# AIM

EXPERIMENT 1[c]

Simulate the CPU Scheduling algorithm using Priority Scheduling

# DESCRIPTION

**Priority Scheduling** is a method of scheduling processes that is based on priority. In this algorithm, the scheduler selects the tasks to work as per the priority. The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis. Priority depends upon memory requirements, time requirements, etc.

## Types of Priority Scheduling

Priority scheduling divided into two main types:

Preemptive Scheduling

In Preemptive Scheduling, the tasks are mostly assigned with their priorities. Sometimes it is important to run a task with a higher priority before another lower priority task, even if the lower priority task is still running. The lower priority task holds for some time and resumes when the higher priority task finishes its execution.

Non-Preemptive Scheduling

In this type of scheduling method, the CPU has been allocated to a specific process. The process that keeps the CPU busy, will release the CPU either by switching context or terminating. It is the only method that can be used for various hardware platforms. That’s because it doesn’t need special hardware (for example, a timer) like preemptive scheduling.

## Characteristics of Priority Scheduling

* A CPU algorithm that schedules processes based on priority.
* It used in Operating systems for performing batch processes.
* If two jobs having the same priority are READY, it works on a FIRST COME, FIRST SERVED basis.
* In priority scheduling, a number is assigned to each process that indicates its priority level.
* Lower the number, higher is the priority.
* In this type of scheduling algorithm, if a newer process arrives, that is having a higher priority than the currently running process, then the currently running process is preempted.

## Advantages of priority scheduling

* Easy to use scheduling method
* Processes are executed on the basis of priority so high priority does not need to wait for long which saves time
* This method provides a good mechanism where the relative important of each process may be precisely defined.
* Suitable for applications with fluctuating time and resource requirements.

## Disadvantages of priority scheduling

* If the system eventually crashes, all low priority processes get lost.
* If high priority processes take lots of CPU time, then the lower priority processes may starve and will be postponed for an indefinite time.
* This scheduling algorithm may leave some low priority processes waiting indefinitely.
* A process will be blocked when it is ready to run but has to wait for the CPU because some other process is running currently.
* If a new higher priority process keeps on coming in the ready queue, then the process which is in the waiting state may need to wait for a long duration of time.

. **ALGORITHM** START

Step 1 → First input the processes with their arrival time, burst time and priority.

Step 2 → First process will schedule, which have the lowest arrival time, if two or more processes will have lowest arrival time, then whoever has higher priority will schedule first.

Step 3 → Now further processes will be schedule according to the arrival time and priority of the process. (Here we are assuming that lower the priority number having higher priority). I f two process priority are same then sort according to process number.

Step 4 → Once all the processes have been arrived, we can schedule them based on their priority. STOP

# SOURCE CODE

#include<stdio.h> #include<conio.h> #include<string.h> void main()

{

int et[20],at[10],n,i,j,temp,p[10],st[10],ft[10],wt[10],ta[10]; int totwt=0,totta=0;

float awt,ata;

char pn[10][10],t[10];

//clrscr();

printf("Enter the number of process:"); scanf("%d",&n);

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

{

printf("Enter process name,arrivaltime,execution time & priority:");

//flushall(); scanf("%s%d%d%d",pn[i],&at[i],&et[i],&p[i]);

}

for(i=0; i<n; i++) for(j=0; j<n; j++)

{

if(p[i]<p[j])

{

temp=p[i]; p[i]=p[j]; p[j]=temp; temp=at[i]; at[i]=at[j]; at[j]=temp; temp=et[i]; et[i]=et[j]; et[j]=temp; strcpy(t,pn[i]);

strcpy(pn[i],pn[j]);

strcpy(pn[j],t);

}

}

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

{

if(i==0)

{

st[i]=at[i]; wt[i]=st[i]-at[i];

ft[i]=st[i]+et[i];

ta[i]=ft[i]-at[i];

}

else

{

st[i]=ft[i-1];

wt[i]=st[i]-at[i];

ft[i]=st[i]+et[i];

ta[i]=ft[i]-at[i];

}

totwt+=wt[i]; totta+=ta[i];

}

awt=(float)totwt/n; ata=(float)totta/n;

printf("\nPname\tarrivaltime\texecutiontime\tpriority\twaitingtime\ttatime"); for(i=0; i<n; i++)

printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],p[i],wt[i],ta[i]); printf("\nAverage waiting time is:%f",awt);

printf("\nAverage turnaroundtime is:%f",ata); getch();

}

# OUTPUT

