,totta=0;
   float awt,ata;
   char pn[10][10],t[10];
   printf("Enter the number of process:");
   scanf("%d",&n);
```

```
for(i=0; i<n; i++)
{
    printf("Enter process name,arrivaltime,execution time, priority :");
    scanf("%s%d%d%d",pn[i],&at[i],&et[i],&p[i]);
}

for(i=0; i<n; i++)
    for(j=0; j<n; j++)
    {
        if(p[i]>p[j])
        {
            temp=p[i];
            p[i]=p[j];
            p[j]=temp;
            temp=at[i];
            at[i]=at[j];
            at[j]=temp;
            temp=et[i];
            et[i]=et[j];
            et[j]=temp;
            strcpy(t,pn[i]);
            strcpy(pn[i],pn[j]);
            strcpy(pn[j],t);
        }
    }
for(i=0; i<n; i++)

{

    if(i==0)
    {
        st[i]=at[i];
        wt[i]=st[i]-at[i];
        ft[i]=st[i]+et[i];
        ta[i]=ft[i]-at[i];
    }
    else
    {
        st[i]=ft[i-1];
        wt[i]=st[i]-at[i];
        ft[i]=st[i]+et[i];
        ta[i]=ft[i]-at[i];
    }
    totwt+=wt[i];
    totta+=ta[i];
}
awt=(float)totwt/n;
ata=(float)totta/n;
printf("\nPname\tarrivaltime\texecutiontime\tpriority\twaitingtime\ttatime");
for(i=0; i<n; i++)
    printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],p[i],wt[i],ta[i]);
printf("\nAverage waiting time is:%f",awt);
```

```
    printf("\nAverage turnaroundtime is:%f",ata);
}
```



```
#include<stdio.h>
#include<string.h>
void main()
{
    int et[20],at[10],n,i,j,temp,p[10],st[10],ft[10],wt[10
    char ch;
    int totwt=0,totta=0;
    float awt,ata;
    char pn[10][10],t[10];
    printf("Enter the number of process:");
    scanf("%d",&n);
    for(i=0; i<n; i++)
    {
        printf("Enter process name,arrivaltime,execution t
        scanf("%s%d%d%d",pn[i],&at[i],&et[i],&p[i]);
    }

    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
        {
            if(p[i]>p[j])
            {
                temp=p[i];
                p[i]=p[j];
                p[j]=temp;
```

```
Enter the number of process:5
Enter process name,arrivaltime,execution time, priority :P1 0 10 0
Enter process name,arrivaltime,execution time, priority :P2 2 1 0
Enter process name,arrivaltime,execution time, priority :P3 4 2 0
Enter process name,arrivaltime,execution time, priority :P4 6 1 1
Enter process name,arrivaltime,execution time, priority :P5 8 5 0

Pname   arrivaltime     executiontime   priority        waitingtime     tatime
P4          6                1              1                0              1
P2          2                1              0                5              6
P3          4                2              0                4              6
P1          0               10              0               10             20
P5          8                5              0               12             17
Average waiting time is:6.200000
Average turnaroundtime is:10.000000
Process returned 0 (0x0)   execution time : 54.004 s
Press ENTER to continue.
```

```
gcc   -c /home/mock3/Desktop/priority1.c -o /home/mock3/Desktop/priority1.o
g++  -o /home/mock3/Desktop/priority1 /home/mock3/Desktop/priority1.o
Process terminated with status 0 (0 minute(s), 0 second(s))
0 error(s), 0 warning(s) (0 minute(s), 0 second(s))

Checking for existence: /home/mock3/Desktop/priority1
Executing: xterm -T '/home/mock3/Desktop/priority1' -e /usr/bin/cb_console_runner "/home/mock3/Desktop/priority1" (in /home/mock3/Desktop)
```

<u>OUTPUT:</u>

b) Many CPU-scheduling algorithms are parameterized. For example, the RR algorithm requires a parameter to indicate the time slice. Multilevel feedback queues require parameters to define the number of queues, the scheduling algorithm for each queue, the criteria used to move processes between queues, and so on. These algorithms are thus really sets of algorithms (for example, the set of RR algorithms for all time slices, and so on). One set of algorithms may include another (for example, the FCFS algorithm is the RR algorithm with an infinite time quantum). What (if any) relation holds between the following pairs of algorithm sets? Implement the below mentioned algorithms and determine the efficiency of each algorithm. (Assume your own data for input) 1. Priority and SJF 2. Multilevel feedback queues and FCFS 3. Priority and FCFS 4. RR and SJF

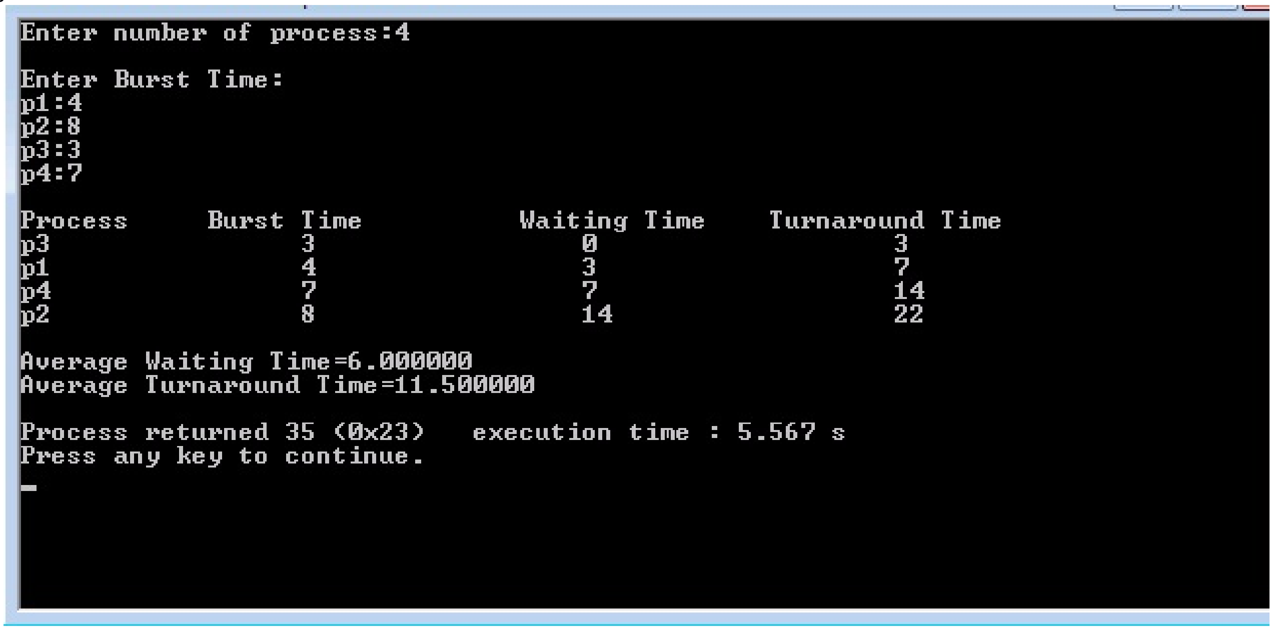
SOLUTION:

1. Priority and SJF

The shortest job has the highest priority

SJF:

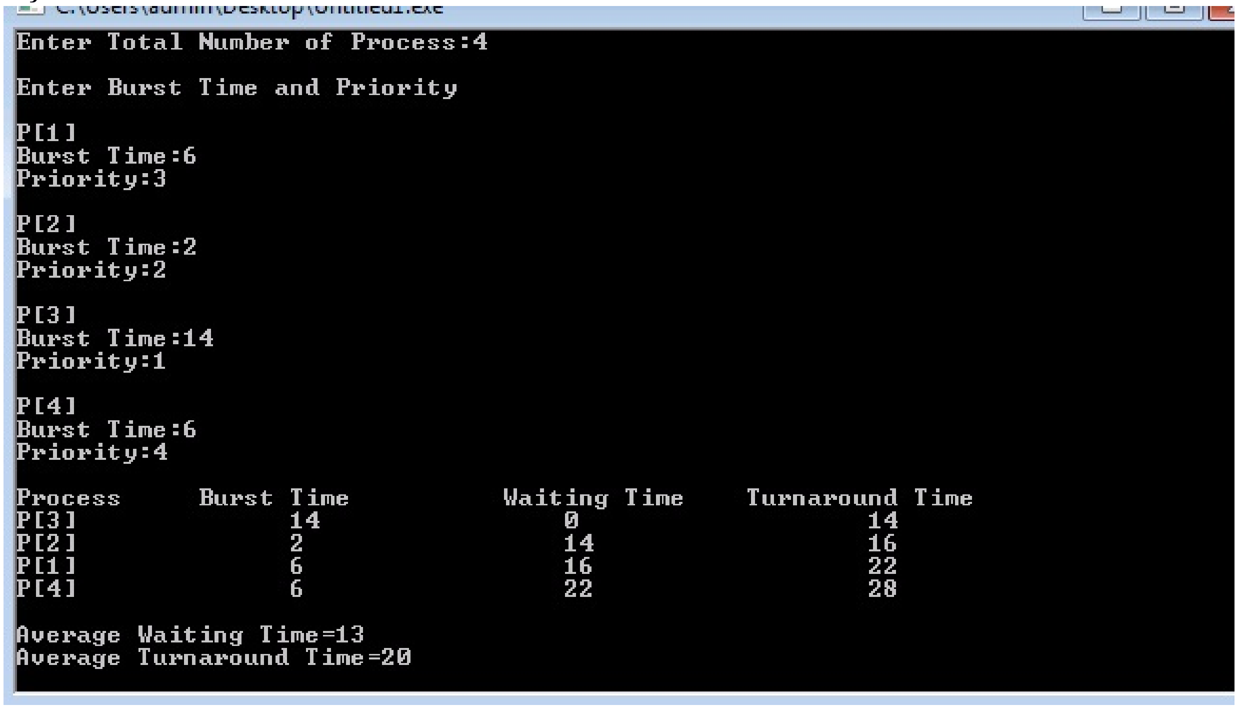
```
#include<stdio.h>

void main()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf("Enter number of process:");
    scanf("%d",&n);

    printf("\nEnter Burst Time:\n");
    for(i=0;i<n;i++)
    {
        printf("p%d:",i+1);
        scanf("%d",&bt[i]);
        p[i]=i+1;           //contains process number
    }

    //sorting burst time in ascending order using selection sort
    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);
}
```



```
Enter number of process:4

Enter Burst Time:
p1:4
p2:8
p3:3
p4:7

Process     Burst Time          Waiting Time    Turnaround Time
p3                3                 0                   3
p1                4                 3                   7
p4                7                 7                   14
p2                8                 14                  22

Average Waiting Time=6.000000
Average Turnaround Time=11.500000

Process returned 35 (0x23)   execution time : 5.567 s
Press any key to continue.
```

PRIORITY:

```
#include<stdio.h>

int main()
{
   int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
   printf("Enter Total Number of Process:");
   scanf("%d",&n);

   printf("\nEnter Burst Time and Priority\n");
   for(i=0;i<n;i++)
   {
      printf("\nP[%d]\n",i+1);
      printf("Burst Time:");
      scanf("%d",&bt[i]);
      printf("Priority:");
      scanf("%d",&pr[i]);
      p[i]=i+1;           //contains process number
   }

   //sorting burst time, priority and process number in ascending order using selection
sort
   for(i=0;i<n;i++)
   {
      pos=i;
      for(j=i+1;j<n;j++)
      {
         if(pr[j]<pr[pos])
            pos=j;
      }

      temp=pr[i];
      pr[i]=pr[pos];
      pr[pos]=temp;

      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 is 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=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=total/n;     //average turnaround time
    printf("\n\nAverage Waiting Time=%d",avg_wt);
    printf("\nAverage Turnaround Time=%d\n",avg_tat);

    return 0;
}
```


```
Enter Total Number of Process:4

Enter Burst Time and Priority

P[1]
Burst Time:6
Priority:3

P[2]
Burst Time:2
Priority:2

P[3]
Burst Time:14
Priority:1

P[4]
Burst Time:6
Priority:4

Process     Burst Time          Waiting Time     Turnaround Time
P[3]              14                0                    14
P[2]              2                 14                   16
P[1]              6                 16                   22
P[4]              6                 22                   28

Average Waiting Time=13
Average Turnaround Time=20
```

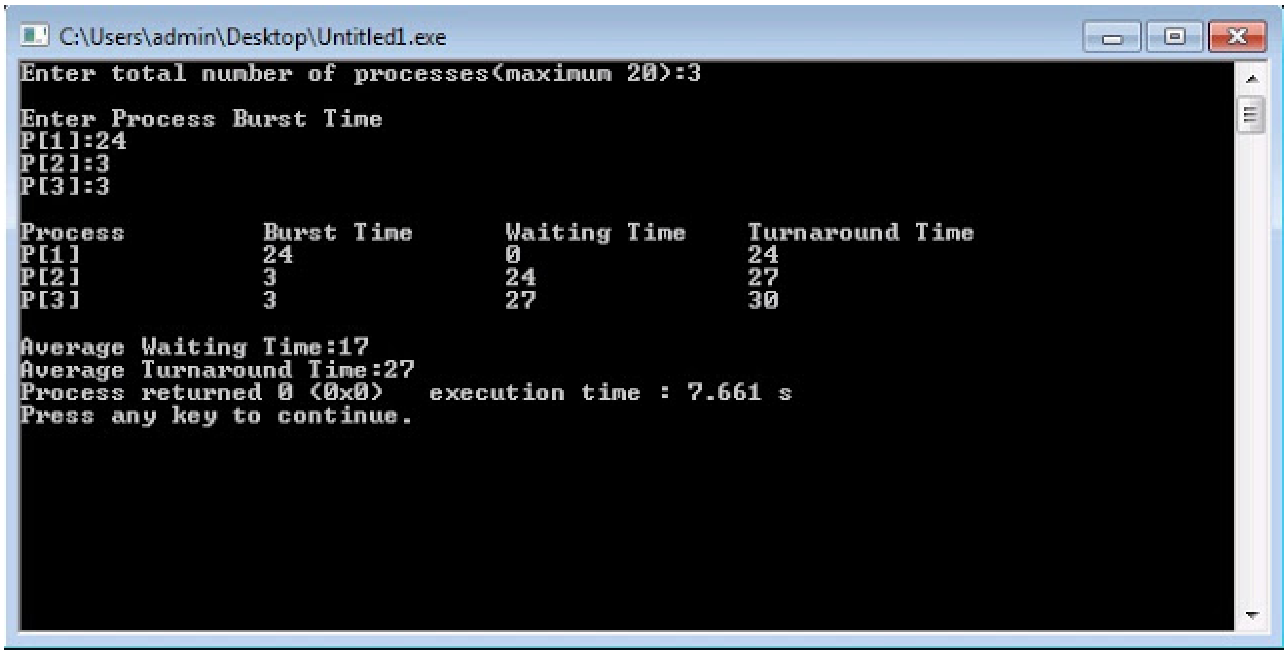
2. Multilevel feedback queues and FCFS

The lowest level of MLFQ is FCFS.

<u>FCFC:</u>

```
#include<stdio.h>

int main()
{
   int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
   printf("Enter total number of processes(maximum 20):");
   scanf("%d",&n);

   printf("\nEnter Process Burst Time\n");
   for(i=0;i<n;i++)
   {
      printf("P[%d]:",i+1);
      scanf("%d",&bt[i]);
   }

   wt[0]=0;    //waiting time for first process is 0

   //calculating waiting time
   for(i=1;i<n;i++)
   {
      wt[i]=0;
      for(j=0;j<i;j++)
         wt[i]+=bt[j];
   }

   printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

   //calculating turnaround time
   for(i=0;i<n;i++)
   {
      tat[i]=bt[i]+wt[i];
      avwt+=wt[i];
      avtat+=tat[i];
      printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
   }

   avwt/=i;
```

```
    avtat/=i;
    printf("\n\nAverage Waiting Time:%d",avwt);
    printf("\nAverage Turnaround Time:%d",avtat);

    return 0;
}
```

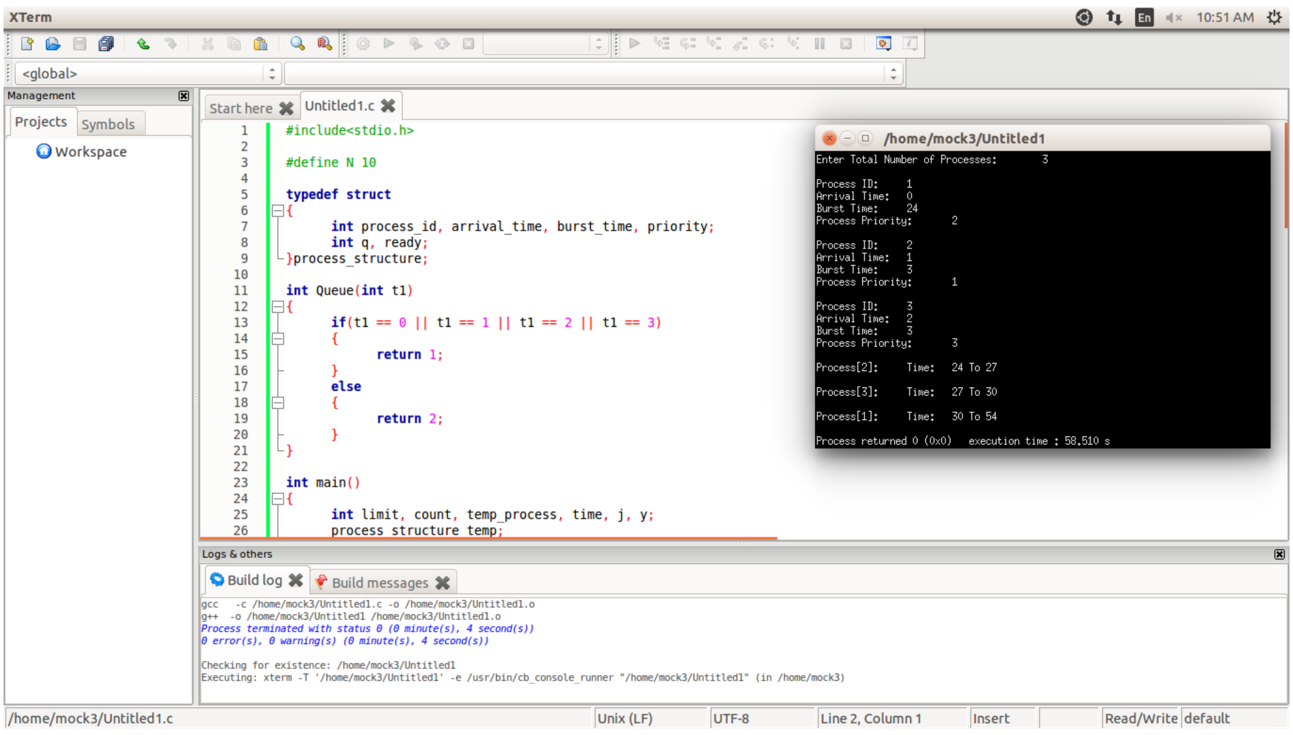


```
Enter total number of processes(maximum 20):3

Enter Process Burst Time
P[1]:24
P[2]:3
P[3]:3

Process         Burst Time      Waiting Time    Turnaround Time
P[1]            24              0               24
P[2]            3               24              27
P[3]            3               27              30

Average Waiting Time:17
Average Turnaround Time:27
Process returned 0 (0x0)   execution time : 7.661 s
Press any key to continue.
```

MLFQ:

```
#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:\t");
    scanf("%d", &limit);
    process_structure process[limit];
    for(count = 0; count < limit; count++)
    {
        printf("\nProcess ID:\t");
        scanf("%d", &process[count].process_id);
        printf("Arrival Time:\t");
        scanf("%d", &process[count].arrival_time);
        printf("Burst Time:\t");
        scanf("%d", &process[count].burst_time);
        printf("Process Priority:\t");
        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]:\tTime:\t%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;
}
```



```c
#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;
```

```
Enter Total Number of Processes:        3

Process ID:     1
Arrival Time:   0
Burst Time:     24
Process Priority:       2

Process ID:     2
Arrival Time:   1
Burst Time:     3
Process Priority:       1

Process ID:     3
Arrival Time:   2
Burst Time:     3
Process Priority:       3

Process[2]:     Time:   24 To 27

Process[3]:     Time:   27 To 30

Process[1]:     Time:   30 To 54

Process returned 0 (0x0)   execution time : 58.510 s
```

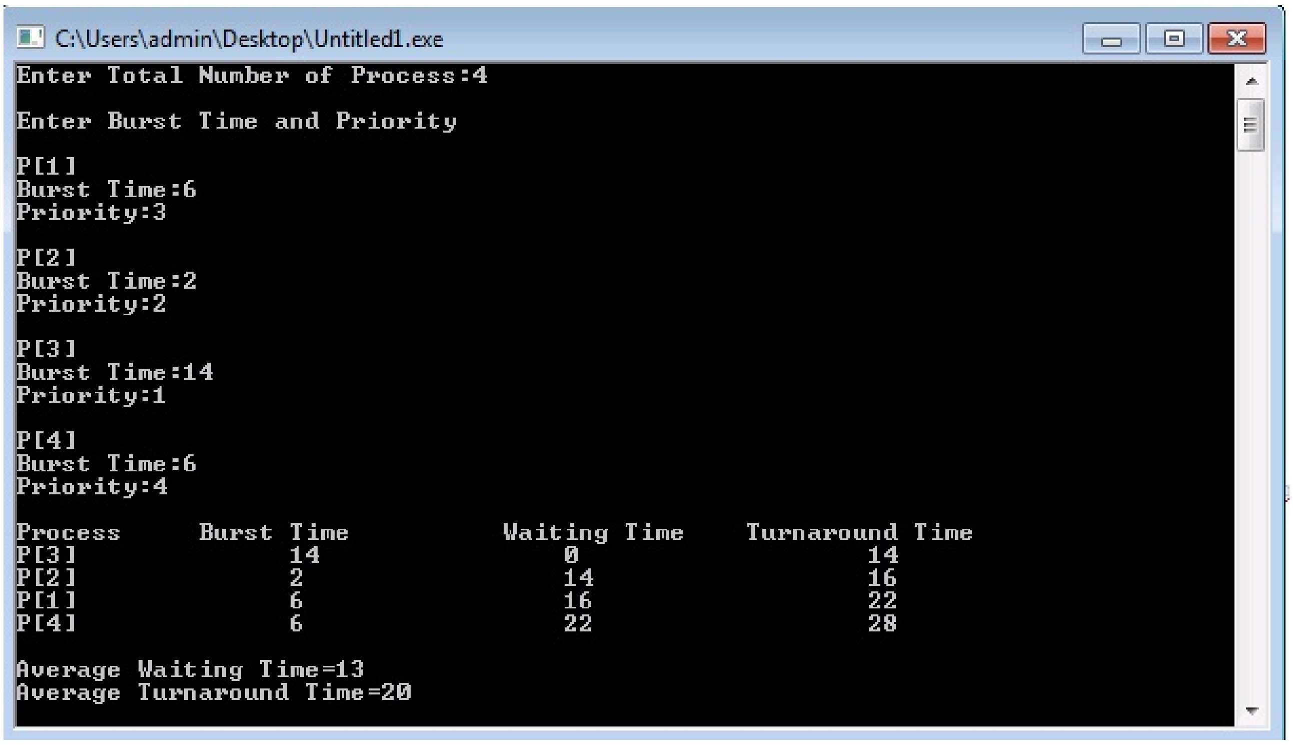
```
gcc   -c /home/mock3/Untitled1.c -o /home/mock3/Untitled1.o
g++  -o /home/mock3/Untitled1 /home/mock3/Untitled1.o
Process terminated with status 0 (0 minute(s), 4 second(s))
0 error(s), 0 warning(s) (0 minute(s), 4 second(s))

Checking for existence: /home/mock3/Untitled1
Executing: xterm -T '/home/mock3/Untitled1' -e /usr/bin/cb_console_runner "/home/mock3/Untitled1" (in /home/mock3)
```

3. Priority and FCFS

FCFS gives the highest priority to the job having been in existence the longest.

PRIORITY:

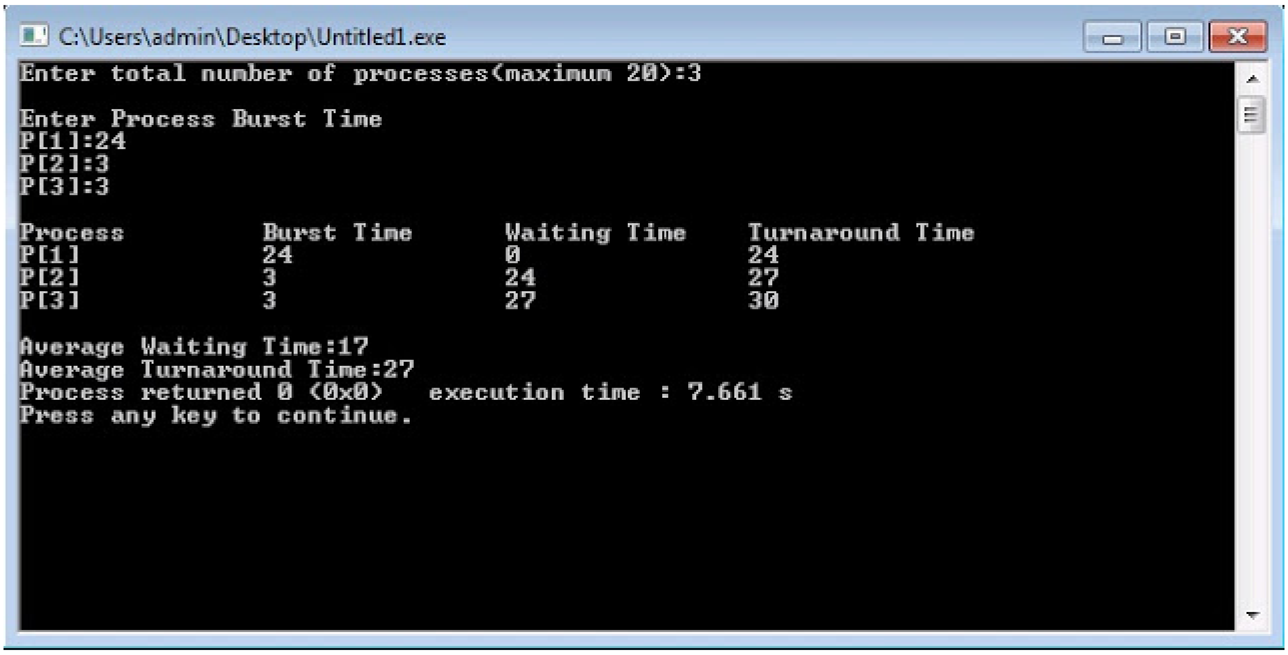
```c
#include<stdio.h>

int main()
{
    int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
    printf("Enter Total Number of Process:");
    scanf("%d",&n);

    printf("\nEnter Burst Time and Priority\n");
    for(i=0;i<n;i++)
    {
        printf("\nP[%d]\n",i+1);
        printf("Burst Time:");
        scanf("%d",&bt[i]);
        printf("Priority:");
        scanf("%d",&pr[i]);
        p[i]=i+1;           //contains process number
    }
    //sorting burst time, priority and process number in ascending order using selection
sort
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(pr[j]<pr[pos])
                pos=j;
        }

        temp=pr[i];
        pr[i]=pr[pos];
        pr[pos]=temp;

        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 is 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=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=total/n;     //average turnaround time
    printf("\n\nAverage Waiting Time=%d",avg_wt);
    printf("\nAverage Turnaround Time=%d\n",avg_tat);

    return 0;
}
```



```
Enter Total Number of Process:4

Enter Burst Time and Priority

P[1]
Burst Time:6
Priority:3

P[2]
Burst Time:2
Priority:2

P[3]
Burst Time:14
Priority:1

P[4]
Burst Time:6
Priority:4

Process     Burst Time          Waiting Time    Turnaround Time
P[3]             14                  0                  14
P[2]             2                   14                 16
P[1]             6                   16                 22
P[4]             6                   22                 28

Average Waiting Time=13
Average Turnaround Time=20
```

FCFC:

```
#include<stdio.h>

int main()
{
    int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
    printf("Enter total number of processes(maximum 20):");
    scanf("%d",&n);

    printf("\nEnter Process Burst Time\n");
    for(i=0;i<n;i++)
    {
        printf("P[%d]:",i+1);
        scanf("%d",&bt[i]);
    }

    wt[0]=0;    //waiting time for first process is 0
    //calculating waiting time
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];
    }

    printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

    //calculating turnaround time
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        avwt+=wt[i];
        avtat+=tat[i];
        printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
    }

    avwt/=i;
    avtat/=i;
    printf("\n\nAverage Waiting Time:%d",avwt);
    printf("\nAverage Turnaround Time:%d",avtat);

    return 0;
}
```



C:\Users\admin\Desktop\Untitled1.exe

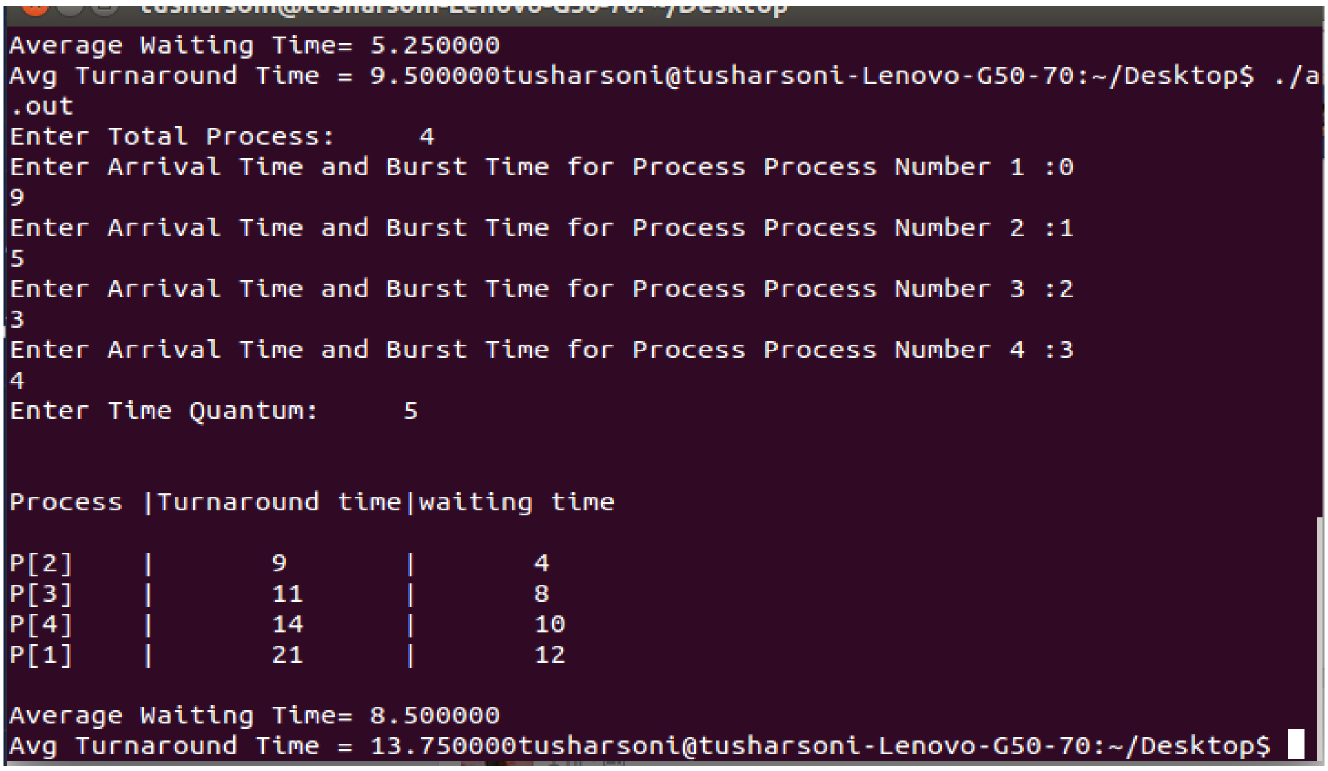
```
Enter total number of processes(maximum 20):3

Enter Process Burst Time
P[1]:24
P[2]:3
P[3]:3

Process         Burst Time      Waiting Time    Turnaround Time
P[1]            24              0               24
P[2]            3               24              27
P[3]            3               27              30

Average Waiting Time:17
Average Turnaround Time:27
Process returned 0 (0x0)   execution time : 7.661 s
Press any key to continue.
```

4. RR and SJF

None.

ROUND-ROBIN:

```
#include<stdio.h>

int main()
{

  int count,j,n,time,remain,flag=0,time_quantum;
  int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
  printf("Enter Total Process:\t ");
  scanf("%d",&n);
  remain=n;
  for(count=0;count<n;count++)
  {
    printf("Enter Arrival Time and Burst Time for Process Process
Number %d :",count+1);
    scanf("%d",&at[count]);
    scanf("%d",&bt[count]);
    rt[count]=bt[count];
  }
  printf("Enter Time Quantum:\t");
  scanf("%d",&time_quantum);
  printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");
  for(time=0,count=0;remain!=0;)
  {
    if(rt[count]<=time_quantum && rt[count]>0)
    {
      time+=rt[count];
      rt[count]=0;
      flag=1;
    }
    else if(rt[count]>0)
    {
      rt[count]-=time_quantum;
      time+=time_quantum;
    }
    if(rt[count]==0 && flag==1)
    {
      remain--;
      printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);
      wait_time+=time-at[count]-bt[count];
      turnaround_time+=time-at[count];
      flag=0;
    }
    if(count==n-1)
      count=0;
    else if(at[count+1]<=time)
      count++;
    else
      count=0;
  }
  printf("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
  printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);

  return 0;
}
```



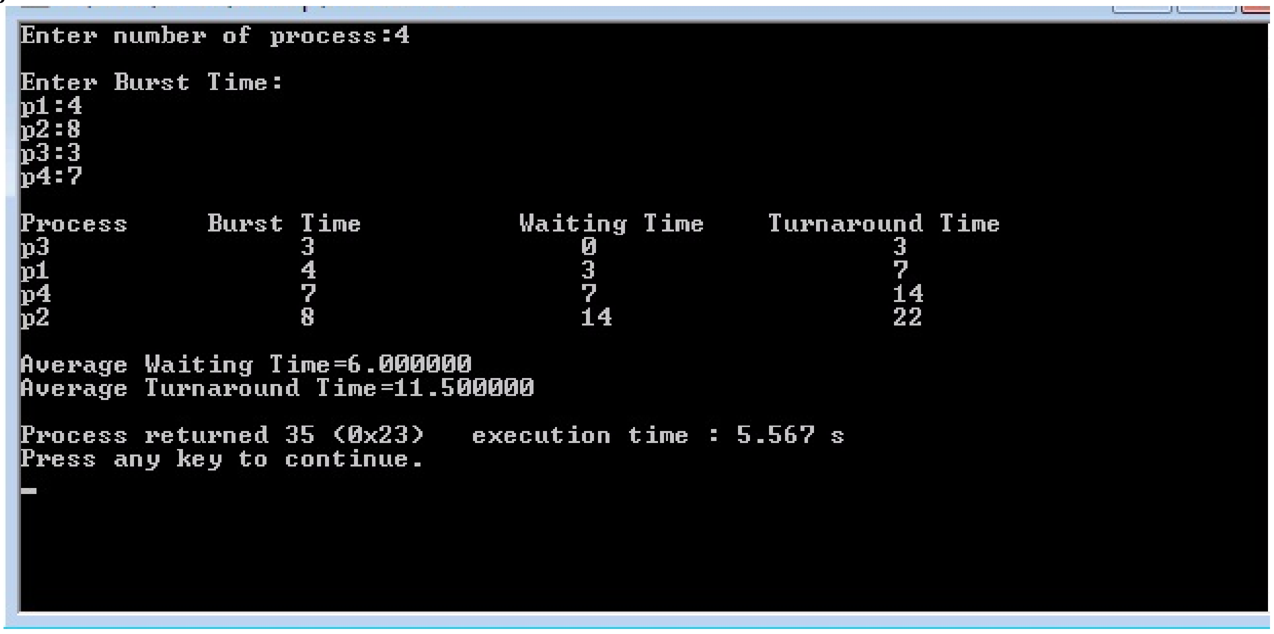
```
Average Waiting Time= 5.250000
Avg Turnaround Time = 9.500000tusharsoni@tusharsoni-Lenovo-G50-70:~/Desktop$ ./a
.out
Enter Total Process:     4
Enter Arrival Time and Burst Time for Process Process Number 1 :0
9
Enter Arrival Time and Burst Time for Process Process Number 2 :1
5
Enter Arrival Time and Burst Time for Process Process Number 3 :2
3
Enter Arrival Time and Burst Time for Process Process Number 4 :3
4
Enter Time Quantum:     5


Process |Turnaround time|waiting time

P[2]    |       9       |       4
P[3]    |       11      |       8
P[4]    |       14      |       10
P[1]    |       21      |       12

Average Waiting Time= 8.500000
Avg Turnaround Time = 13.750000tusharsoni@tusharsoni-Lenovo-G50-70:~/Desktop$
```

SJF:

```
#include<stdio.h>

void main()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf("Enter number of process:");
```

```
scanf("%d",&n);

printf("\nEnter Burst Time:\n");
for(i=0;i<n;i++)
{
   printf("p%d:",i+1);
   scanf("%d",&bt[i]);
   p[i]=i+1;          //contains process number
}

//sorting burst time in ascending order using selection sort
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);
}
```



```
Enter number of process:4

Enter Burst Time:
p1:4
p2:8
p3:3
p4:7

Process      Burst Time          Waiting Time    Turnaround Time
p3                 3                   0                   3
p1                 4                   3                   7
p4                 7                   7                   14
p2                 8                   14                  22

Average Waiting Time=6.000000
Average Turnaround Time=11.500000

Process returned 35 (0x23)   execution time : 5.567 s
Press any key to continue.
```