**Student Name: Aman kumar**

**Student ID : 11715964**

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

**GitHub Link:** <https://github.com/Assbomber/OS-Project>

**Code:**

The Code

|  |
| --- |
| #include<stdio.h> |
|  | #include<conio.h> |
|  |  |
|  | void function1(); |
|  | static int Total\_wait\_time,Average\_wait\_time; |
|  | void main() |
|  | { |
|  | function1(); |
|  | printf("Total Waiting Time=%d and Average Waiting Time=%f",Total\_wait\_time,Average\_wait\_time); |
|  | getch(); |
|  | } |
|  |  |
|  | void function1(){ |
|  | char p[10][5],temp[5]; |
|  | int i,j,pt[10],wt[10],totwt=0,pr[10],temp1,n; |
|  | float avgwt; |
|  | printf("Please enter the number of processes to create:"); |
|  | scanf("%d",&n); |
|  | for(i=0;i<n;i++){ |
|  | printf("Enter the NAME of process >> %d <<:",i+1); |
|  | scanf("%s",&p[i]); |
|  | printf("Enter the TIME for process >> %d <<:",i+1); |
|  | scanf("%d",&pt[i]); |
|  | printf("Enter the PRIORITY for process >> %d <<:",i+1); |
|  | scanf("%d",&pr[i]); |
|  |  |
|  |  |
|  | } |
|  | for(i=0;i<n-1;i++){ |
|  | for(j=i+1;j<n;j++){ |
|  | if(pr[i]>pr[j]){ |
|  | temp1=pr[i]; |
|  | pr[i]=pr[j]; |
|  | pr[j]=temp1; |
|  | temp1=pt[i]; |
|  | pt[i]=pt[j]; |
|  | pt[j]=temp1; |
|  | strcpy(temp,p[i]); |
|  | strcpy(p[i],p[j]); |
|  | strcpy(p[j],temp); |
|  | } |
|  | } |
|  | } |
|  | wt[0]=0; |
|  | for(i=1;i<n;i++){ |
|  | wt[i]=wt[i-1]+pt[i-1]; |
|  | totwt=totwt+wt[i]; |
|  | } |
|  | avgwt=(float)totwt/n; |
|  | printf("p\_name\t p\_time\t priority\t w\_time\n"); |
|  | for(i=0;i<n;i++){ |
|  | printf(" %s\t %d\t %d\t\t %d\n" ,p[i],pt[i],pr[i],wt[i]); |
|  | } |
|  | Total\_wait\_time=totwt; |
|  | Average\_wait\_time=avgwt; |
|  |  |
|  | } |

Description

**Description:**

CPU schedules N processes which arrive at different time intervals and each process is allocated the CPU for a specific user input time unit, processes are scheduled using a preemptive round robin scheduling algorithm. Each process must be assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes one task has priority 0. The length of a time quantum is T units, where T is the custom time considered as time quantum for processing. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue. Design a scheduler so that the task with priority 0 does not starve for resources and gets the CPU at some time unit to execute. Also compute waiting time, turn around.

Algorithm

**Algorithm:**

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 turn around time and burst time.  
   Waiting Time = Turn Around Time – Burst Time

***Calculating Waiting time:***

1- Create an array **rem\_bt[]** to keep track of remaining

burst time of processes. This array is initially a

copy of bt[] (burst times array)

2- Create another array **wt[]** to store waiting times

of processes. Initialize this array as 0.

3- Initialize time : t = 0

4- Keep traversing the all processes while all processes

are not done. Do following for i'th process if it is

not done yet.

a- If rem\_bt[i] > quantum

(i) t = t + quantum

(ii) bt\_rem[i] -= quantum;

c- Else // Last cycle for this process

(i) t = t + bt\_rem[i];

(ii) wt[i] = t - bt[i]

(ii) bt\_rem[i] = 0; // This process is over

Complexity

**Complexity:**

|  |
| --- |
| **for(i=0;i<n;i++){** |
| **printf("Enter the NAME of process >> %d <<:",i+1);** |
| **scanf("%s",&p[i]);** |
| **printf("Enter the TIME for process >> %d <<:",i+1);** |
| **scanf("%d",&pt[i]);** |
| **printf("Enter the PRIORITY for process >> %d <<:",i+1);** |
| **scanf("%d",&pr[i]);** |
|  |
| **}** |

The above piece of code throws a complexity of **O(n).**

|  |
| --- |
| **for(i=0;i<n-1;i++){** |
| **for(j=i+1;j<n;j++){** |
| **if(pr[i]>pr[j]){** |
| **temp1=pr[i];** |
| **pr[i]=pr[j];** |
| **pr[j]=temp1;** |
| **temp1=pt[i];** |
| **pt[i]=pt[j];** |
| **pt[j]=temp1;** |
| **strcpy(temp,p[i]);** |
| **strcpy(p[i],p[j]);** |
| **strcpy(p[j],temp);** |
| **}** |
| **}** |
| **}** |

The above piece of code throws complexity of **O(n^2).**

|  |
| --- |
| **for(i=1;i<n;i++){** |
| **wt[i]=wt[i-1]+pt[i-1];** |
| **totwt=totwt+wt[i];** |
| **}** |
|  |
| The above piece of code throws complexity of **O(n).** |
|  |

|  |
| --- |
| for(i=0;i<n;i++){ |
| printf(" %s\t %d\t %d\t\t %d\n" ,p[i],pt[i],pr[i],wt[i]); |
| } |

The above piece of code throws complexity of **O(n).**

**The Total Complexity of Algorithm hence becomes= O(n^2).**

Test-Cases

**TestCases:**

The First line takes the Input “n” as the number of TestCases.

|  |  |
| --- | --- |
| **INPUT** | **OUTPUT** |
| enter no of processes: 5  enter process1 name: aaa  enter process time: 4  enter priority:5  enter process2 name: bbb  enter process time: 3  enter priority:4  enter process3 name: ccc  enter process time: 2  enter priority:3  enter process4 name: ddd  enter process time: 5  enter priority:2  enter process5 name: eee  enter process time: 1  enter priority:1 | p\_name P\_time priority w\_time  eee 1 1 0  ddd 5 2 1  ccc 2 3 6  bbb 3 4 8  aaa 4 5 11  total waiting time=26  avg waiting time=5.20 |