Student Name :- S.P.Chandan.Kandikattu

Student ID :- 11802942(A15)

Student Email :- kspchandannani@gmail.com

GitHub Link :- https://github.com/AdityaHada1510/CSE-316.git

Question 15 :-

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 pre-emptive 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 are arriving at same time.

Answer :-

We can solve this problem by using Shortest Job First CPU scheduling algorithm which is of two kinds 1. Non Pre-emptive

2. Pre-emptive

In this case we solve this problem using Pre-emptive case.

1. **Completion Time**: Time at which process completes its execution.
2. **Turn Around Time**: Time Difference between completion time and arrival time. Turn Around Time = Completion Time – Arrival Time
3. **Waiting Time (W.T):** Time Difference between turnaround time and burst time.  
   Waiting Time = Turn Around Time – Burst Time

***CODE :-***

#include <iostream>

using namespace std;

int main(){

int i,j,k,p,s=0, got=0, idle=0, temp\_burst, temp\_row, pre\_process\_row, done=0;

float sum=0;

cout<<"Please enter the number of process : ";

cin>>p;

int x[p][5];

int y[p][5];

cout<<"\nProcess\tarrival\tburst\n-------\t-------\t-----\n";

for(i=0;i<p;i++){

for(j=0;j<3;j++){

cin>>x[i][j];

}

x[i][3]=x[i][2];

}

cout<<"\n\nTime-Line is as follows (Verticle View)....\n\n";

i=x[0][1];

while(done!=p){

got=0;

k=0;

while(k<p){

if(x[k][1]<=i){

if(x[k][2]!=0){

got=1;

temp\_burst=x[k][2];

temp\_row=k;

idle=0;

break;

}

else

k++;

}

else{

if(idle==0)

printf("%5d-----------\n |Idle |\n",i);

idle=1;

break;

}

}

if(got!=0){

k=0;

while(x[k][1]<=i && k<p){

if(x[k][2]!=0){

if(temp\_burst>x[k][2]){

temp\_burst=x[k][2];

temp\_row=k;

}

}

k++;

}

x[temp\_row][2]-=1;

if(i==x[0][1])

printf("%5d-----------\n |p-%-4d|\n",i,x[temp\_row][0]);

else{

if(pre\_process\_row!=temp\_row)

printf("%5d-----------\n |p-%-4d|\n",i,x[temp\_row][0]);

}

pre\_process\_row=temp\_row;

if(x[temp\_row][2]==0){

done++;

y[s][0]=x[temp\_row][0];

y[s][1]=x[temp\_row][1];

y[s][2]=i;

y[s][3]=x[temp\_row][3];

y[s][4]=((i-x[temp\_row][1])-x[temp\_row][3])+1;

sum+=((i-x[temp\_row][1])-x[temp\_row][3])+1;

s++;

}

}

i++;

}

printf("%5d-----------\n",i);

cout<<endl<<endl;

cout<<"Table of processes with completion record as they were completed\n\n";

cout<<"\n\nProcess\tarrival\tFin\tTotal\tWait\n-------\t-------\t---\t-----\t----\n";

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

cout<<y[i][0]<<"\t"<<y[i][1]<<"\t"<<y[i][2]+1<<"\t"<<y[i] [3]<<"\t"<<y[i][4]<<"\n";

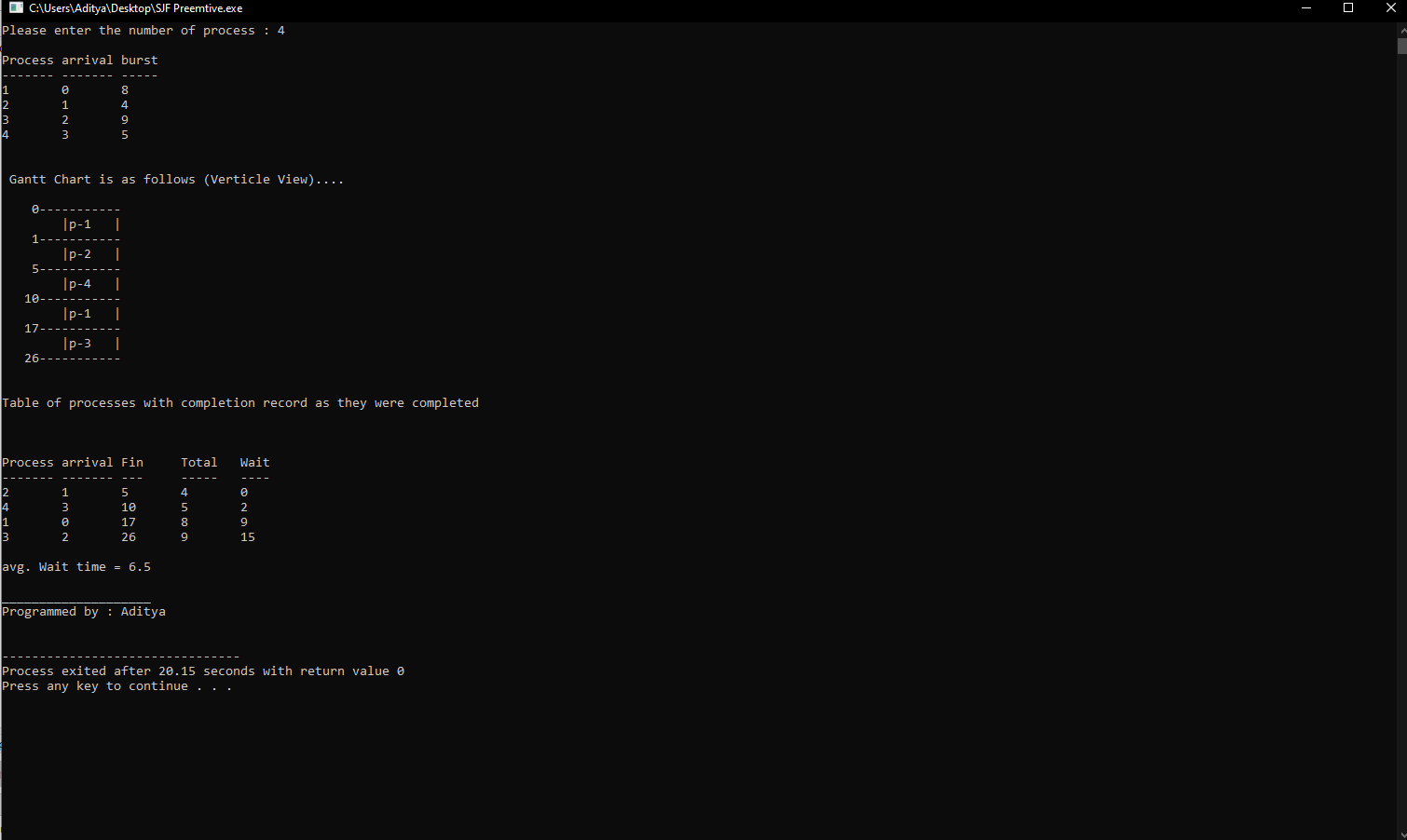
cout<<"\navg. Wait time = "<<sum/p<<endl<<endl;

cout<<"Programmed by : Aditya Hada 11801403 K18KH \n\n";

return 0;

}

***Output :-***



***Algorithm :-***

1.Start of Program.

2.Int main()

3.int i,j,k,p,s=0, got=0, idle=0, temp\_burst, temp\_row, pre\_process\_row, done=0;

float sum=0;

1. Print(“Enter no of process in variable N”)
2. For loop for taking the input from user
3. Input from user in form of 1. Process ID, 2.Arrival Time, 3. Burst Time.
4. While(done!=p)
5. {
6. got=0;
7. k=0;
8. while(k<p){
9. if(x[k][1]<=i){
10. if(x[k][2]!=0){
11. got=1;
12. temp\_burst=x[k][2];
13. temp\_row=k;
14. idle=0;
15. break;
16. }
17. else
18. k++;
19. }
20. else{
21. if(idle==0)
22. printf("%5d-----------\n |Idle |\n",i);
23. idle=1;
24. break;
25. }
26. }
27. if(got!=0){
28. k=0;
29. while(x[k][1]<=i && k<p){
30. if(x[k][2]!=0){
31. if(temp\_burst>x[k][2]){
32. temp\_burst=x[k][2];
33. temp\_row=k;
34. }
35. }
36. k++;
37. }
39. x[temp\_row][2]-=1;
41. if(i==x[0][1])
42. printf("%5d-----------\n |p-%-4d|\n",i,x[temp\_row][0]);
43. else{
44. if(pre\_process\_row!=temp\_row)
45. printf("%5d-----------\n |p-%-4d|\n",i,x[temp\_row][0]);
46. }
48. pre\_process\_row=temp\_row;
50. if(x[temp\_row][2]==0){
51. done++;
52. y[s][0]=x[temp\_row][0];
53. y[s][1]=x[temp\_row][1];
54. y[s][2]=i;
55. y[s][3]=x[temp\_row][3];
56. y[s][4]=((i-x[temp\_row][1])-x[temp\_row][3])+1;
57. sum+=((i-x[temp\_row][1])-x[temp\_row][3])+1;
58. s++;
59. }
60. }
61. i++;
62. }
63. Another while loop for Y-axis as Gantt chart is in 2-Dimensional;
64. for(i=0;i<s;i++)
65. print<<y[i][0]<<"\t"<<y[i][1]<<"\t"<<y[i][2]+1<<"\t"<<y[i] [3]<<"\t"<<y[i][4]<<"\n";

***Complexity :-***

Time Complexity of the whole Code or algorithm is N^2 in worst Case.

Therefore theta n^2 is Time complexity.