**LIST OF THE EXPERIMENTS**

1. To implement CPU Scheduling Algorithms using C/C++ languages.

(1.1) FCFS

Code: -

#include<stdio.h>

// Function to find the waiting time for all

// processes

void findWaitingTime(int processes[], int n,

int bt[], int wt[])

{

// waiting time for first process is 0

wt[0] = 0;

// calculating waiting time

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

wt[i] = bt[i-1] + wt[i-1] ;

}

// Function to calculate turn around time

void findTurnAroundTime( int processes[], int n,

int bt[], int wt[], int tat[])

{

// calculating turnaround time by adding

// bt[i] + wt[i]

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

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

}

//Function to calculate average time

void findavgTime( int processes[], int n, int bt[])

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

//Function to find waiting time of all processes

findWaitingTime(processes, n, bt, wt);

//Function to find turn around time for all processes

findTurnAroundTime(processes, n, bt, wt, tat);

//Display processes along with all details

printf("Processes Burst time Waiting time Turn around time\n");

// Calculate total waiting time and total turn

// around time

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

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

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

printf(" %d ", bt[i] );

printf(" %d",wt[i] );

printf(" %d\n",tat[i] );

}

int s=(float)total\_wt / (float)n;

int t=(float)total\_tat / (float)n;

printf("Average waiting time = %d",s);

printf("\n");

printf("Average turn around time = %d ",t);

}

// Driver code

int main()

{

//process id's

int processes[] = { 1, 2, 3};

int n = sizeof processes / sizeof processes[0];

//Burst time of all processes

int burst\_time[] = {10, 5, 8};

findavgTime(processes, n, burst\_time);

return 0;

}

**Output: -**

Processes Burst time Waiting time Turnaround time

1 10 0 10

2 5 10 15

3 8 15 23

Average waiting time = 8

Average turnaround time = 16

(1.2) SJF

Code: -

#include<stdio.h>

int main()

{

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

}

//sorting of burst times

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;

//finding the waiting time of all the processes

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

{

wt[i]=0;

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

//individual WT by adding BT of all previous completed processes

wt[i]+=bt[j];

//total waiting time

total+=wt[i];

}

//average waiting time

avg\_wt=(float)total/n;

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

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

{

//turnaround time of individual processes

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

//total turnaround time

totalT+=tat[i];

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

}

//average turnaround time

avg\_tat=(float)totalT/n;

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

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

}

Output: -

Enter number of processes:3

Enter Burst Time:

p1:10

p2:5

p3:6

Process Burst Time Waiting Time Turnaround Time

p2 5 0 5

p3 6 5 11

p1 10 11 21

Average Waiting Time=5.333333

Average Turnaround Time=12.333333

(1.3) SRTF

Code: -

#include <stdio.h>

int main()

{

int arrival\_time[10], burst\_time[10], temp[10];

int i, smallest, count = 0, time, limit;

double wait\_time = 0, turnaround\_time = 0, end;

float average\_waiting\_time, average\_turnaround\_time;

printf("\n Enter the Total Number of Processes:- ");

scanf("%d", &limit);

printf("\nEnter Details of %d Processes:-", limit);

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

{

printf("\nEnter Arrival Time: ");

scanf("%d", &arrival\_time[i]);

printf("\nEnter Burst Time: ");

scanf("%d", &burst\_time[i]);

temp[i] = burst\_time[i];

}

burst\_time[9] = 9999;

for(time = 0; count != limit; time++)

{

smallest = 9;

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

{

if(arrival\_time[i] <= time && burst\_time[i] < burst\_time[smallest] && burst\_time[i] > 0)

{

smallest = i;

}

}

burst\_time[smallest]--;

if(burst\_time[smallest] == 0)

{

count++;

end = time + 1;

wait\_time = wait\_time + end - arrival\_time[smallest] - temp[smallest];

turnaround\_time = turnaround\_time + end - arrival\_time[smallest];

}

}

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

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

printf("\nAverage Waiting Time: %lf", average\_waiting\_time);

printf("\nAverage Turnaround Time: %lf", average\_turnaround\_time);

return 0;

}

OUTPUT: -

Enter the Total Number of Processes: - 3

Enter Details of 3 Processes: -

Enter Arrival Time: 0

Enter Burst Time: 10

Enter Arrival Time: 0

Enter Burst Time: 5

Enter Arrival Time: 0

Enter Burst Time: 6

Average Waiting Time: 5.333333

Average Turnaround Time: 12.333333

1.4 **Priority Scheduling**

#include <stdio.h>

//non-pre-emptive priority scheduling program in c

void swap(int \*a,int \*b)

{

int temp=\*a;

\*a=\*b;

\*b=temp;

}

int main()

{

int n;

printf("Enter Number of Processes: ");

scanf("%d",&n);

int burst[n],priority[n],index[n];

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

{

printf("Enter Burst Time and Priority Value for Process %d: ",i+1);

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

index[i]=i+1;

}

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

{

int temp=priority[i],m=i;

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

{

if(priority[j] > temp)

{

temp=priority[j];

m=j;

}

}

swap(&priority[i], &priority[m]);

swap(&burst[i], &burst[m]);

swap(&index[i],&index[m]);

}

int t=0;

printf("Order of process Execution is\n");

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

{

printf("P%d is executed from %d to %d\n",index[i],t,t+burst[i]);

t+=burst[i];

}

printf("\n");

printf("Process Id\tBurst Time\tWait Time\n");

int wait\_time=0;

int total\_wait\_time = 0;

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

{

printf("P%d\t\t%d\t\t%d\n",index[i],burst[i],wait\_time);

total\_wait\_time += wait\_time;

wait\_time += burst[i];

}

float avg\_wait\_time = (float) total\_wait\_time / n;

printf("Average waiting time is %f\n", avg\_wait\_time);

int total\_Turn\_Around = 0;

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

total\_Turn\_Around += burst[i];

}

float avg\_Turn\_Around = (float) total\_Turn\_Around / n;

printf("Average TurnAround Time is %f",avg\_Turn\_Around);

return 0;

}

**OUTPUT:**

Enter Number of Processes: 2

Enter Burst Time and Priority Value for Process 1: 5 3

Enter Burst Time and Priority Value for Process 2: 4 2

Order of process Execution is

P1 is executed from 0 to 5

P2 is executed from 5 to 9

Process Id Burst Time Wait Time

P1 5 0

P2 4 5

Average waiting time is 2.500000

Average TurnAround Time is 4.500000

1.5 **Round Robin Scheduling**

#include<stdio.h>

int main()

{

int i, limit, total = 0, x, counter = 0, time\_quantum;

int wait\_time = 0, turnaround\_time = 0, arrival\_time[10], burst\_time[10], temp[10];

float average\_wait\_time, average\_turnaround\_time;

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

scanf("%d", &limit);

x = limit;

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

{

printf("\nEnter Details of Process[%d]n", i + 1);

printf("\nArrival Time:");

scanf("%d", &arrival\_time[i]);

printf("\nBurst Time:");

scanf("%d", &burst\_time[i]);

temp[i] = burst\_time[i];

}

printf("\nEnter Time Quantum:");

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

printf("\nProcess\_ID\tBurst Time\tTurnaround Time\tWaiting Timen");

for(total = 0, i = 0; x != 0;)

{

if(temp[i] <= time\_quantum && temp[i] > 0)

{

total = total + temp[i];

temp[i] = 0;

counter = 1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - time\_quantum;

total = total + time\_quantum;

}

if(temp[i] == 0 && counter == 1)

{

x--;

printf("\nProcess[%d]\t%d\t %d\t\t %d", i + 1, burst\_time[i], total - arrival\_time[i], total - arrival\_time[i] - burst\_time[i]);

wait\_time = wait\_time + total - arrival\_time[i] - burst\_time[i];

turnaround\_time = turnaround\_time + total - arrival\_time[i];

counter = 0;

}

if(i == limit - 1)

{

i = 0;

}

else if(arrival\_time[i + 1] <= total)

{

i++;

}

else

{

i = 0;

}

}

average\_wait\_time = wait\_time \* 1.0 / limit;

average\_turnaround\_time = turnaround\_time \* 1.0 / limit;

printf("\nAverage Waiting Time:t%f", average\_wait\_time);

printf("\nAvg Turnaround Time:t%fn", average\_turnaround\_time);

return 0;

}

OUTPUT:

Enter Total Number of Processes:3

Enter Details of Process[1]n

Arrival Time:0

Burst Time:5

Enter Details of Process[3]n

Arrival Time:3

Burst Time:4

Enter Time Quantum:2

Process\_ID Burst Time Turnaround Time Waiting Time

Process[3] 4 9 5

Process[1] 5 13 8

Process[2] 7 15 8

Average Waiting Time:t7.000000

Avg Turnaround Time:t12.333333

2**. Paging**

* 1. FIFO

#include <stdio.h>

int main()

{

int incomingStream[] = {4 , 1 , 2 , 4 , 5};

int pageFaults = 0;

int frames = 3;

int m, n, s, pages;

pages = sizeof(incomingStream)/sizeof(incomingStream[0]);

printf(" Incoming \t Frame 1 \t Frame 2 \t Frame 3 ");

int temp[ frames ];

for(m = 0; m < frames; m++)

{

temp[m] = -1;

}

for(m = 0; m < pages; m++)

{

s = 0;

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

{

if(incomingStream[m] == temp[n])

{

s++;

pageFaults--;

}

}

pageFaults++;

if((pageFaults <= frames) && (s == 0))

{

temp[m] = incomingStream[m];

}

else if(s == 0)

{

temp[(pageFaults - 1) % frames] = incomingStream[m];

}

printf("\n");

printf("%d\t\t\t",incomingStream[m]);

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

{

if(temp[n] != -1)

printf(" %d\t\t\t", temp[n]);

else

printf(" - \t\t\t");

}

}

printf("\nTotal Page Faults: %d\n", pageFaults);

return 0;

}

OUTPUT: -

Incoming Frame 1 Frame 2 Frame 3

4 4 - -

1 4 1 -

2 4 1 2

4 4 1 2

5 5 1 2

Total Page Faults: 4

//**LRU** code in c

#include<stdio.h>

int findLRU(int time[], int n){

int i, minimum = time[0], pos = 0;

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

if(time[i] < minimum){

minimum = time[i];

pos = i;

}

}

return pos;

}

int main()

{

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;

printf("Enter number of frames: ");

scanf("%d", &no\_of\_frames);

printf("Enter number of pages: ");

scanf("%d", &no\_of\_pages);

printf("Enter reference string: ");

for(i = 0; i < no\_of\_pages; ++i){

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

}

for(i = 0; i < no\_of\_frames; ++i){

frames[i] = -1;

}

for(i = 0; i < no\_of\_pages; ++i){

flag1 = flag2 = 0;

for(j = 0; j < no\_of\_frames; ++j){

if(frames[j] == pages[i]){

counter++;

time[j] = counter;

flag1 = flag2 = 1;

break;

}

}

if(flag1 == 0){

for(j = 0; j < no\_of\_frames; ++j){

if(frames[j] == -1){

counter++;

faults++;

frames[j] = pages[i];

time[j] = counter;

flag2 = 1;

break;

}

}

}

if(flag2 == 0){

pos = findLRU(time, no\_of\_frames);

counter++;

faults++;

frames[pos] = pages[i];

time[pos] = counter;

}

printf("\n");

for(j = 0; j < no\_of\_frames; ++j){

printf("%d\t", frames[j]);

}

}

printf("\n\nTotal Page Faults = %d", faults);

return 0;

}

Output:

Enter number of frames: 3

Enter number of pages: 6

Enter reference string: 5 7 5 6 7 3

5 -1 -1

5 7 -1

5 7 -1

5 7 6

5 7 6

3 7 6

Total Page Faults = 4

// **Simulate Paging Technique of Memory Management**

#include<stdio.h>

#include<conio.h>

int main()

{

int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;

int s[10], fno[10][20];

//clrscr();

printf("\nEnter the memory size -- ");

scanf("%d",&ms);

printf("\nEnter the page size -- ");

scanf("%d",&ps);

nop = ms/ps;

printf("\nThe no. of pages available in memory are -- %d ",nop);

printf("\nEnter number of processes -- ");

scanf("%d",&np);

rempages = nop;

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

{

printf("\nEnter no. of pages required for p[%d]-- ",i);

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

if(s[i] > rempages)

{

printf("\nMemory is Full");

break;

}

rempages = rempages - s[i];

printf("\nEnter pagetable for p[%d] --- ",i);

for(j=0;j<s[i];j++)

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

}

printf("\nEnter Logical Address to find Physical Address ");

printf("\nEnter process no. and pagenumber and offset -- ");

scanf("%d %d %d",&x,&y, &offset);

if(x>np || y>=s[i] || offset>=ps)

printf("\nInvalid Process or Page Number or offset");

else

{ pa=fno[x][y]\*ps+offset;

printf("\nThe Physical Address is -- %d",pa);

}

return 0;

getch();

}

OUTPUT:

INPUT

Enter the memory size – 1000 Enter the page size -- 100

The no. of pages available in memory are -- 10

Enter number of processes -- 3

Enter no. of pages required for p[1]-- 4

Enter pagetable for p[1] --- 8 6

9

5

Enter no. of pages required for p[2]-- 5

Enter pagetable for p[2] --- 1 4 5 7 3

Enter no. of pages required for p[3]-- 5

OUTPUT

Memory is Full

Enter Logical Address to find Physical Address Enter process no. and pagenumber and offset -- 2

3

60

The Physical Address is -- 760