**Name:** Rochan Joshi

**Reg No:** 11810809

**Email Address:** rochanjoshi1752@gmail.com

**GitHub Link:**

**Code:**

#include<stdio.h>

struct process

{

char name;

int arr\_time, br\_time, comp\_time, wait\_time, turn\_ar\_time, priority ,status;

}queue[10];

int n;

void SortArrivalTime()

{

struct process temp;

int a, b;

for(a = 0; a < n - 1; a++)

{

for(b = a + 1; b < n; b++)

{

if(queue[a].arr\_time > queue[b].arr\_time)

{

temp = queue[a];

queue[a] = queue[b];

queue[b] = temp;

}

}

}

}

int main()

{

int l, time = 0, btime = 0, largest;

char c;

float wtime = 0, tatime = 0, avg\_wt, avg\_tat;

printf("\nEnter Total Number of Processes:\t");

scanf("%d", &n);

for(l = 0, c = 'A'; l < n; l++, c++)

{

queue[l].name = c;

printf("\nEnter Details For Process[%c]:\n", queue[l].name);

printf("Enter Arrival Time:\t");

scanf("%d", &queue[l].arr\_time );

printf("Enter Burst Time:\t");

scanf("%d", &queue[l].br\_time);

printf("Enter Priority:\t");

scanf("%d", &queue[l].priority);

queue[l].status = 0;

btime = btime + queue[l].br\_time;

}

SortArrivalTime();

queue[9].priority = -9999;

printf("\nProcess Name\tArrival Time\tBurst Time\tPriority\tWaiting Time");

for(time = queue[0].arr\_time; time < btime;)

{

largest = 9;

for(l = 0; l < n; l++)

{

if(queue[l].arr\_time <= time && queue[l].status != 1 && queue[l].priority > queue[largest].priority)

{

largest = l;

}

}

time = time + queue[largest].br\_time;

queue[largest].comp\_time = time;

queue[largest].wait\_time = queue[largest].comp\_time - queue[largest].arr\_time -queue[largest].br\_time;

queue[largest].turn\_ar\_time = queue[largest].comp\_time - queue[largest].arr\_time;

queue[largest].status = 1;

wtime = wtime + queue[largest].wait\_time;

tatime = tatime + queue[largest].turn\_ar\_time;

printf("\n%c\t\t%d\t\t%d\t\t%d\t\t%d",queue[largest].name,queue[largest].arr\_time,queue[largest].br\_time,queue[largest].priority,queue[largest].wait\_time);

}

avg\_wt = wtime/n;

avg\_tat = tatime/n;

printf("\n\nAverage waiting time:\t%f\n", avg\_wt);

printf("Average Turnaround Time:\t%f\n", avg\_tat);

}

**The Problem:**

Design a scheduler that uses a pre-emptive priority scheduling algorithm based on dynamically changing priority. Larger number for priority indicates higher priority.

Assume that the following processes with arrival time and service time wants to execute (for reference):

ProcessID         Arrival Time     Service Time

P1                        0                          4

P2                        1                          1

P3                        2                          2

P4                        3                          1

When the process starts execution (i.e. CPU assigned), priority for that process changes at the rate of m=1.When the process waits for CPU in the ready queue (but not yet started execution), its priority changes at a rate n=2. All the processes are initially assigned priority value of 0 when they enter ready queue for the first time. The time slice for each process is q = 1. When two processes want to join ready queue simultaneously, the process which has not executed recently is given priority. Calculate the average waiting time for each process. The program must be generic i.e. number of processes, their burst time and arrival time must be entered by user.

**Description:**

**Multi-level queue** scheduling algorithm is used in scenarios where the processes can be classified into groups based on property like process type, CPU time, IO access, memory size, etc. In a multi-level queue scheduling algorithm, there will be many queues depends on the number of groups the processes are classified into. Each queue will be assigned a priority and will have its own scheduling algorithm. For the process in a queue to execute, all the queues of priority higher than it should be empty, meaning the process in those high priority queues should have completed its execution. In this scheduling algorithm, once assigned to a queue, the process will not move to any other queues.

In multilevel queue scheduling, all processes have their own queue. Each queue has its own Scheduling algorithm. So, scheduling among the queues also required.

There are two ways to for scheduling of queues:

1. Fixed priority scheduling
2. Time slicing

**Algorithm:**

struct process (comp\_time, arr\_time, br\_time, c, wait\_time, turn\_ar\_time, priority, status)

**SortArrivalTime:**

1. Set: i:=0 j:=0
2. Set: struct process temp
3. Repeat the loop until i!=n:
   1. Repeat the loop from j=I to j!=n:
      1. Sort arrival time in increasing order
4. Exit

**MainAlgorithm**(n, c, avg\_wt, avg\_tat, largest)

1. Set: time: =0, btime: =0, wtime: =0.0, tatime: =0.0, l: =0
2. Declare struct process t []
3. Read n
4. Repeat the loop from l: =0 and c: =’A’ to l! =n:
   1. Read t[l]. arr\_time
   2. Read t[l].br\_time
   3. Read t[l]. priority
   4. btime: = btime+q. time
5. Call SortArrivalTime
6. Repeat loop from l=t.arr\_time until i! =btime:
   1. Repeat loop until i! =n:
      1. Set largest: = 9
      2. if t[i].arr\_time <= l and t[i].status!= 1 and t[i].priority > t[largest].priority
         1. largest=l;
      3. l: =l+t[largest].br\_time
      4. t[largest]. comp\_time: = l
      5. t[largest]. wait\_time: =t[largest].comp\_time-t[largest].arr\_time
      6. t[largest].status: =1
      7. wtime: =wtime+ t[largest].wait\_time
      8. tatime: =tatime+ t[largest].turn\_ar\_time
      9. print t[largest].name
      10. print t[largest].arr\_time
      11. print t[largest].br\_time
      12. print t[largest].priority
      13. print t[largest].wait\_time
7. avg\_wt: =wtime/n
8. avg\_tat: =tatime/n
9. print avg\_wt and avg\_tat
10. exit

**Output:**

