1. Implementation of First come first served

#include<stdio.h>

int main()

{

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("Enter Process Burst Timen");

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("Process Burst Time Waiting Time Turnaround 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\n",i+1,bt[i],wt[i],tat[i]);

}

avwt/=i;

avtat/=i;

printf("Average Waiting Time:%d",avwt);

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

return 0;

}

1. Implementation of shortest job first

#include<stdio.h>

int main()

{

int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,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++)

{

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

{

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

{

temp=bt[i];

bt[i]=bt[j];

bt[j]=temp;

temp=p[i];

p[i]=p[j];

p[j]=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("\nProcesst Burst Time Waiting Time Turnaround 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%d",p[i],bt[i],wt[i],tat[i]);

}

avg\_tat=(float)total/n;

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

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

}

4. Write a program to implement round robin scheduling algorithm.

#include<stdio.h>

void main()

{

// initlialize the variable name

int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];

float avg\_wt, avg\_tat;

printf(" Total number of process in the system: ");

scanf("%d", &NOP);

y = NOP; // Assign the number of process to variable y

// Use for loop to enter the details of the process like Arrival time and the Burst Time

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

{

printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);

printf(" Arrival time is: \t"); // Accept arrival time

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

printf(" \nBurst time is: \t"); // Accept the Burst time

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

temp[i] = bt[i]; // store the burst time in temp array

}

// Accept the Time qunat

printf("Enter the Time Quantum for the process: \t");

scanf("%d", &quant);

// Display the process No, burst time, Turn Around Time and the waiting time

printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");

for(sum=0, i = 0; y!=0; )

{

if(temp[i] <= quant && temp[i] > 0) // define the conditions

{

sum = sum + temp[i];

temp[i] = 0;

count=1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - quant;

sum = sum + quant;

}

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

{

y--; //decrement the process no.

printf("\nProcess No[%d] \t\t %d\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]);

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

tat = tat+sum-at[i];

count =0;

}

if(i==NOP-1)

{

i=0;

}

else if(at[i+1]<=sum)

{

i++;

}

else

{

i=0;

}

}

// represents the average waiting time and Turn Around time

avg\_wt = wt \* 1.0/NOP;

avg\_tat = tat \* 1.0/NOP;

printf("\n Average Turn Around Time: \t%f", avg\_wt);

printf("\n Average Waiting Time: \t%f", avg\_tat);

}

5. Write a program to implement deadlock detection algorithm.

// Banker's Algorithm

#include<stdio.h>

int main()

{

int n , m , i , j , k,avail[10],alloc[10][10],max[10][10];

printf("\nEnter the no of process: ");

scanf("%d",&n);

printf("\nEnter the no of resources: ");

scanf("%d",&m);

// Available Resources

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

{

printf("\nTotal Amount of the Resource R%d: ",i+1);

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

}

printf("\nEnter the request matrix:");

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

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

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

printf("\nEnter the allocation matrix:");

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

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

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

int f[n] , ans[n] , ind = 0 ;

for (k = 0; k < n; k++) {

f[k] = 0;

}

int need[n][m];

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

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

need[i][j] = max[i][j] - alloc[i][j] ;

}

int y = 0;

for (k = 0; k < 5; k++){

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

if (f[i] == 0){

int flag = 0;

for (j = 0; j < m; j++) {

if(need[i][j] > avail[j]){

flag = 1;

break;

}

}

if ( flag == 0 ) {

ans[ind++] = i;

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

avail[y] += alloc[i][y] ;

f[i] = 1;

}

}

}

}

int flag = 1;

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

{

if(f[i] == 0)

{

flag = 0;

printf(" The following system is not safe ");

break;

}

}

if (flag == 1)

{

printf(" Following is the SAFE Sequence \ n ");

for (i = 0; i < n - 1; i++)

printf(" P%d -> " , ans[i]);

printf(" P%d ", ans[n - 1]);

printf("\nDeadlock will not occur");

}

else

{

printf("Deadlock occurs");

}

return(0);

}

6. Program to simulate first come first serve disk scheduling

#include <stdio.h>

#include <math.h>

int main()

{

int n,seek\_count = 0,cur\_track, distance,head = 50;

printf("Enter the size of n ");

scanf("%d",&n);

int arr[n];

printf("Enter the value of array ");

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

{

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

}

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

{

cur\_track = arr[i];

// calculate absolute distance

distance = fabs(head - cur\_track);

// increase the total count

seek\_count += distance;

// accessed track is now new head

head = cur\_track;

}

printf("Total number of seek operations: %d\n",seek\_count);

// Seek sequence would be the same

// as request array sequence

printf("Seek Sequence is\n");

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

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

}

return 0;

}

7. Program to simulate shortest seek time first disk scheduling.

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

int main()

{

int i,n,k,req[50],mov=0,cp,index[50],min,a[50],j=0,mini,cp1;

printf("enter the current position\n");

scanf("%d",&cp);

printf("enter the number of requests\n");

scanf("%d",&n);

cp1=cp;

printf("enter the request order\n");

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

{

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

}

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

{

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

{

index[i]=abs(cp-req[i]); // calculate distance of each request from current position

}

// to find the nearest request

min=index[0];

mini=0;

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

{

if(min>index[i])

{

min=index[i];

mini=i;

}

}

a[j]=req[mini];

j++;

cp=req[mini]; // change the current position value to next request

req[mini]=999;

} // the request that is processed its value is changed so that it is not processed again

printf("Sequence is : ");

printf("%d",cp1);

mov=mov+abs(cp1-a[0]); // head movement

printf(" -> %d",a[0]);

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

{

mov=mov+abs(a[i]-a[i-1]); ///head movement

printf(" -> %d",a[i]);

}

printf("\n");

printf("total head movement = %d\n",mov);

}

8. Program to implement SCAN disk scheduling algorithm

#include<stdio.h>

int main()

{

int queue[20],n,head,i,j,k,seek=0,max,diff,temp,queue1[20],queue2[20],

temp1=0,temp2=0;

float avg;

printf("Enter the max range of disk\n");

scanf("%d",&max);

printf("Enter the initial head position\n");

scanf("%d",&head);

printf("Enter the size of queue request\n");

scanf("%d",&n);

printf("Enter the queue of disk positions to be read\n");

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

{

scanf("%d",&temp);

if(temp>=head)

{

queue1[temp1]=temp;

temp1++;

}

else

{

queue2[temp2]=temp;

temp2++;

}

}

for(i=0;i<temp1-1;i++)

{

for(j=i+1;j<temp1;j++)

{

if(queue1[i]>queue1[j])

{

temp=queue1[i];

queue1[i]=queue1[j];

queue1[j]=temp;

}

}

}

for(i=0;i<temp2-1;i++)

{

for(j=i+1;j<temp2;j++)

{

if(queue2[i]<queue2[j])

{

temp=queue2[i];

queue2[i]=queue2[j];

queue2[j]=temp;

}

}

}

for(i=1,j=0;j<temp1;i++,j++)

queue[i]=queue1[j];

queue[i]=max;

for(i=temp1+2,j=0;j<temp2;i++,j++)

queue[i]=queue2[j];

queue[i]=0;

queue[0]=head;

for(j=0;j<=n+1;j++)

{

diff=abs(queue[j+1]-queue[j]);

seek+=diff;

printf("Disk head moves from %d to %d with seek %d\n",queue[j],queue[j+1],diff);

}

printf("Total seek time is %d\n",seek);

avg=seek/(float)n;

printf("Average seek time is %f\n",avg);

return 0;

}