
# OPERATING SYSTEMS-CSE 316

# PROGRAMMING ASSIGNMENT

**By**
**RITIKA GARG**
**11807538**
**K18PB51**

QUESTION: Consider a scheduling approach which is non pre-emptive similar to shortest job next in nature. The priority of each job is dependent on its estimated run time, and also the amount of time it has spent waiting. Jobs gain higher priority the longer they wait, which prevents indefinite postponement. The jobs that have spent a long time waiting compete against those estimated to have short run times. The priority can be computed as:
Priority = 1+ Waiting time / Estimated run time
Write a program to implement such an algorithm.

# PROGRAM:

```
#include<stdio.h>

int main()
{
  printf("\t\t\t---------------------- Scheduling ----------------------
\n\n\n\n");
  long int n,i=0,j=0;
  printf("Enter Number of Processes : ");
  scanf("%ld",&n );
  double
 priority[n],avg_waiting,avg_turnaround,burstTime[n],arrivalTime[n],waitingTime[n
],turnaroundTime[n], process[n], temp,
 completionTime[n],min,sum=0,sum2=0,wait_final, turnaround_final, wait_avg,
 turnaround_avg;
  for(i=0;i<n;i++)
  {
    printf("\nEnter Burst Time for Process [%d] : ", i+1 );
    scanf("%lf", &burstTime[i]);
    printf("Enter Arrival Time for Process [%d] : ", i+1 );
    scanf("%lf", &arrivalTime[i] );
    process[i]=i+1;
  }

  printf("\n\n\t\t\t -------------- Entered Values are --------------\n\n");
  printf("\t\t\t---------------------------------------\n");
  printf("\t\t\t| Process | Arrival Time | Burst Time |\n");
  printf("\t\t\t---------------------------------------\n");
  for(i=0;i<n;i++)
  {
    printf("\t\t\t|  P[%0.0lf]   |      %0.0lf      |     %0.0lf
|\n",process[i],arrivalTime[i],burstTime[i]);
```

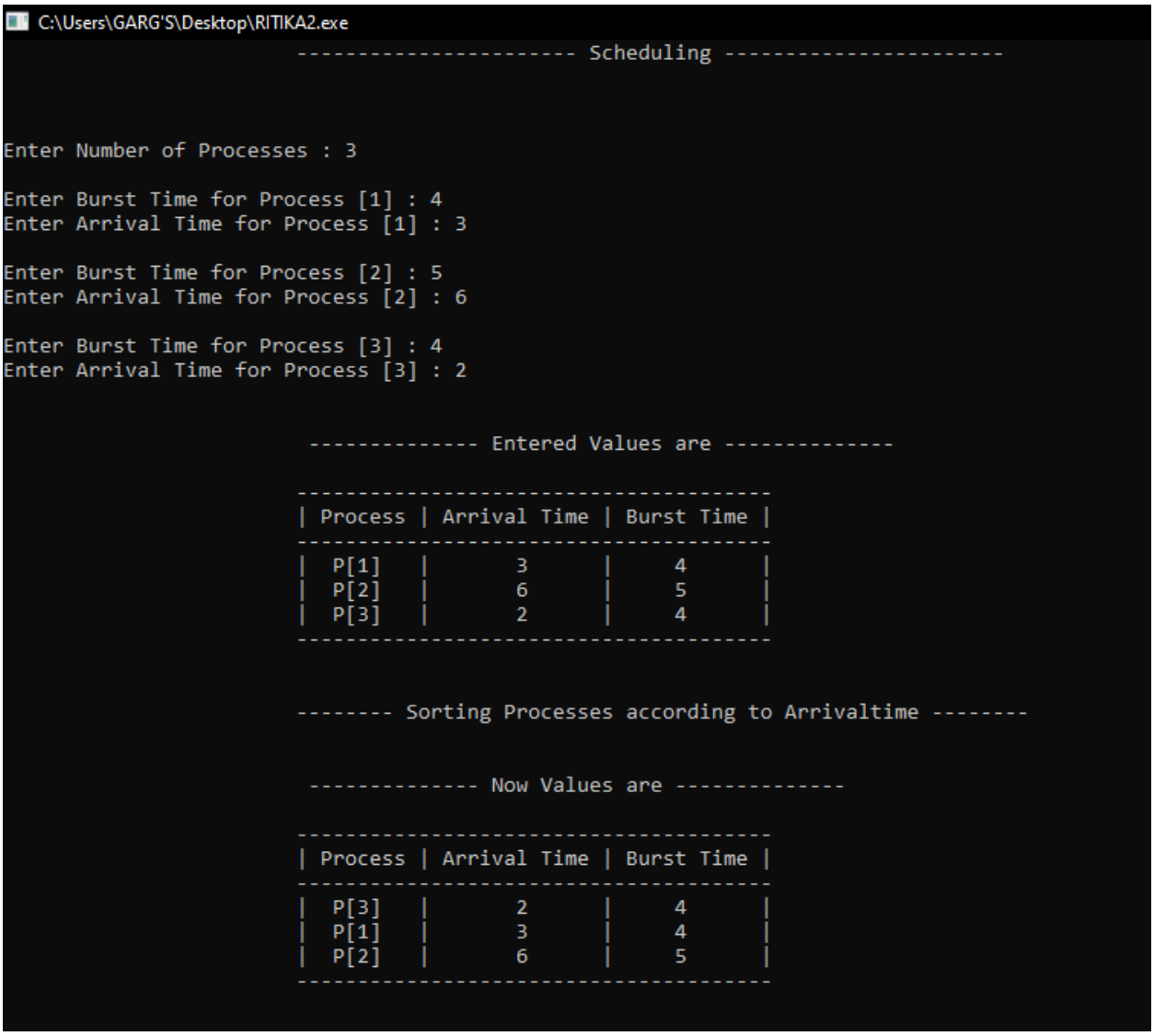
```
  }
    printf("\t\t\t--------------------------------------\n");


  printf("\n\n\t\t\t-------- Sorting Processes according to Arrivaltime --------
\n");

  for(i=0;i<n;i++)
  {
    for(j=0;j<n;j++)
    {
      if(arrivalTime[i]<arrivalTime[j])
      {

        temp = burstTime[j];
        burstTime[j] = burstTime[i];
        burstTime [i] = temp;

        temp = process[j];
        process[j] = process[i];
        process[i] = temp;

        temp = arrivalTime[j];
        arrivalTime[j] = arrivalTime[i];
        arrivalTime[i] = temp;

      }
    }
  }
  printf("\n\n\t\t\t -------------- Now Values are --------------\n\n");
  printf("\t\t\t--------------------------------------\n");
  printf("\t\t\t| Process | Arrival Time | Burst Time |\n");
  printf("\t\t\t--------------------------------------\n");
  for(i=0;i<n;i++)
  {
    printf("\t\t\t|  P[%0.0lf]   |      %0.0lf      |     %0.0lf
|\n",process[i],arrivalTime[i],burstTime[i]);
  }
    printf("\t\t\t--------------------------------------\n");



  long int k = 1;
  double b_time = 0;
```

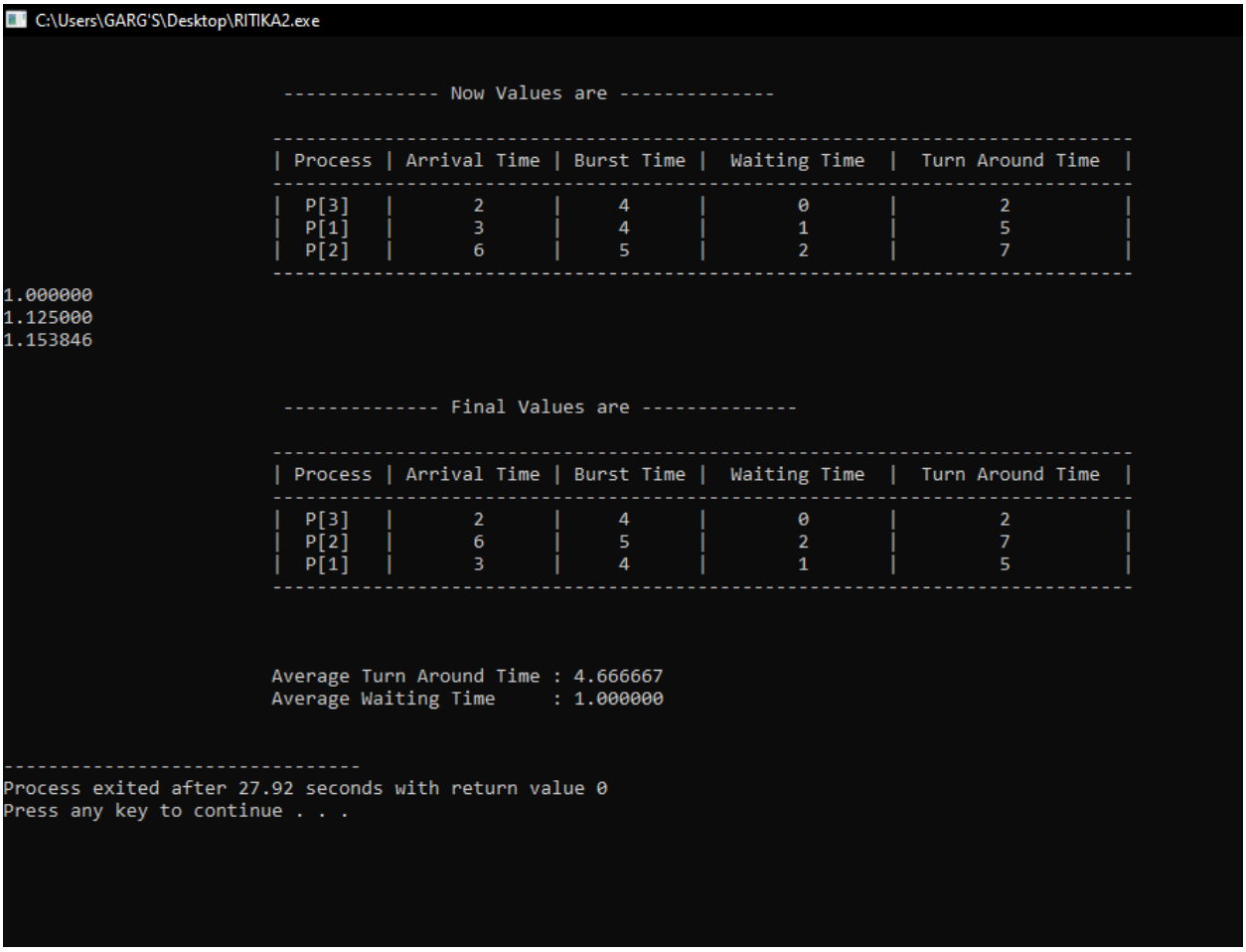
```
 for(j=0;j<n;j++)
  {
    b_time = b_time + burstTime[j];
    min = burstTime[k];

    for(i=k;i<n;i++)
    {
      if((b_time >= arrivalTime[i])&&(burstTime[i]<min))
      {
        temp = burstTime[k];
        burstTime[k] = burstTime[i];
        burstTime[i] = temp;

        temp = arrivalTime[k];
        arrivalTime[k] = arrivalTime[i];
        arrivalTime[i] = temp;

        temp = process[k];
        process[k] = process[i];
        process[i] = temp;
      }
    }
    k++;
  }
  waitingTime[0] = 0;
  for(i=1;i<n;i++)
  {
    sum += burstTime[i-1];
    waitingTime[i] = sum - arrivalTime[i];
    wait_final += waitingTime[i];
  }
  wait_avg = wait_final/n;
  for(i=0;i<n;i++)
  {
    sum2 += burstTime[i];
    turnaroundTime[i] = sum2 - arrivalTime[i];
    turnaround_final += turnaroundTime[i];
  }
  turnaround_avg=turnaround_final/n;
printf("\n\n\t\t\t -------------- Now Values are --------------\n\n");
  printf("\t\t\t-----------------------------------------------------------------
------------\n");
  printf("\t\t\t| Process | Arrival Time | Burst Time |  Waiting Time  |  Turn
 Around Time  |\n");
```

```
  printf("\t\t\t-------------------------------------------------------------------
------------\n");
  for(i=0;i<n;i++)
  {
    printf("\t\t\t|  P[%0.0lf]   |      %0.0lf      |     %0.0lf      |
%0.0lf       |         %0.0lf
|\n",process[i],arrivalTime[i],burstTime[i],waitingTime[i],turnaroundTime[i]);
  }
    printf("\t\t\t-------------------------------------------------------------------
--------------\n");

 completionTime[0] = burstTime[0];

  for(i=1;i<n;i++)
  {
    completionTime[i] = completionTime[i-1] + burstTime[i];
  }

  for(i=0;i<n;i++)
  {
    priority[i] = 1+waitingTime[i]/completionTime[i];
    printf("%lf\n",priority[i]);
  }


  printf("\n\n\t\t\t ------------- Final Values are -------------\n\n");
  printf("\t\t\t-------------------------------------------------------------------
------------\n");
  printf("\t\t\t| Process | Arrival Time | Burst Time |  Waiting Time  |  Turn
 Around Time  |\n");
  printf("\t\t\t-------------------------------------------------------------------
------------\n");
  printf("\t\t\t|  P[%0.0lf]   |      %0.0lf      |     %0.0lf      |
%0.0lf       |         %0.0lf
|\n",process[0],arrivalTime[0],burstTime[0],waitingTime[0],turnaroundTime[0]);
  for(i=n-1;i>0;i--)
  {
    printf("\t\t\t|  P[%0.0lf]   |      %0.0lf      |     %0.0lf      |
%0.0lf       |         %0.0lf
|\n",process[i],arrivalTime[i],burstTime[i],waitingTime[i],turnaroundTime[i]);
  }
    printf("\t\t\t-------------------------------------------------------------------
--------------\n");

  printf("\n\n\n\t\t\tAverage Turn Around Time : %lf",turnaround_avg);
```

```
  printf("\n\t\t\tAverage Waiting Time     : %lf\n\n",wait_avg);


  return 0;
}
```

## SNAPSHOTS:




C:\Users\GARG'S\Desktop\RITIKA2.exe
---------- Now Values are ----------
| Process | Arrival Time | Burst Time | Waiting Time | Turn Around Time |
| P[3] | 2 | 4 | 0 | 2 |
| P[1] | 3 | 4 | 1 | 5 |
| P[2] | 6 | 5 | 2 | 7 |
1.000000
1.125000
1.153846
---------- Final Values are ----------
| Process | Arrival Time | Burst Time | Waiting Time | Turn Around Time |
| P[3] | 2 | 4 | 0 | 2 |
| P[2] | 6 | 5 | 2 | 7 |
| P[1] | 3 | 4 | 1 | 5 |
Average Turn Around Time : 4.666667
Average Waiting Time : 1.000000
Process exited after 27.92 seconds with return value 0
Press any key to continue . . .

**EXPLANATION: Shortest Job First is an algorithm in which the** process **having the smallest execution time is chosen for the next execution. This scheduling method can be** preemptive **or non-**preemptive**. It significantly reduces the average waiting time for other** processes **awaiting execution.**

In non-preemptive scheduling, once the CPU cycle is allocated to process, the process holds it till it reaches a waiting state or terminated.