|  |
| --- |
| **Subject: Operating System Sub teacher: Prof. S.S.Shete**  **Class: S.E. Computer Engg Roll no :-**  **Practical No.: Date:** |

**Title:** Write a program implements CPU scheduling algorithms**.**

**Aim:** Write a program for CPU scheduling algorithms.

**Theory:**

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms which we are going to discuss in this chapter a’

First-Come, First-Served (FCFS) Scheduling

Shortest-Job-Next (SJN) Scheduling

Priority Scheduling

Shortest Remaining Time

Round Robin(RR) Scheduling

Multiple-Level Queues Scheduling

These algorithms are either non-preemptive or preemptive. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

**First Come First Serve (FCFS)**

Jobs are executed on first come, first serve basis.

It is a non-preemptive, pre-emptive scheduling algorithm.

Easy to understand and implement.

Its implementation is based on FIFO queue.

Poor in performance as average wait time is high.

First Come First Serve Scheduling Algorithm

Wait time of each process is as follows âˆ’

Process Wait Time : Service Time - Arrival Time

P0 0 - 0 = 0

P1 5 - 1 = 4

P2 8 - 2 = 6

P3 16 - 3 = 13

Average Wait Time: (0+4+6+13) / 4 = 5.75

Shortest Job Next (SJN)

This is also known as shortest job first, or SJF

This is a non-preemptive, pre-emptive scheduling algorithm.

Best approach to minimize waiting time.

Easy to implement in Batch systems where required CPU time is known in advance.

Impossible to implement in interactive systems where required CPU time is not known.

**Shortest Job Next (SJN)**

This is also known as shortest job first, or SJF

This is a non-preemptive, pre-emptive scheduling algorithm.

Best approach to minimize waiting time.

Easy to implement in Batch systems where required CPU time is known in advance.

Impossible to implement in interactive systems where required CPU time is not known.

The processer should know in advance how much time process will take.

The processer should know in advance how much time process will take.

Round Robin is the preemptive process scheduling algorithm.

Each process is provided a fix time to execute, it is called a quantum.

Once a process is executed for a given time period, it is preempted and other process executes for a given time period.

Context switching is used to save states of preempted processes.

Round Robin Scheduling Algorithm

Wait time of each process is as follows âˆ’

Process Wait Time : Service Time - Arrival Time

P0 (0 - 0) + (12 - 3) = 9

P1 (3 - 1) = 2

P2 (6 - 2) + (14 - 9) + (20 - 17) = 12

P3 (9 - 3) + (17 - 12) = 11

Average Wait Time: (9+2+12+11) / 4 = 8.5

Multiple-Level Queues Scheduling

Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

Multiple queues are maintained for processes with common characteristics.

Each queue can have its own scheduling algorithms.

Priorities are assigned to each queue.

For example, CPU-bound jobs can be scheduled in one queue and all I/O-bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.

**Program:**

**FCFS:**

#include<stdio.h>

main()

{

int n,a[10],b[10],t[10],w[10],g[10],i,m;

float att=0,awt=0;

for(i=0;i<10;i++)

{

a[i]=0;

b[i]=0;

w[i]=0;

g[i]=0;

}

printf("enter the numberof process");

scanf("%d",&n);

printf("enter the burst time");

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

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

printf("\n enter the arrival times");

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

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

g[0]=0;

for(i=0;i<10;i++)

g[i+1]=g[i]+b[i];

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

{

w[i]=g[i]-a[i];

t[i]=g[i+1]-a[i];

awt=awt+w[i];

att=att+t[i];

}

awt=awt/n;

att=att/n;

printf("\n \tprocess\t waiting time\t turn arround time\n");

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

{

printf("\tp%d\t\t%d\t\t%d\n",i,w[i],t[i]);

}

printf("the average waiting time is %f\n",awt);

printf("the average turn arround time is %f\n",att);

}

**/\*output**

gcoe@GCOE-LINUX:~$ cd Desktop

gcoe@GCOE-LINUX:~/Desktop$ gcc psjf.c

gcoe@GCOE-LINUX:~/Desktop$ ./a.out

Enter no of processes : 4

Enter arrival time for process P1 : 2

Enter burst time for process P1 : 3

Enter arrival time for process P2 : 1

Enter burst time for process P2 : 3

Enter arrival time for process P3 : 4

Enter burst time for process P3 : 2

Enter arrival time for process P4 : 4

Enter burst time for process P4 : 2

Process |Turnaround Time| Waiting Time

P[2] | 3 | 0

P[3] | 2 | 0

P[4] | 4 | 2

P[1] | 9 | 6

average waiting time = 2.000000

average turnaround time = 4.500000gcoe@GCOE-LINUX:~/Desktop$ \*/

**RR:**

#include<stdio.h>

int main()

{

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 Process Number %d :",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);

return 0;

}

**//output//**

/\*gcoe@GCOE-LINUX:~/Desktop$ gcc RR.c

gcoe@GCOE-LINUX:~/Desktop$ ./a.out

Enter Total Process: 4

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

2

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

4

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

23

Enter Arrival Time and Burst Time for Process Process Number 4 :1

5

Enter Time Quantum: 2

Process |Turnaround Time|Waiting Time

P[1] | 1 | -1

P[2] | 4 | 0

P[4] | 16 | 11

P[3] | 29 | 6

Average Waiting Time= 4.000000

Avg Turnaround Time = 12.50000\*/

**SJF (Non Primative):**

#include<stdio.h>

int main()

{

int i,n,p[10]={1,2,3,4,5,6,7,8,9,10},min,k=1,btime=0;

int bt[10],temp,j,at[10],wt[10],tt[10],ta=0,sum=0;

float wavg=0,tavg=0,tsum=0,wsum=0;

printf(" -------Shortest Job First Scheduling ( NP )-------\n");

printf("\nEnter the No. of processes :");

scanf("%d",&n);

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

{

printf("\tEnter the burst time of %d process :",i+1);

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

printf("\tEnter the arrival time of %d process :",i+1);

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

}

/\*Sorting According to Arrival Time\*/

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

{

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

{

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

{

temp=p[j];

p[j]=p[i];

p[i]=temp;

temp=at[j];

at[j]=at[i];

at[i]=temp;

temp=bt[j];

bt[j]=bt[i];

bt[i]=temp;

}

}

}

/\*Arranging the table according to Burst time,

Execution time and Arrival Time

Arrival time <= Execution time

\*/

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

{

btime=btime+bt[j];

min=bt[k];

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

{

if (btime>=at[i] && bt[i]<min)

{

temp=p[k];

p[k]=p[i];

p[i]=temp;

temp=at[k];

at[k]=at[i];

at[i]=temp;

temp=bt[k];

bt[k]=bt[i];

bt[i]=temp;

}

}

k++;

}

wt[0]=0;

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

{

sum=sum+bt[i-1];

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

wsum=wsum+wt[i];

}

wavg=(wsum/n);

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

{

ta=ta+bt[i];

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

tsum=tsum+tt[i];

}

tavg=(tsum/n);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\n RESULT:-");

printf("\nProcess\t Burst\t Arrival\t Waiting\t Turn-around" );

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

{

printf("\n p%d\t %d\t %d\t\t %d\t\t\t%d",p[i],bt[i],at[i],wt[i],tt[i]);

}

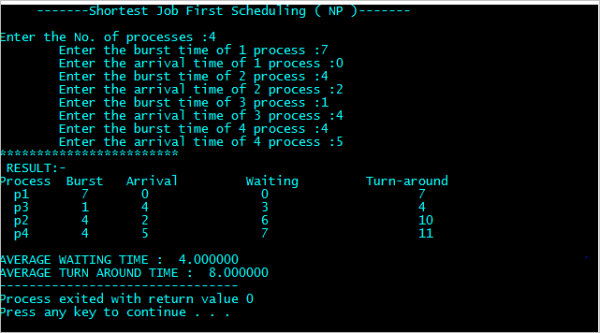
printf("\n\nAVERAGE WAITING TIME : %f",wavg);

printf("\nAVERAGE TURN AROUND TIME : %f",tavg);

return 0;

}

**OUTPUT:**



**Priority (Primative):**

**#include<stdio.h>**

**struct process**

**{**

**char process\_name;**

**int arrival\_time, burst\_time, ct, waiting\_time, turnaround\_time, priority;**

**int status;**

**}process\_queue[10];**

**int limit;**

**void Arrival\_Time\_Sorting()**

**{**

**struct process temp;**

**int i, j;**

**for(i = 0; i < limit - 1; i++)**

**{**

**for(j = i + 1; j < limit; j++)**

**{**

**if(process\_queue[i].arrival\_time > process\_queue[j].arrival\_time)**

**{**

**temp = process\_queue[i];**

**process\_queue[i] = process\_queue[j];**

**process\_queue[j] = temp;**

**}**

**}**

**}**

**}**

**void main()**

**{**

**int i, time = 0, burst\_time = 0, largest;**

**char c;**

**float wait\_time = 0, turnaround\_time = 0, average\_waiting\_time, average\_turnaround\_time;**

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

**scanf("%d", &limit);**

**for(i = 0, c = 'A'; i < limit; i++, c++)**

**{**

**process\_queue[i].process\_name = c;**

**printf("\nEnter Details For Process[%C]:\n", process\_queue[i].process\_name);**

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

**scanf("%d", &process\_queue[i].arrival\_time );**

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

**scanf("%d", &process\_queue[i].burst\_time);**

**printf("Enter Priority:\t");**

**scanf("%d", &process\_queue[i].priority);**

**process\_queue[i].status = 0;**

**burst\_time = burst\_time + process\_queue[i].burst\_time;**

**}**

**Arrival\_Time\_Sorting();**

**process\_queue[9].priority = -9999;**

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

**for(time = process\_queue[0].arrival\_time; time < burst\_time;)**

**{**

**largest = 9;**

**for(i = 0; i < limit; i++)**

**{**

**if(process\_queue[i].arrival\_time <= time && process\_queue[i].status != 1 && process\_queue[i].priority > process\_queue[largest].priority)**

**{**

**largest = i;**

**}**

**}**

**time = time + process\_queue[largest].burst\_time;**

**process\_queue[largest].ct = time;**

**process\_queue[largest].waiting\_time = process\_queue[largest].ct - process\_queue[largest].arrival\_time - process\_queue[largest].burst\_time;**

**process\_queue[largest].turnaround\_time = process\_queue[largest].ct - process\_queue[largest].arrival\_time;**

**process\_queue[largest].status = 1;**

**wait\_time = wait\_time + process\_queue[largest].waiting\_time;**

**turnaround\_time = turnaround\_time + process\_queue[largest].turnaround\_time;**

**printf("\n%c\t\t%d\t\t%d\t\t%d\t\t%d", process\_queue[largest].process\_name, process\_queue[largest].arrival\_time, process\_queue[largest].burst\_time, process\_queue[largest].priority, process\_queue[largest].waiting\_time);**

**}**

**average\_waiting\_time = wait\_time / limit;**

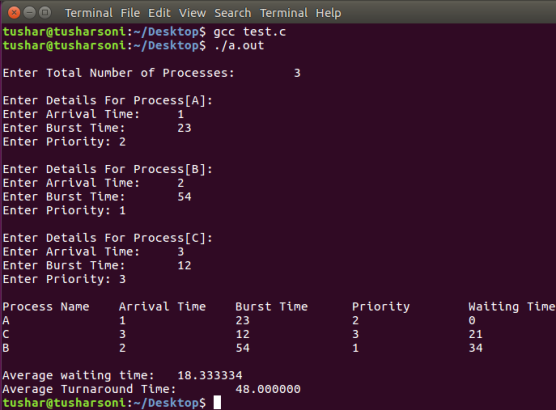
**average\_turnaround\_time = turnaround\_time / limit;**

**printf("\n\nAverage waiting time:\t%f\n", average\_waiting\_time);**

**printf("Average Turnaround Time:\t%f\n", average\_turnaround\_time);**

**}**

**OUTPUT:**



**Priority (Non primitive):**

#include<stdio.h>

int main()

{

  int i,j,n,time,sum\_wait=0,sum\_turnaround=0;

  int smallest,at[10],bt[10],priority[10],remain;

  printf("Enter no of Processes : ");

  scanf("%d",&n);

  remain=n;

  for(i=0;i<n;i++)

  {

    printf("Enter arrival time, burst time and priority for process p%d :",i+1);

    scanf("%d",&at[i]);

    scanf("%d",&bt[i]);

    scanf("%d",&priority[i]);

  }

  priority[9]=11;

  printf("\n\nProcess\t|Turnaround time|waiting time\n");

  for(time=0;remain!=0;)

  {

    smallest=9;

    for(i=0;i<n;i++)

    {

      if(at[i]<=time && priority[i]<priority[smallest] && bt[i]>0)

      {

        smallest=i;

      }

    }

    time+=bt[smallest];

    remain--;

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

    sum\_wait+=time-at[smallest]-bt[smallest];

    sum\_turnaround+=time-at[smallest];

    bt[smallest]=0;

  }

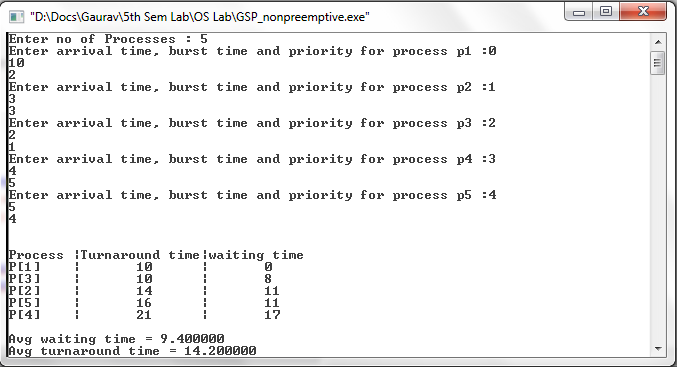
  printf("\nAvg waiting time = %f\n",sum\_wait\*1.0/n);

  printf("Avg turnaround time = %f",sum\_turnaround\*1.0/n);

  return 0;

}

**OUTPUT:**



**Conclusion:** Hence we have to perform “Cpu scheduling Algorithms such that fcfs, sjf (preemptive &non preemptive), round robin, priority.