**CPU PROCESS SCHEDULING SIMULATOR**

A

Mini Project Report

Submitted in partial fulfilment of the

Requirements for the award of the Degree of

**BACHELOR OF ENGINEERING**

IN

**COMPUTER SCIENCE & ENGINEERING**

By

**<P.VAISHNAVI REDDY,M.TEJASWINI>**

**<1602-19-733-113,1602-19-733-114>**

****

**Department of Computer Science & Engineering**

**Vasavi College of Engineering (Autonomous)**

**(Affiliated to Osmania University)**

**Ibrahimbagh, Hyderabad-31**

**2019**

**Vasavi College of Engineering (Autonomous)**

**(Affiliated to Osmania University)**

**Hyderabad-500 031**

**Department of Computer Science & Engineering**

****

**DECLARATION BY THE CANDIDATE**

I, **<M TEJASWINI,P VAISHNAVI REDDY>,** bearing hall ticket number, **<1602-19-733-113,1602-19-733-114>**, hereby declare that the project report entitled **<SIMULATION OF CPU SCHEDULING ALGORITHMS>** Department of Computer Science & Engineering, VCE, Hyderabad, is submitted in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering** in **Computer Science & Engineering**.

This is a record of bonafide work carried out by me and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

**M TEJASWINI**

**(1602-19-733-113)**

**P VAISHNAVI REDDY**

**(1602-19-733-114)**

**Vasavi College of Engineering (Autonomous)**

**(Affiliated to Osmania University)**

**Hyderabad-500 031**

**Department of Computer Science & Engineering**

****

**BONAFIDE CERTIFICATE**

Thisis to certify that the project entitled **< SIMULATION OF CPU SCHEDULING ALGORITHMS >** being submitted by **<P VAISHNAVI REDDY,M TEJASWINI >,** bearing <**1602-19-733-114,1602-19-733-113>,** in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science & Engineering is a record of bonafide work carried out by him/her under my guidance.

**Dr. K. Srinivas**

**Professor**

**Dept. of CSE,**

**1. ACKNOWLEDGEMENT**

We would like to express our heartfelt gratitude to Dr. K. Srinivas, our project guide, for her valuable guidance and constant support, along with her capable instruction and persistent encouragement.

We are grateful to our Head of Department, Dr. T. Adilakshmi, for her steady support and the provision of every resource required for the completion of this project.

We would like to take this opportunity to thank our Principal, Dr. S. V. RAMANA, as well as the management of the institute, for having designed an excellent learning atmosphere.

**ABSTARCT**

Operating System (OS) is the most vital software program that runs at the pc. It now not most effective manages hardware, reminiscence, and different resources but it’s also answerable for optimizing the computer performance. It interacts with I/O gadgets and user’s programs and packages. You can not assume to apply a pc without an operating gadget.

One of the important thing principles in OS design is CPU scheduling (or processor scheduling). CPU scheduling dramatically affects overall performance of the gadget as it determines which approaches will wait and which will progress. Because OS also acts as a useful resource supervisor, it need to make the satisfactory use of CPU. This project – CPU Scheduling Simulator (CSS) – focusses on CPU scheduling.

**Program will simulate following scheduling algorithms:**

1)First Come First Serve Scheduling  
2) Shortest Job First Scheduling (SJF)  
three) Shortest Remaining Time First Scheduling (SRTF) – preemptive version of SJF  
4) Priority Scheduling(pre-emptive and non pre-emptive)  
5) Round Robin Scheduling

**Program will take input from the user and calculates these:**

1) Waiting time of each technique  
2) Turnaround time of each manner  
three) Average ready time of given workload  
four) Average turnaround time of given workload  
5) Gantt chart indicating schedule of each process  
Benefits:

**TABLE OF CONTENTS**

List of figures - 7

Introduction - 8

Design and analysis - 9

Source Code - 10 – 14

Files - 15

Results - 16 - 23

Conclusion - 24

References - 25

**List of Figures**

Figure 1 - giving input(burst time,arrival time,priority)

Figure 2 - cpu scheduling using FCFS

Figure 3 - using SJF(pre-emptive)

Figure 4 - using SJF(non pre-emptive)

Figure 5 - using Priority Scheduling(Pre-emptive)

Figure 6 - using Priority Scheduling(Non Pre-emptive)

Figure 7 - using Round Robin Scheduling

**Introduction**

* 1. **SCHEDULING**

An operating system is a program that manages the hardware and software resources of a computer. It is the first thing that is loaded into memory when we turn on the computer. Without the operating system, each programmer would have to create a way in which a program will display text and graphics on the monitor. The programmer would have to create a way to send data to a printer, tell it how to read a disk file, and how to deal with other programs. In the beginning, programmers needed a way to handle complex input/output operations. The evolution of computer programs and their complexities required new necessities. Because machines began to become more powerful, the time a program needed to run decreased. However, the time needed for handing off the equipment between different programs became evident and this led to programs like DOS. As we can see the acronym DOS stands for Disk Operating System. This confirms that operating systems were originally made to handle these complex input/output operations like communicating among a variety of disk drives. Earlier computers were not as powerful as they are today. In the early computer systems you would only be able to run one program at a time. For instance, you could not be writing a paper and browsing the internet all at the same time. However, today’s operating systems are very capable of handling not only two but multiple applications at the same time. In fact, if a computer is not able to do this it is considered useless by most computer users. In order for a computer to be able to handle multiple applications simultaneously, there must be an effective way of using the CPU. Several processes may be running at the same time, so there has to be some kind of order to allow each process to get its share of CPU time.

An operating system must allocate computer resources among the potentially competing requirements of multiple processes. In the case of the processor, the resource to be allocated is execution time on the processor and the means of allocation is scheduling. The scheduling function must be designed to satisfy a number of objectives, including fairness, lack of starvation of any particular process, efficient use of processor time, and low overhead. In addition, the scheduling 12

function may need to take into account different levels of priority or real-time deadlines for the start or completion of certain processes. Over the years, scheduling has been the focus of intensive research, and many different algorithms have been implemented. Today, the emphasis in scheduling research is on exploiting multiprocessor systems, particularly for multithreaded applications, and real-time scheduling. In a multiprogramming system, multiple processes exist concurrently in main memory. Each process alternates between using a processor and waiting for some event to occur, such as the completion of an I/O operation. The processor or processors are kept busy by executing one process while the others wait, hence the key to multiprogramming is **scheduling**.

* 1. **GOAL & OBJECTIVE OF THE PROJECT**

The implementation of the CPU scheduling algorithms is an automation that provides efficient and errorless computation of waiting times, turnaround times, finishing times and normalized turnaround times of First Come First Served (FCFS), Shortest Process Next (SPN), Shortest Remaining Time (SRT), Round-Robin (RR), and Priority Scheduling algorithms. The system provides a clean and convenient way to test the given data and do the analysis of the CPU scheduling algorithms mentioned above. For the purpose of analysis and testing, the user first specifies each process along with its information such as arrival times and service times and then FCFS, SPN, SRT, RR, or Priority can be computed producing output in appropriate format for readability.

* 1. **SCOPE OF THE PROJECT**

In order to make the computer more productive in multiprogramming, the operating system needs to switch the CPU among processes. It must provides the basic algorithm to determine which process is allowed to get the CPU at the current time, and whether that process is allowed to finish its execution comparing to other processes in the system. Therefore, CPU scheduling algorithms such as First Come First Served (FCFS), Shortest Process Next (SPN), Shortest Remaining Time (SRT), Round-Robin (RR), and Priority Scheduling are among possible solutions for multiprogramming operating system.

A multiprogramming operating system allows more than one process to be loaded into the executable memory at a time and for the loaded process to share the CPU using time multiplexing. Part of the reason for using multiprogramming is that the operating system itself is implemented as one or more processes, so there must be a way for the operating system and application processes to share the CPU. Another main reason is the need for processes to perform I/O operations in the normal course of computation. Since I/O operations ordinarily require orders of magnitude more time to complete than do CPU instructions, multiprogramming systems allocate the CPU to another process whenever a process invokes an I/O operation.

**DESIGN AND ANALYSIS**



*Comparison of Various Scheduling Algorithms*



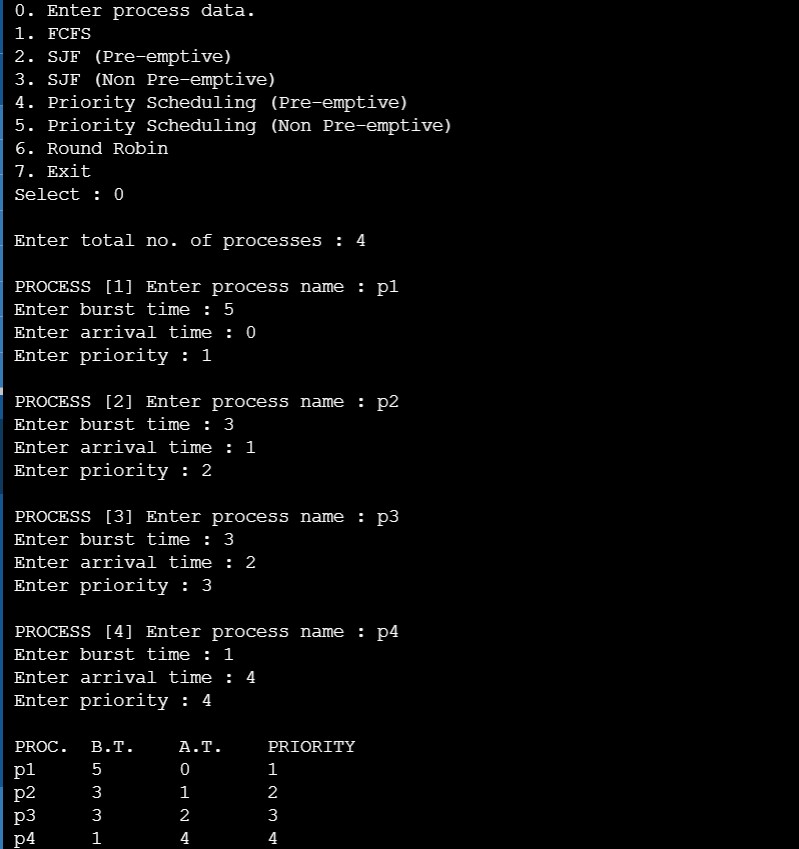
Clearly, the performance of various scheduling policies is a critical factor in the choice of a scheduling policy. However, it is impossible to make definitive comparisons because relative performance will depend on a variety of factors, including the probability distribution of service times of the various processes, the efficiency of the scheduling and context switching mechanisms, and the nature of the I/O demand and the performance of the I/O subsystem.

**IMPLEMENTATION**

|  |
| --- |
| #include<stdio.h> |
|  | #include<stdlib.h> |
|  |  |
|  | typedef struct process{ |
|  | char name[5]; |
|  | int bt; |
|  | int at; |
|  | int prt; |
|  | int wt,ta; |
|  | int flag; |
|  | }processes; |
|  |  |
|  |  |
|  | void b\_sort(processes temp[],int n) |
|  | { |
|  | processes t; |
|  | int i,j; |
|  | for(i=1;i<n;i++) |
|  | for(j=0;j<n-i;j++){ |
|  | if(temp[j].at > temp[j+1].at){ |
|  | t = temp[j]; |
|  | temp[j] = temp[j+1]; |
|  | temp[j+1] = t; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | int accept(processes P[]){ |
|  | int i,n; |
|  | printf("\n Enter total no. of processes : "); |
|  | scanf("%d",&n); |
|  | for(i=0;i<n;i++){ |
|  | printf("\n PROCESS [%d]",i+1); |
|  | printf(" Enter process name : "); |
|  | scanf("%s",&P[i].name); |
|  | printf(" Enter burst time : "); |
|  | scanf("%d",&P[i].bt); |
|  | printf(" Enter arrival time : "); |
|  | scanf("%d",&P[i].at); |
|  | printf(" Enter priority : "); |
|  | scanf("%d",&P[i].prt); |
|  | } |
|  | printf("\n PROC.\tB.T.\tA.T.\tPRIORITY"); |
|  | for(i=0;i<n;i++) |
|  | printf("\n %s\t%d\t%d\t%d",P[i].name,P[i].bt,P[i].at,P[i].prt); |
|  | return n; |
|  | } |
|  |  |
|  | // FCFS Algorithm |
|  | void FCFS(processes P[],int n){ |
|  | processes temp[10]; |
|  | int sumw=0,sumt=0; |
|  | int x = 0; |
|  | float avgwt=0.0,avgta=0.0; |
|  | int i,j; |
|  | for(i=0;i<n;i++) |
|  | temp[i]=P[i]; |
|  |  |
|  | b\_sort(temp,n); |
|  |  |
|  | printf("\n\n PROC.\tB.T.\tA.T."); |
|  | for(i=0;i<n;i++) |
|  | printf("\n %s\t%d\t%d",temp[i].name,temp[i].bt,temp[i].at); |
|  |  |
|  | sumw = temp[0].wt = 0; |
|  | sumt = temp[0].ta = temp[0].bt - temp[0].at; |
|  |  |
|  | for(i=1;i<n;i++){ |
|  | temp[i].wt = (temp[i-1].bt + temp[i-1].at + temp[i-1].wt) - temp[i].at;; |
|  | temp[i].ta = (temp[i].wt + temp[i].bt); |
|  | sumw+=temp[i].wt; |
|  | sumt+=temp[i].ta; |
|  | } |
|  | avgwt = (float)sumw/n; |
|  | avgta = (float)sumt/n; |
|  | printf("\n\n PROC.\tB.T.\tA.T.\tW.T\tT.A.T"); |
|  | for(i=0;i<n;i++) |
|  | printf("\n %s\t%d\t%d\t%d\t%d",temp[i].name,temp[i].bt,temp[i].at,temp[i].wt,temp[i].ta); |
|  |  |
|  | printf("\n\n GANTT CHART\n "); |
|  | for(i=0;i<n;i++) |
|  | printf(" %s ",temp[i].name); |
|  | printf("\n "); |
|  |  |
|  | printf("0\t"); |
|  | for(i=1;i<=n;i++){ |
|  | x+=temp[i-1].bt; |
|  | printf("%d ",x); |
|  | } |
|  | printf("\n\n Average waiting time = %0.2f\n Average turn-around = %0.2f.",avgwt,avgta); |
|  | } |
|  |  |
|  |  |
|  | //SJF Non Pre-emptive |
|  | void SJF\_NP(processes P[],int n){ |
|  | processes temp[10]; |
|  | processes t; |
|  | int sumw=0,sumt=0; |
|  | int x = 0; |
|  | float avgwt=0.0,avgta=0.0; |
|  | int i,j; |
|  |  |
|  | for(i=0;i<n;i++) |
|  | temp[i]=P[i]; |
|  |  |
|  | b\_sort(temp,n); |
|  |  |
|  | for(i=2;i<n;i++) |
|  | for(j=1;j<n-i+1;j++){ |
|  | if(temp[j].bt > temp[j+1].bt){ |
|  | t = temp[j]; |
|  | temp[j] = temp[j+1]; |
|  | temp[j+1] = t; |
|  | } |
|  | } |
|  |  |
|  | printf("\n\n PROC.\tB.T.\tA.T."); |
|  | for(i=0;i<n;i++) |
|  | printf("\n %s\t%d\t%d",temp[i].name,temp[i].bt,temp[i].at); |
|  |  |
|  | sumw = temp[0].wt = 0; |
|  | sumt = temp[0].ta = temp[0].bt - temp[0].at; |
|  |  |
|  | for(i=1;i<n;i++){ |
|  | temp[i].wt = (temp[i-1].bt + temp[i-1].at + temp[i-1].wt) - temp[i].at;; |
|  | temp[i].ta = (temp[i].wt + temp[i].bt); |
|  | sumw+=temp[i].wt; |
|  | sumt+=temp[i].ta; |
|  | } |
|  | avgwt = (float)sumw/n; |
|  | avgta = (float)sumt/n; |
|  | printf("\n\n PROC.\tB.T.\tA.T.\tW.T\tT.A.T"); |
|  | for(i=0;i<n;i++) |
|  | printf("\n %s\t%d\t%d\t%d\t%d",temp[i].name,temp[i].bt,temp[i].at,temp[i].wt,temp[i].ta); |
|  |  |
|  | printf("\n\n GANTT CHART\n "); |
|  | for(i=0;i<n;i++) |
|  | printf(" %s ",temp[i].name); |
|  | printf("\n "); |
|  |  |
|  | printf("0\t"); |
|  | for(i=1;i<=n;i++){ |
|  | x+=temp[i-1].bt; |
|  | printf("%d ",x); |
|  | } |
|  | printf("\n\n Average waiting time = %0.2f\n Average turn-around = %0.2f.",avgwt,avgta); |
|  | } |
|  |  |
|  | //Priority Non Pre-emptive |
|  | void PRT\_NP(processes P[],int n) |
|  | { |
|  | processes temp[10]; |
|  | processes t; |
|  | int sumw=0,sumt=0; |
|  | float avgwt=0.0,avgta=0.0; |
|  | int i,j; |
|  | int x = 0; |
|  |  |
|  | for(i=0;i<n;i++) |
|  | temp[i]=P[i]; |
|  |  |
|  | b\_sort(temp,n); |
|  |  |
|  | for(i=2;i<n;i++) |
|  | for(j=1;j<n-i+1;j++){ |
|  | if(temp[j].prt > temp[j+1].prt){ |
|  | t = temp[j]; |
|  | temp[j] = temp[j+1]; |
|  | temp[j+1] = t; |
|  | } |
|  | } |
|  |  |
|  | printf("\n\n PROC.\tB.T.\tA.T."); |
|  | for(i=0;i<n;i++) |
|  | printf("\n %s\t%d\t%d",temp[i].name,temp[i].bt,temp[i].at); |
|  |  |
|  | sumw = temp[0].wt = 0; |
|  | sumt = temp[0].ta = temp[0].bt - temp[0].at; |
|  |  |
|  | for(i=1;i<n;i++){ |
|  | temp[i].wt = (temp[i-1].bt + temp[i-1].at + temp[i-1].wt) - temp[i].at;; |
|  | temp[i].ta = (temp[i].wt + temp[i].bt); |
|  | sumw+=temp[i].wt; |
|  | sumt+=temp[i].ta; |
|  | } |
|  | avgwt = (float)sumw/n; |
|  | avgta = (float)sumt/n; |
|  | printf("\n\n PROC.\tB.T.\tA.T.\tW.T\tT.A.T"); |
|  | for(i=0;i<n;i++) |
|  | printf("\n %s\t%d\t%d\t%d\t%d",temp[i].name,temp[i].bt,temp[i].at,temp[i].wt,temp[i].ta); |
|  |  |
|  | printf("\n\n GANTT CHART\n "); |
|  | for(i=0;i<n;i++) |
|  | printf(" %s ",temp[i].name); |
|  | printf("\n "); |
|  |  |
|  | printf("0\t"); |
|  | for(i=1;i<=n;i++){ |
|  | x+=temp[i-1].bt; |
|  | printf("%d ",x); |
|  | } |
|  | printf("\n\n Average waiting time = %0.2f\n Average turn-around = %0.2f.",avgwt,avgta); |
|  | } |
|  |  |
|  | //Round Robin Scheduling |
|  | void RR(processes P[],int n) |
|  | { |
|  | int pflag=0,t,tcurr=0,k,i,Q=0; |
|  | int sumw=0,sumt=0; |
|  | float avgwt=0.0,avgta=0.0; |
|  | processes temp1[10],temp2[10]; |
|  |  |
|  | for(i=0;i<n;i++) |
|  | temp1[i]=P[i]; |
|  |  |
|  | b\_sort(temp1,n); |
|  |  |
|  | for(i=0;i<n;i++) |
|  | temp2[i]=temp1[i]; |
|  |  |
|  | printf("\n Enter quantum time : "); |
|  | scanf("%d",&Q); |
|  |  |
|  | for(k=0;;k++){ |
|  | if(k>n-1) |
|  | k=0; |
|  | if(temp1[k].bt>0) |
|  | printf(" %d %s",tcurr,temp1[k].name); |
|  | t=0; |
|  | while(t<Q && temp1[k].bt > 0){ |
|  | t++; |
|  | tcurr++; |
|  | temp1[k].bt--; |
|  | } |
|  | if(temp1[k].bt <= 0 && temp1[k].flag != 1){ |
|  | temp1[k].wt = tcurr - temp2[k].bt - temp1[k].at; |
|  | temp1[k].ta = tcurr - temp1[k].at; |
|  | pflag++; |
|  | temp1[k].flag = 1; |
|  | sumw+=temp1[k].wt; |
|  | sumt+=temp1[k].ta; |
|  | } |
|  | if(pflag == n) |
|  | break; |
|  | } |
|  | printf(" %d",tcurr); |
|  | avgwt = (float)sumw/n; |
|  | avgta = (float)sumt/n; |
|  | printf("\n\n Average waiting time = %0.2f\n Average turn-around = %0.2f.",avgwt,avgta); |
|  | } |
|  |  |
|  | //Shortest Job First - Pre-emptive |
|  | void SJF\_P(processes P[],int n){ |
|  | int i,t\_total=0,tcurr,b[10],min\_at,j,x,min\_bt; |
|  | int sumw=0,sumt=0; |
|  | float avgwt=0.0,avgta=0.0; |
|  | processes temp[10],t; |
|  |  |
|  | for(i=0;i<n;i++){ |
|  | temp[i]=P[i]; |
|  | t\_total+=P[i].bt; |
|  | } |
|  |  |
|  | b\_sort(temp,n); |
|  |  |
|  | for(i=0;i<n;i++) |
|  | b[i] = temp[i].bt; |
|  |  |
|  | i=j=0; |
|  | printf("\n GANTT CHART\n\n %d %s",i,temp[i].name); |
|  | for(tcurr=0;tcurr<t\_total;tcurr++){ |
|  |  |
|  | if(b[i] > 0 && temp[i].at <= tcurr) |
|  | b[i]--; |
|  |  |
|  | if(i!=j) |
|  | printf(" %d %s",tcurr,temp[i].name); |
|  |  |
|  | if(b[i]<=0 && temp[i].flag != 1){ |
|  |  |
|  | temp[i].flag = 1; |
|  | temp[i].wt = (tcurr+1) - temp[i].bt - temp[i].at; |
|  | temp[i].ta = (tcurr+1) - temp[i].at; |
|  | sumw+=temp[i].wt; |
|  | sumt+=temp[i].ta; |
|  | } |
|  | j=i; min\_bt = 999; |
|  | for(x=0;x<n;x++){ |
|  |  |
|  | if(temp[x].at <= (tcurr+1) && temp[x].flag != 1){ |
|  |  |
|  | if(min\_bt != b[x] && min\_bt > b[x]){ |
|  | min\_bt = b[x]; |
|  | i=x; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | } |
|  | printf(" %d",tcurr); |
|  | avgwt = (float)sumw/n; avgta = (float)sumt/n; |
|  | printf("\n\n Average waiting time = %0.2f\n Average turn-around = %0.2f.",avgwt,avgta); |
|  | } |
|  |  |
|  |  |
|  | void PRT\_P(processes P[],int n){ |
|  | int i,t\_total=0,tcurr,b[10],j,x,min\_pr; |
|  | int sumw=0,sumt=0; |
|  | float avgwt=0.0,avgta=0.0; |
|  | processes temp[10],t; |
|  |  |
|  | for(i=0;i<n;i++){ |
|  | temp[i]=P[i]; |
|  | t\_total+=P[i].bt; |
|  | } |
|  |  |
|  | b\_sort(temp,n); |
|  |  |
|  | for(i=0;i<n;i++) |
|  | b[i] = temp[i].bt; |
|  |  |
|  | i=j=0; |
|  | printf("\n GANTT CHART\n\n %d %s",i,temp[i].name); |
|  | for(tcurr=0;tcurr<t\_total;tcurr++) |
|  | { |
|  |  |
|  | if(b[i] > 0 && temp[i].at <= tcurr) |
|  | b[i]--; |
|  |  |
|  | if(i!=j) |
|  | printf(" %d %s",tcurr,temp[i].name); |
|  |  |
|  | if(b[i]<=0 && temp[i].flag != 1) |
|  | { |
|  | temp[i].flag = 1; |
|  | temp[i].wt = (tcurr+1) - temp[i].bt - temp[i].at; |
|  | temp[i].ta = (tcurr+1) - temp[i].at; |
|  | sumw+=temp[i].wt; |
|  | sumt+=temp[i].ta; |
|  | } |
|  | j=i; |
|  | min\_pr = 999; |
|  | for(x=0;x<n;x++){ |
|  |  |
|  | if(temp[x].at <= (tcurr+1) && temp[x].flag != 1){ |
|  |  |
|  | if(min\_pr != temp[x].prt && min\_pr > temp[x].prt){ |
|  | min\_pr = temp[x].prt; |
|  | i=x; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | } |
|  | printf(" %d",tcurr); |
|  | avgwt = (float)sumw/n; |
|  | avgta = (float)sumt/n; |
|  | printf("\n\n Average waiting time = %0.2f\n Average turn-around = %0.2f.",avgwt,avgta); |
|  | } |
|  |  |
|  |  |
|  | int main(){ |
|  |  |
|  | processes P[10]; |
|  | int ch,n; |
|  | do{ |
|  | printf("\n\n SIMULATION OF CPU SCHEDULING ALGORITHMS\n"); |
|  | printf("\n Options:"); |
|  | printf("\n 0. Enter process data."); |
|  | printf("\n 1. FCFS"); |
|  | printf("\n 2. SJF (Pre-emptive)"); |
|  | printf("\n 3. SJF (Non Pre-emptive)"); |
|  | printf("\n 4. Priority Scheduling (Pre-emptive)"); |
|  | printf("\n 5. Priority Scheduling (Non Pre-emptive)"); |
|  | printf("\n 6. Round Robin"); |
|  | printf("\n 7. Exit\n Select : "); |
|  | scanf("%d",&ch); |
|  | switch(ch){ |
|  | case 0: |
|  | n=accept(P); |
|  | break; |
|  | case 1: |
|  | FCFS(P,n); |
|  | break; |
|  | case 2: |
|  | SJF\_P(P,n); |
|  | break; |
|  | case 3: |
|  | SJF\_NP(P,n); |
|  | break; |
|  | case 4: |
|  | PRT\_P(P,n); |
|  | break; |
|  | case 5: |
|  | PRT\_NP(P,n); |
|  | break; |
|  | case 6: |
|  | RR(P,n); |
|  | break; |
|  | case 7:exit(0); |
|  | } |
|  | }while(ch != 7); |
|  | getch(); |
|  | return 0; |
|  | } |

**OUTPUTS/RESULTS**

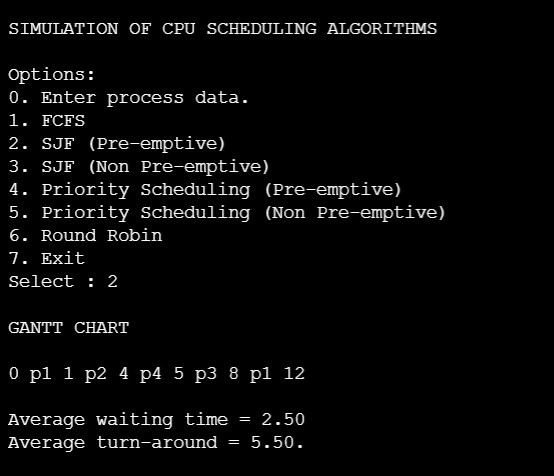
O/P:1: giving input(burst time,arrival time,priority)



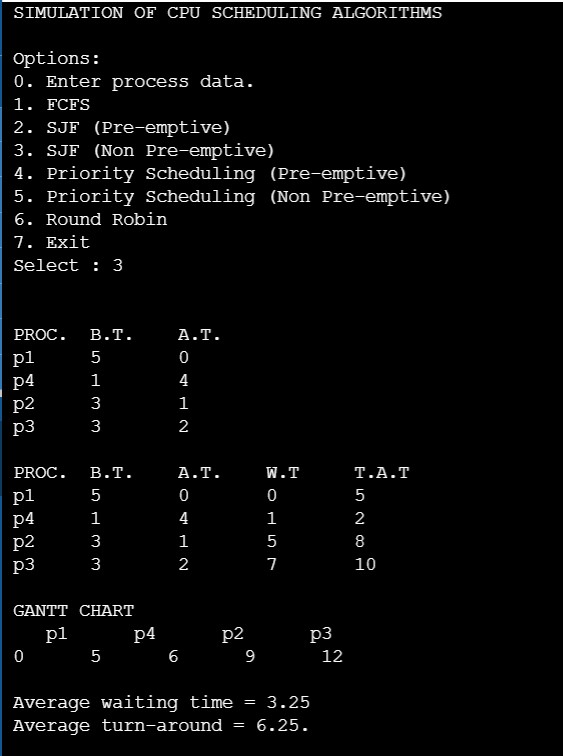
O/P:2: cpu scheduling using FCFS



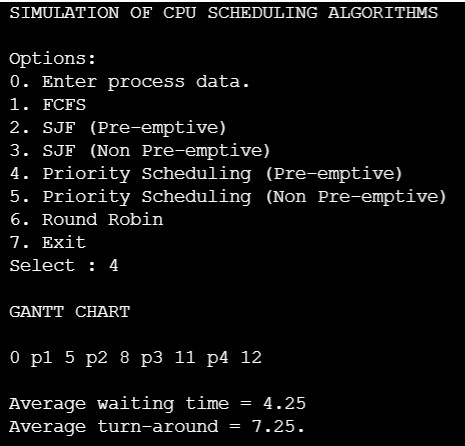
O/P:3: using SJF(pre-emptive)



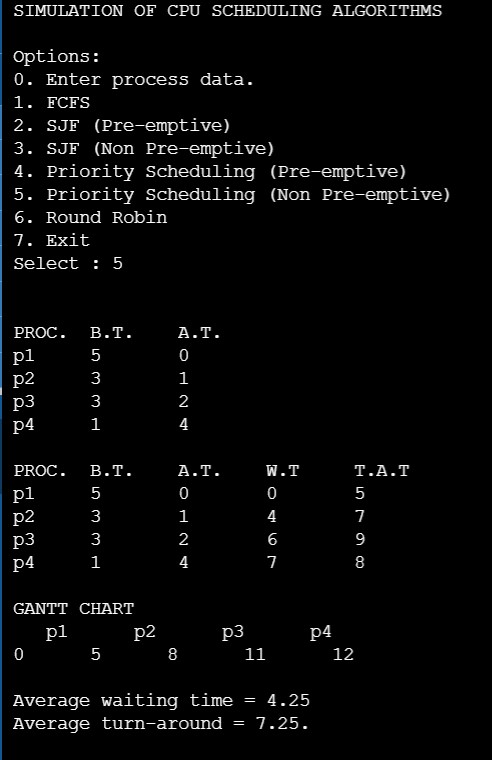
O/P:4: using SJF(non pre-emptive)



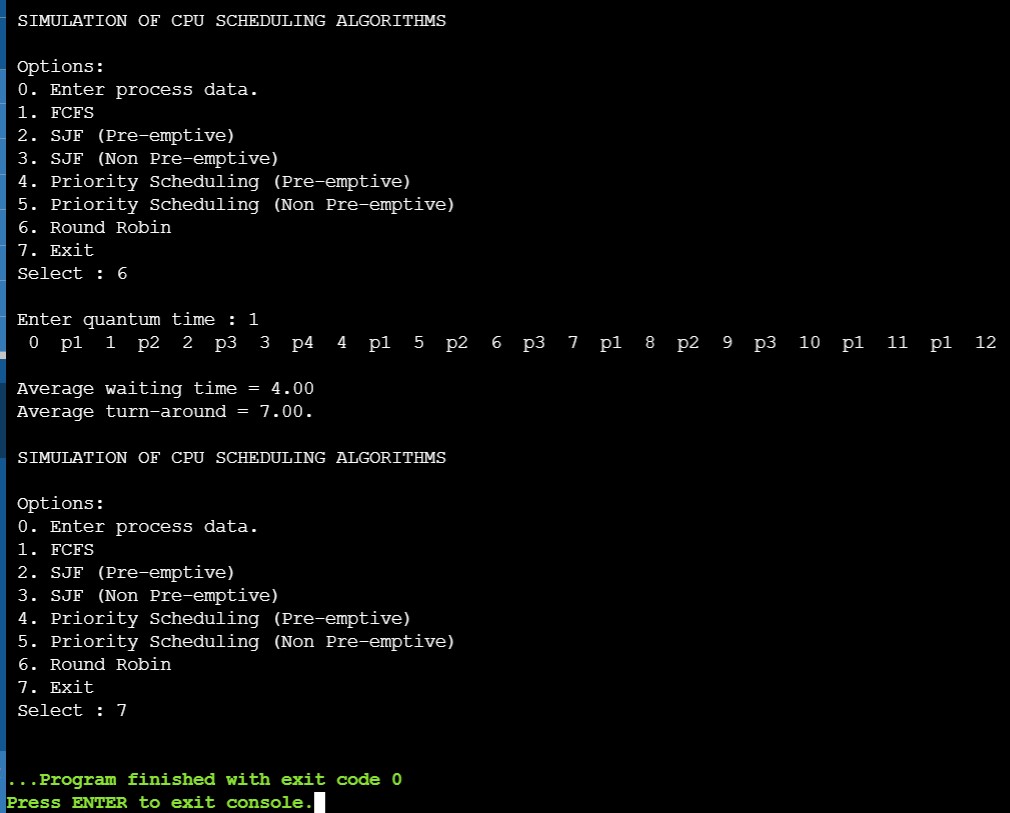
O/P:5: using Priority Scheduling(Pre-emptive)



O/P:6: using Priority Scheduling(Non Pre-emptive)



O/P:7: using Round Robin Scheduling



**CONCLUSION**

In conclusion we would like to tell that this mini project can be further extended to design application which acts like a multiprogramming OS. ***CPU Scheduling*** *is a key concept in computer multitasking, multiprocessing operating system and real-time operating system designs. It refers to the way processes are assigned to run on the available CPUs, since there are typically many more processes running than there are available CPUs. CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU. By switching the CPU among processes, the operating system can make the computer more productive. A multiprogramming operating system allows more than one process to be loaded into the executable memory at a time and for the loaded process to share the CPU using time-multiplexing.*

**References**

1.Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, Operating System Concepts, 2008

2. William Stallings, Operating System, 2007