**OPERATING SYSTEM MINI PROJECT**

**AIM:** Program to implement all Process Scheduling Algorithms.

**THEORY:**

CPU scheduling deals with the problem of deciding which of the processes in the

ready queue is to be allocated the CPU. There are many different CPU-scheduling algorithms. In this section, we describe several of them.

**->Types of Process Scheduling Algorithms:**

**1.FCFS**

By far the simplest CPU-scheduling algorithm is the ﬁrst-come, ﬁrst-served (FCFS) scheduling algorithm. With this scheme, the process that requests the CPU ﬁrst is allocated the CPU ﬁrst. The implementation of the FCFS policy is easily managed with a FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue.

The code for FCFS scheduling is simple to write and understand.

On the negative side, the average waiting time under the FCFS policy is often quite long.

**2.SJF**

A different approach to CPU scheduling is the shortest-job-ﬁrst (SJF) scheduling algorithm. This algorithm associates with each process the length of the process’s next CPU burst. When the CPU is available, it is assigned to the

process that has the smallest next CPU burst. If the next CPU bursts of two

processes are the same, FCFS scheduling is used to break the tie. Note that a

more appropriate term for this scheduling method would be the *shortest-next-*

*CPU-burst* algorithm, because scheduling depends on the length of the next

CPU burst of a process, rather than its total length.

We use the term SJF because most people and textbooks use this term to refer to this type of scheduling

**3.PRIORITY SCHEDULING ALGORITHM**

The SJF algorithm is a special case of the general priority-scheduling algorithm. A priority is associated with each process, and the

CPU is allocated to the process with the highest priority. Equal-priority processes are scheduled in FCFS order.

An SJF algorithm is simply a priority algorithm where the priority (*p*)is the inverse of the (predicted) next CPU burst. The larger the CPU burst, the lower the priority, and vice versa. Note that we discuss scheduling in terms of *high* priority and *low* priority.

**4.RR**

The round-robin (RR) scheduling algorithm is designed especially for timesharing systems. It is similar to FCFS scheduling, but preemption is added to enable the system to switch between processes. A small unit of time, called a

time quantum or time slice,isdeﬁned.Atimequantumisgenerallyfrom10

to 100 milliseconds in length. The ready queue is treated as a circular queue.

The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1 time quantum.

To implement RR scheduling, we again treat the ready queue as a FIFO

queue of processes. New processes are added to the tail of the ready queue.

The CPU scheduler picks the ﬁrst process from the ready queue, sets a timer to

interrupt after 1 time quantum, and dispatches the process.

One of two things will then happen. The process may have a CPU burst of less than 1 time quantum. In this case, the process itself will release the

CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. If the CPU burst of the currently running process is longer than 1 time quantum, the timer will go off and will cause an interrupt to the operating

system. A context switch will be executed, and the process will be put at the

tail of the ready queue. The CPU scheduler will then select the next process in

the ready queue.

The average waiting time under the RR policy is often long.

**PROGRAM:**

#include<stdio.h>

void fcfs()

{

int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;

printf("Enter total number of processes(maximum 20):");

scanf("%d",&n);

printf("\nEnter Process Burst Time\n");

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

{

printf("P[%d]:",i+1);

scanf("%d",&bt[i]);

}

wt[0]=0;

for(i=1;i<n;i++)

{

wt[i]=0;

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

wt[i]+=bt[j];

}

printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");

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

{

tat[i]=bt[i]+wt[i];

avwt+=wt[i];

avtat+=tat[i];

printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);

}

avwt/=i;

avtat/=i;

printf("\n\nAverage Waiting Time:%d",avwt);

printf("\nAverage Turnaround Time:%d",avtat);

}

void sjf()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;

float avg\_wt,avg\_tat;

printf("Enter number of process:");

scanf("%d",&n);

printf("\nEnter Burst Time:\n");

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

{

printf("p%d:",i+1);

scanf("%d",&bt[i]);

p[i]=i+1;

}

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

{

pos=i;

for(j=i+1;j<n;j++)

{

if(bt[j]<bt[pos])

pos=j;

}

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0;

for(i=1;i<n;i++)

{

wt[i]=0;

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

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=(float)total/n;

total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");

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

{

tat[i]=bt[i]+wt[i];

total+=tat[i];

printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

avg\_tat=(float)total/n;

printf("\n\nAverage Waiting Time=%f",avg\_wt);

printf("\nAverage Turnaround Time=%f\n",avg\_tat);

}

void priority()

{

int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg\_wt,avg\_tat;

printf("Enter Total Number of Process:");

scanf("%d",&n);

printf("\nEnter Burst Time and Priority\n");

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

{

printf("\nP[%d]\n",i+1);

printf("Burst Time:");

scanf("%d",&bt[i]);

printf("Priority:");

scanf("%d",&pr[i]);

p[i]=i+1;

}

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

{

pos=i;

for(j=i+1;j<n;j++)

{

if(pr[j]<pr[pos])

pos=j;

}

temp=pr[i];

pr[i]=pr[pos];

pr[pos]=temp;

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0;

for(i=1;i<n;i++)

{

wt[i]=0;

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

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=total/n;

total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");

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

{

tat[i]=bt[i]+wt[i];

total+=tat[i];

printf("\nP[%d]\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

avg\_tat=total/n;

printf("\n\nAverage Waiting Time=%d",avg\_wt);

printf("\nAverage Turnaround Time=%d\n",avg\_tat);

}

void rr()

{

int count,j,n,time,remain,flag=0,time\_quantum;

int wait\_time=0,turnaround\_time=0,at[10],bt[10],rt[10];

printf("Enter Total Process:\t ");

scanf("%d",&n);

remain=n;

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

{

printf("Enter Arrival Time and Burst Time for Process Number %d :\n",count+1);

scanf("%d",&at[count]);

scanf("%d",&bt[count]);

rt[count]=bt[count];

}

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

scanf("%d",&time\_quantum);

printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");

for(time=0,count=0;remain!=0;)

{

if(rt[count]<=time\_quantum && rt[count]>0)

{

time+=rt[count];

rt[count]=0;

flag=1;

}

else if(rt[count]>0)

{

rt[count]-=time\_quantum;

time+=time\_quantum;

}

if(rt[count]==0 && flag==1)

{

remain--;

printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);

wait\_time+=time-at[count]-bt[count];

turnaround\_time+=time-at[count];

flag=0;

}

if(count==n-1)

count=0;

else if(at[count+1]<=time)

count++;

else

count=0;

}

printf("\nAverage Waiting Time= %f\n",wait\_time\*1.0/n);

printf("Avg Turnaround Time = %f",turnaround\_time\*1.0/n);

}

void main()

{

int i;

do

{

printf("\n\t\t\t\*\*\*PROCESS SHEDULING ALGORITHM\*\*\*\t\t\t");

printf("\n\n1:FCFS\n2.SJF\n3:PRIORITY\n4:ROUND ROBIN\n5:EXIT \nEnter your choice:");

scanf("%d",&i);

switch(i)

{

case 1:

fcfs();

break;

case 2:

sjf();

break;

case 3:

priority();

break;

case 4:

rr();

break;

case 5:

exit(1);

break;

default :

printf("wrong choice");

}

}while(1);

}

**OUTPUT:**

\*\*\*PROCESS SHEDULING ALGORITHM\*\*\*

1:FCFS 2.SJF 3:PRIORITY 4:ROUND ROBIN 5:EXIT

Enter your choice:1

Enter total number of processes(maximum 20):3

Enter Process Burst Time

P[1]:24 P[2]:3 P[3]:3

Process Burst Time Waiting Time Turnaround Time

P[1] 24 0 24

P[2] 3 24 27

P[3] 3 27 30

Average Waiting Time:17

Average Turnaround Time:27

\*\*\*PROCESS SHEDULING ALGORITHM\*\*\*

1:FCFS 2.SJF 3:PRIORITY 4:ROUND ROBIN 5:EXIT

Enter your choice:2

Enter number of process:4

Enter Burst Time:

p1:6 p2:8 p3:7 p4:3

Process Burst Time Waiting Time Turnaround Time

p4 3 0 3

p1 6 3 9

p3 7 9 16

p2 8 16 24

Average Waiting Time=7.000000

Average Turnaround Time=13.000000

\*\*\*PROCESS SHEDULING ALGORITHM\*\*\*

1:FCFS 2.SJF 3:PRIORITY 4:ROUND ROBIN 5:EXIT

Enter your choice:4

Enter Total Process: 4

Enter Arrival Time and Burst Time for Process Number 1 : 3 8

Enter Arrival Time and Burst Time for Process Number 2 : 0 9

Enter Arrival Time and Burst Time for Process Number 3 : 2 5

Enter Arrival Time and Burst Time for Process Number 4 : 0 6

Enter Time Quantum: 2

Process | Turnaround Time|Waiting Time

P[3] | 19 | 14

P[4] | 23 | 17

P[1] | 22 | 14

P[2] | 28 | 19

Average Waiting Time= 16.000000

Avg Turnaround Time = 23.000000

\*\*\*PROCESS SHEDULING ALGORITHM\*\*\*

1:FCFS 2.SJF 3:PRIORITY 4:ROUND ROBIN 5:EXIT

Enter your choice:5