OPERATING SYSTEM LAB (CSE-317)

SUBMITTED BY:-

NAME: - SHUBHAM BAUSKAR

SCHOLAR No. : - 151112221

SECTION : - CSE-2

**MAULANA AZAD**

**NATIONAL INSTITUTE OF TECHNOLOGY, BHOPAL**

Q.1) Write a program in C language (or the language in your choice) to simulate UniProgramming. Your program should take as an input execution trace of three programs (i.e. CPU Burst and I/O Burst activities) and should output.

a) Total Turn Around Time

b) Average Turn Around Time

c) CPU Utilization

d) I/O Utilization

e) Throughput

Answer:-

#include<bits/stdc++.h>

using namespace std;

struct process

{

int id;

int cpu1,cpu2;

int io;

int completion;

int turn,wait;

};

int main()

{

struct process p[3];

cout<<"Enter the CPU IO CPU Burst for 3 Processes:-"<<endl;

for(int i=0;i<3;i++)

{

cin>>p[i].cpu1>>p[i].io>>p[i].cpu2;

p[i].id=i;

}

int ti=0,tf=0;

float avgT=0,avgW=0;

cout<<endl<<"GANTT CHART:-"<<endl;

cout<<"id Completion TurnAT WaitT"<<endl;

float cpuburst=0;

for(int i=0;i<3;i++)

{

tf=ti+p[i].cpu1+p[i].io+p[i].cpu2;

cpuburst+=p[i].cpu1+p[i].cpu2;

p[i].completion=tf;

p[i].turn=p[i].completion;

if(i==0)

p[i].wait=0;

else

p[i].wait=p[i-1].completion;

cout<<"P"<<p[i].id<<" "<<p[i].completion<<" "<<p[i].turn<<" "<<p[i].wait<<" "<<endl;

ti=tf;

avgT+=p[i].turn;

avgW+=p[i].wait;

}

cout<<endl<<"Total Turn Around Time="<<int(avgT)<<endl;

cout<<"Average Turn Around Time="<<avgT/3<<endl;

cout<<"Average Waiting Time="<<avgW/3<<endl;

cout<<"CPU Utilization="<<cpuburst\*100/tf<<endl;

cout<<"IO Utilization="<<100-(cpuburst\*100)/tf<<endl;

cout<<"ThroughPut="<<3.0/tf<<endl;

return 0;

}

------------------------------------------------------------------------------------------------------------------------------------------

Q.2) Write a program in C language (or the language in your choice) to simulate Multiprogramming. Your program should take as an input execution trace of three programs (i.e. CPU Burst and I/O Burst activities) and should output.

a) Total Turn Around Time

b) Average Turn Around Time

c) CPU Utilization

d) I/O Utilization

e) Throughput

Answer:-

#include<bits/stdc++.h>

using namespace std;

struct process

{

int id;

int cpu1,cpu2;

int io;

int completion;

int turn;

int io\_cpu;//0=CPU1 && 1=I/O 2=CPU2

int status\_io\_cpu;//0=CPU 1=I/O;

int terminated;//1=Terminated 0=Not Terminated

};

int select\_process\_for\_cpu(struct process p[],float tf)

{

for(int i=0;i<3;i++)

{

if(p[i].status\_io\_cpu==1 && p[i].terminated==0)//Process is Performing IO

{

if(p[i].completion+p[i].io<=tf)//Process has completed its IO burst

{

p[i].status\_io\_cpu=0;

p[i].io\_cpu=2;

}

}

}

for(int i=0;i<3;i++)

{

if(p[i].terminated==0 && p[i].status\_io\_cpu==0)//Process has CPU burst left

return i;

}

int terminate=1;

for(int i=0;i<3;i++)

{

if(p[i].terminated==0)

terminate=0;

}

if(terminate==1)

return -1;

return 100;

}

int main()

{

struct process p[3];

cout<<"Enter the CPU IO CPU Burst for 3 Processes:-"<<endl;

for(int i=0;i<3;i++)

{

cin>>p[i].cpu1>>p[i].io>>p[i].cpu2;

p[i].id=i;

p[i].io\_cpu=0;//First Burst in CPU

p[i].status\_io\_cpu=0;

p[i].terminated=0;

p[i].completion=0;

}

float avgT=0,io=0;

int tf=0;

cout<<endl<<"GANTT CHART:-"<<endl;

cout<<"id Completion TurnAT "<<endl;

float cpuburst=0;

while(true)

{

int id=select\_process\_for\_cpu(p,tf);

if(id==100)

{

tf++;

continue;

}

if(id==-1)

break;

if(p[id].io\_cpu==2)//Last CPU Burst

{

p[id].terminated=1;

tf+=p[id].cpu2;

}

if(p[id].io\_cpu==0)//First CPU Burst

{

p[id].status\_io\_cpu=1;

tf+=p[id].cpu1;

p[id].io\_cpu=1;

}

p[id].completion=tf;

}

for(int i=0;i<3;i++)

{

p[i].turn=p[i].completion;

avgT+=p[i].turn;

cpuburst+=p[i].cpu1+p[i].cpu2;

io+=p[i].io;

}

for(int i=0;i<3;i++)

cout<<p[i].id<<"\t"<<p[i].completion<<"\t"<<" "<<p[i].turn<<endl;

cout<<endl<<"Total Turn Around Time="<<int(avgT)<<endl;

cout<<"Average Turn Around Time="<<avgT/3<<endl;

cout<<"CPU Utilization="<<cpuburst\*100/tf<<endl;

cout<<"IO Utilization="<<io\*100/tf<<endl;

cout<<"ThroughPut="<<3.0/tf<<endl;

return 0;

}

------------------------------------------------------------------------------------------------------------------------------------------

Q.6) Write a program in C language (or the language in your choice) to simulate File allocation strategies. Your program should take as an input starting block and length of the file and should output whether allocation is possible or not for following file storage strategies.

a) Sequential File allocation strategies

b) Indexed File allocation strategies

Answer:-

#include<bits/stdc++.h>

using namespace std;

struct file

{

int id;

int size;

int start\_block;

int allocation\_status;

};

struct file\_index

{

int id;

int size;

int index\_block;

vector<int> block\_occupy;

int allocation\_status;

};

void sequential\_allocation(struct file f[],int m,int n,int block[],int st\_block[])

{

for(int i=0;i<m;i++)//Try to Allocate each File

{

int low=f[i].start\_block;

int high=f[i].start\_block+f[i].size-1;

int flag=1;

for(int j=low;j<=high;j++)

{

if(st\_block[j]==1)//Block is Full

{

flag=0;

break;

}

}

if(flag==0)//No Allocation Possible

f[i].allocation\_status=0;

else//Allocation Possible

{

for(int j=low;j<=high;j++)

{

block[j]=i;

st\_block[j]=1;

}

f[i].allocation\_status=1;

}

}

}

bool is\_possible\_allocate(int st\_block[],int count,int n)

{

int temp=0;

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

{

if(st\_block[i]==-1)

temp++;

}

if(temp>=count)

return true;

return false;

}

void index\_allocation(struct file\_index fi[],int m,int n,int block[],int st\_block[])

{

for(int i=0;i<m;i++)//Try to Allocate Each File

{

bool flag=is\_possible\_allocate(st\_block,fi[i].size+1,n);

if(flag==false)//Allocation Not Possible

fi[i].allocation\_status=0;

else//Allocation Possible

{

fi[i].allocation\_status=1;

int temp=0,j;

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

{

if(st\_block[j]==-1)//Block is Vacant assign file

{

block[j]=i;

st\_block[j]=1;

fi[i].block\_occupy.push\_back(j);

temp++;

}

if(temp==fi[i].size)

break;

}

while(j<n)

{

if(st\_block[j]==-1)//Block is Vacant assign index\_block

{

fi[i].index\_block=j;

fi[i].block\_occupy.push\_back(j);

st\_block[j]=1;

block[j]=i;

break;

}

j++;

}

}

}

}

int main()

{

cout<<"Enter Number of Blocks in the Disk:-"<<endl;

int n;

cin>>n;

int block[n],st\_block[n];

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

st\_block[i]=block[i]=-1;

cout<<"Enter the Number of Files:-"<<endl;

int m;

cin>>m;

struct file f[m];

struct file\_index fi[m];

cout<<"Enter <Starting-Block Size> of File for Sequential Allocation Strategy:-"<<endl;

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

{

f[i].id=fi[i].id=i;

cin>>f[i].start\_block>>f[i].size;

fi[i].size=f[i].size;

f[i].allocation\_status=fi[i].allocation\_status=0;

fi[i].index\_block=-1;

}

sequential\_allocation(f,m,n,block,st\_block);

cout<<endl<<"SEQUENTIAL ALLOCATION:-"<<endl;

cout<<"File-Id Status:-"<<endl;

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

{

cout<<i<<"\t";

if(f[i].allocation\_status==0)

cout<<"Not Allocated";

else

cout<<"Allocated";

cout<<endl;

}

cout<<"Disk Block Status:-"<<endl;

cout<<"Block"<<" "<<"File Id"<<endl;

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

{

cout<<i<<"\t";

if(block[i]==-1)

cout<<"-"<<" ";

else

cout<<block[i]<<" ";

cout<<endl;

}

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

block[i]=st\_block[i]=-1;

index\_allocation(fi,m,n,block,st\_block);

cout<<endl<<"INDEX ALLOCATION:-"<<endl;

cout<<"File-Id Status Blocks-Occupied:-"<<endl;

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

{

cout<<i<<"\t";

if(fi[i].allocation\_status==0)

cout<<"Not Allocated";

else

{

cout<<"Allocated";

cout<<"\t";

for(int j=0;j<fi[i].block\_occupy.size()-1;j++)

cout<<fi[i].block\_occupy[j]<<" ";

cout<<" Index="<<fi[i].block\_occupy[fi[i].block\_occupy.size()-1];

}

cout<<endl;

}

cout<<"Disk Block Status:-"<<endl;

cout<<"Block"<<" "<<"File Id"<<endl;

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

{

cout<<i<<"\t";

if(block[i]==-1)

cout<<"-"<<" ";

else

cout<<block[i]<<" ";

cout<<endl;

}

return 0;

}

------------------------------------------------------------------------------------------------------------------------------------------

Q.7) Write a program in C language (or the language in your choice) to simulate Banker’s algorithm for Deadlock Avoidance. Your program should take as an input execution trace of five processes (i.e allocation matrix and max matrix) should output whether safe state or not ?

Answer:-

#include<bits/stdc++.h>

using namespace std;

bool find(vector<int> &v,int high,int id)

{

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

{

if(v[i]==id)

return true;

}

return false;

}

int main()

{

cout<<"Enter the Number of Processes:-"<<endl;

int n;

cin>>n;

int m;

cout<<"Enter the Number of Resources:-"<<endl;

cin>>m;

int max\_need[n][m];//Maximum Need of Resources

cout<<"Enter the Maximum Need of Resources to all Process:-"<<endl;

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

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

cin>>max\_need[i][j];

cout<<"Enter the Current Resource Allocation:-"<<endl;

int curr\_allocation[n][m];//Current Allocation of Resources

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

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

cin>>curr\_allocation[i][j];

cout<<"Current Available Resources:-"<<endl;

int available[m];

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

cin>>available[i];

int remaining\_need[n][m];//Remaining need of the Resources

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

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

remaining\_need[i][j]=max\_need[i][j]-curr\_allocation[i][j];

int order[n];

cout<<"Enter the Process Id's(Execution Trace):-"<<endl;

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

cin>>order[i];

cout<<"Remaining Need of Processes:-"<<endl;

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

{

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

cout<<remaining\_need[i][j]<<" ";

cout<<endl;

}

int status[n];

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

status[i]=0;

int flag=1;

vector<int> curr\_satisfied\_process;

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

{

int id=order[i];

for(int j=0;j<n;j++)//For each incomplete Process Check

{

int flagt=-1;//Assuming Process can Now be Satisfied

if(status[j]==0)

{

flagt=1;

for(int x=0;x<m;x++)

{

if(remaining\_need[j][x]>available[x])

{

flagt=0;

break;

}

}

}

if(flagt==1)//This Process Can be Satisfied

curr\_satisfied\_process.push\_back(j);

}

if(find(curr\_satisfied\_process,curr\_satisfied\_process.size()-1,id)==false)

{//Wrong Execution Order

flag=0;

break;

}

else//till Now Order is Correct

{

for(int t=0;t<m;t++)

available[t]+=max\_need[id][t];

status[id]=1;

}

curr\_satisfied\_process.clear();

}

if(flag==1)//Safe Execution Order Found

cout<<"\*\* Safe Execution Order \*\*"<<endl;

else

cout<<"\*\* No Safe Execution Order \*\*"<<endl;

return 0;

}

------------------------------------------------------------------------------------------------------------------------------------------

Q.8) Write a program in C language (or the language in your choice) to simulate Memory management scheme. Your program should take as an input memory size i.e. block size and size of block and should output whether allocation is possible or not for following Memory management scheme

a) First Fit

b) Best Fit

c) Worst Fit

Answer:-

#include<bits/stdc++.h>

using namespace std;

struct process

{

int id;

int size;

int partition\_no\_first;

int partition\_no\_best;

int partition\_no\_worst;

};

struct partition

{

int size;

int status;

};

void first\_fit(struct process p[],int n,struct partition par[],int m)

{

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

{

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

{

if(par[j].status==0 && par[j].size>=p[i].size)//Partition is Empty & can Allocate the Process

{

par[j].status=1;

p[i].partition\_no\_first=j;

break;

}

}

}

}

void worst\_fit(struct process p[],int n,struct partition par[],int m)

{

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

{

int maxi=-1;

//Find the Maximum Size Frame Which is Sufficient to Accommodate the Process

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

{

if(par[j].status==0 && par[j].size>=p[i].size)//Partition is Empty & can Allocate the Process

{

if(maxi==-1)//Allocate that Frame but search for Next Best Frame

maxi=j;

else if(par[maxi].size<par[j].size)

maxi=j;

}

}

if(maxi!=-1)//Suitable Partition Found

{

par[maxi].status=1;

p[i].partition\_no\_worst=maxi;

}

}

}

void best\_fit(struct process p[],int n,struct partition par[],int m)

{

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

{

int mini=-1;

//Find the Minimum Size Frame Which is Sufficient to Accommodate the Process

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

{

if(par[j].status==0 && par[j].size>=p[i].size)//Partition is Empty & can Allocate the Process

{

if(mini==-1)//Allocate that Frame but search for Next Best Frame

mini=j;

else if(par[mini].size>par[j].size)

mini=j;

}

}

if(mini!=-1)//Suitable Partition Found

{

par[mini].status=1;

p[i].partition\_no\_best=mini;

}

}

}

int main()

{

cout<<"Enter the Number of Processes:-"<<endl;

int n;

cin>>n;

cout<<"Enter Process-id & Process Size:-"<<endl;

struct process p[n];

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

{

cin>>p[i].id>>p[i].size;

p[i].partition\_no\_first=p[i].partition\_no\_best=p[i].partition\_no\_worst=-1;

}

cout<<"Enter Number of Partitions:-"<<endl;

int m;

cin>>m;

struct partition par[m];

int partition\_status[m];

cout<<"Enter Size & Status{1=Filled,0=Empty} of Each Partition:-"<<endl;

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

{

cin>>par[i].size>>par[i].status;

partition\_status[i]=par[i].status;

}

first\_fit(p,n,par,m);

cout<<endl<<"Process Status in Memory using First Fit Algorithm:-"<<endl;

cout<<"Id"<<"\t"<<"Size"<<"\t"<<"Partition No."<<endl;

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

{

cout<<p[i].id<<"\t"<<p[i].size<<" ";

if(p[i].partition\_no\_first==-1)

cout<<"\t"<<"Not Allocated"<<endl;

else

cout<<"\t"<<" "<<p[i].partition\_no\_first<<endl;

}

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

par[i].status=partition\_status[i];

best\_fit(p,n,par,m);

cout<<endl<<"Process Status in Memory using Best Fit Algorithm:-"<<endl;

cout<<"Id"<<"\t"<<"Size"<<"\t"<<"Partition No."<<endl;

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

{

cout<<p[i].id<<"\t"<<p[i].size<<" ";

if(p[i].partition\_no\_best==-1)

cout<<"\t"<<"Not Allocated"<<endl;

else

cout<<"\t"<<" "<<p[i].partition\_no\_best<<endl;

}

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

par[i].status=partition\_status[i];

worst\_fit(p,n,par,m);

cout<<endl<<"Process Status in Memory using Worst Fit Algorithm:-"<<endl;

cout<<"Id"<<"\t"<<"Size"<<"\t"<<"Partition No."<<endl;

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

{

cout<<p[i].id<<"\t"<<p[i].size<<" ";

if(p[i].partition\_no\_worst==-1)

cout<<"\t"<<"Not Allocated"<<endl;

else

cout<<"\t"<<" "<<p[i].partition\_no\_worst<<endl;

}

return 0;

}

------------------------------------------------------------------------------------------------------------------------------------------

Q.9) Write a program in C language (or the language in your choice) to simulate Page replacement algorithms. Your program should take as an input number of frames and pages and should output number of page faults for following algorithms

a) FIFO

b) LRU

c) LFU

Answer:-

#include<bits/stdc++.h>

using namespace std;

bool find\_page\_frame(int f,int st[],int frame[],int pg\_no)

{

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

{

if(st[i]==1 && frame[i]==pg\_no)

return true;

}

return false;

}

void fifo(int p,int f,int st[],vector<int> &req\_page,int frame[])

{

int pos\_replaced=0,page\_fault=0;

vector<int> page;

for(int i=0;i<req\_page.size();i++)

{

int pos=find\_page\_frame(f,st,frame,req\_page[i]);

if(pos==false)//Page Fault Apply FIFO Page Replacement Algorithm

{

frame[pos\_replaced]=req\_page[i];

st[pos\_replaced]=1;

pos\_replaced=(pos\_replaced+1)%f;

page\_fault++;

page.push\_back(req\_page[i]);

}

}

cout<<endl<<"FIFO:-"<<endl;

cout<<endl<<"No. of Page Faults="<<page\_fault<<endl;

cout<<"Pages For Which Page Fault Occurs:-"<<endl;

for(int i=0;i<page.size();i++)

cout<<page[i]<<" ";

cout<<endl;

}

int getid(int f,int frame\_st[])

{

int max=-1,maxi=-1;

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

{

if(max<frame\_st[i])//Not Frequently used

{

max=frame\_st[i];

maxi=i;

}

}

return maxi;

}

void optimal\_lfu(int p,int f,int st[],vector<int> &req\_page,int frame[])

{

int page\_fault=0,pos\_replaced=0;

vector<int> page;

for(int i=0;i<req\_page.size();i++)

{

int pos=find\_page\_frame(f,st,frame,req\_page[i]);

if(pos==false)//Page Fault Apply FIFO Page Replacement Algorithm

{

page\_fault++;

page.push\_back(req\_page[i]);

if(pos\_replaced<f)//Frames are Vacant

{

frame[pos\_replaced]=req\_page[i];

st[pos\_replaced]=1;

pos\_replaced++;

}

else//Frames are Full Vacant them by using LFU

{

for(int j=0;j<f;j++)

cout<<frame[j]<<" ";

cout<<endl;

int frame\_st[f];

for(int j=0;j<f;j++)

frame\_st[j]=-1;

int max=0;

for(int j=i+1;j<req\_page.size();j++)

{

for(int x=0;x<f;x++)

{

if(frame[x]==req\_page[j] && frame[x]==-1)

{

frame\_st[x]=max;

max++;

}

}

}

int pos=getid(f,frame\_st);

frame[pos]=req\_page[i];

st[pos]=1;

}

}

}

cout<<endl<<"OPTIMAL\_LFU:-"<<endl;

cout<<"No. of Page Faults="<<page\_fault<<endl;

cout<<"Pages For Which Page Fault Occurs:-"<<endl;

for(int i=0;i<page.size();i++)

cout<<page[i]<<" ";

cout<<endl;

}

void lru(int p,int f,int st[],vector<int> &req\_page,int frame[])

{

int page\_fault=0,pos\_replaced=0;

vector<int> page;

for(int i=0;i<req\_page.size();i++)

{

int pos=find\_page\_frame(f,st,frame,req\_page[i]);

if(pos==false)//Page Fault Apply FIFO Page Replacement Algorithm

{

page\_fault++;

page.push\_back(req\_page[i]);

if(pos\_replaced<f)//Frames are Vacant

{

frame[pos\_replaced]=req\_page[i];

st[pos\_replaced]=1;

pos\_replaced++;

}

else//Frames are Full Vacant them by using LFU

{

for(int j=0;j<f;j++)

cout<<frame[j]<<" ";

cout<<endl;

int frame\_st[f];

for(int j=0;j<f;j++)

frame\_st[j]=-1;

int max=0;

for(int j=i-1;j>=0;j--)

{

for(int x=0;x<f;x++)

{

if(frame[x]==req\_page[j] && frame[x]==-1)

{

frame\_st[x]=max;

max++;

}

}

}

int pos=getid(f,frame\_st);

frame[pos]=req\_page[i];

st[pos]=1;

}

}

}

cout<<endl<<"LRU:-"<<endl;

cout<<"No. of Page Faults="<<page\_fault<<endl;

cout<<"Pages For Which Page Fault Occurs:-"<<endl;

for(int i=0;i<page.size();i++)

cout<<page[i]<<" ";

cout<<endl;

}

int main()

{

cout<<"Enter Number of Pages of Process & Frames in Memory:-"<<endl;

int p,f;

cin>>p>>f;

int st[f],frame[f];

cout<<"Enter Number of Requests:-"<<endl;

int req;

cin>>req;

vector<int> req\_page;

cout<<"Enter the Page Needed in Range[0,p-1] :-"<<endl;

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

{

int x;

cin>>x;

req\_page.push\_back(x);

}

for(int i=0;i<f;i++)//All Frames are Empty at Beginning

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

fifo(p,f,st,req\_page,frame);

for(int i=0;i<f;i++)//All Frames are Empty at Beginning

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

optimal\_lfu(p,f,st,req\_page,frame);

lru(p,f,st,req\_page,frame);

return 0;

}

------------------------------------------------------------------------------------------------------------------------------------------

Q.10) Write a program in C language (or the language in your choice) to simulate Disk scheduling algorithms. Your program should take as an input range of disk, initial head position and queue requests and should output total seek time and average seek time for following algorithms.

a) SSTF

b) C-SCAN

c) FCFS

d) SCAN

e) LOOK

f) C-LOOK

Answer:-

#include<bits/stdc++.h>

using namespace std;

bool find(vector<int> &v,int x)

{

for(int i=0;i<v.size();i++)

{

if(v[i]==x)

return true;

}

return false;

}

void fcfs(vector<int> &v,int n,int head)

{

int totalseektime=0;

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

{

totalseektime+=abs(head-v[i]);

head=v[i];

}

cout<<"Total Seek Time="<<totalseektime<<endl;

cout<<"Average Seek Time="<<1.0\*totalseektime/n<<endl;

}

void scan(vector<int> &v,int n,int head,int low,int high)

{

int headi=head;

int totalseektime=0,left=n;

while(left!=0)

{

int flag=1;

for(int i=head;i<=high;i++)

{

if(find(v,i)==true)

{

totalseektime+=abs(head-i);

head=i;

left--;

if(left==0)

{

flag=0;

break;

}

}

}

totalseektime+=high-head;

totalseektime+=high-headi;

head=headi;

for(int i=headi;i>=low;i--)

{

if(find(v,i)==true)

{

totalseektime+=abs(head-i);

head=i;

left--;

if(left==0)

{

flag=0;

break;

}

}

}

if(flag==0)

break;

}

cout<<"Total Seek Time="<<totalseektime<<endl;

cout<<"Average Seek Time="<<1.0\*totalseektime/n<<endl;

}

void look(vector<int> &v,int n,int head,int low,int high)

{

int headi=head;

int min=v[0],max=v[0];

for(int i=1;i<v.size();i++)

{

if(v[i]<min)

min=v[i];

if(v[i]>max)

max=v[i];

}

high=max;

low=min;

int totalseektime=0,left=n;

while(left!=0)

{

int flag=1;

for(int i=head;i<=high;i++)

{

if(find(v,i)==true)

{

totalseektime+=abs(head-i);

head=i;

left--;

if(left==0)

{

flag=0;

break;

}

}

}

totalseektime+=high-head;

totalseektime+=high-headi;

head=headi;

for(int i=headi;i>=low;i--)

{

if(find(v,i)==true)

{

totalseektime+=abs(head-i);

head=i;

left--;

if(left==0)

{

flag=0;

break;

}

}

}

if(flag==0)

break;

}

cout<<"Total Seek Time="<<totalseektime<<endl;

cout<<"Average Seek Time="<<1.0\*totalseektime/n<<endl;

}

void cscan(vector<int> &v,int n,int head,int low,int high)

{

int headi=head;

int totalseektime=0,left=n;

while(left!=0)

{

int flag=1;

for(int i=head;i<=high;i++)

{

if(find(v,i)==true)

{

totalseektime+=abs(head-i);

head=i;

left--;

if(left==0)

{

flag=0;

break;

}

}

}

totalseektime+=high-head;

totalseektime+=high-low+1;

head=0;

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

{

if(find(v,i)==true)

{

totalseektime+=abs(head-i);

head=i;

left--;

if(left==0)

{

flag=0;

break;

}

}

}

if(flag==0)

break;

}

cout<<"Total Seek Time="<<totalseektime<<endl;

cout<<"Average Seek Time="<<1.0\*totalseektime/n<<endl;

}

void clook(vector<int> &v,int n,int head,int low,int high)

{

int headi=head;

int totalseektime=0,left=n;

int min=v[0],max=v[0];

for(int i=1;i<v.size();i++)

{

if(v[i]<min)

min=v[i];

if(v[i]>max)

max=v[i];

}

high=max;

low=min;

while(left!=0)

{

int flag=1;

for(int i=head;i<=high;i++)

{

if(find(v,i)==true)

{

totalseektime+=abs(head-i);

head=i;

left--;

if(left==0)

{

flag=0;

break;

}

}

}

totalseektime+=high-head;

totalseektime+=max-min;

head=min;

for(int i=min;i<headi;i++)

{

if(find(v,i)==true)

{

totalseektime+=abs(head-i);

head=i;

left--;

if(left==0)

{

flag=0;

break;

}

}

}

if(flag==0)

break;

}

cout<<"Total Seek Time="<<totalseektime<<endl;

cout<<"Average Seek Time="<<1.0\*totalseektime/n<<endl;

}

void sstf(vector<int> v,int n,int head)

{

int st[n],totalseektime=0;

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

st[i]=0;

int left=n;

while(left!=0)

{

//Calculate least Head Movement

int hm=9999999,pos=-1;

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

{

if(st[i]==0)//Not Served

{

int x=abs(head-v[i]);

if(x<hm)

{

hm=x;

pos=i;

}

}

}

if(pos!=-1)

{

totalseektime+=abs(head-v[pos]);

head=v[pos];

st[pos]=1;

left--;

if(left==0)

break;

}

else

break;

}

cout<<"Total Seek Time="<<totalseektime<<endl;

cout<<"Average Seek Time="<<1.0\*totalseektime/n<<endl;

}

int main()

{

cout<<"Enter input Range of Disk"<<endl;

int low,high;

cin>>low>>high;

cout<<"Enter initial Head Position:-"<<endl;

int head;

cin>>head;

cout<<"Enter No. of Queue Request"<<endl;

int n;

cin>>n;

cout<<"Enter the Request:-"<<endl;

vector<int> v;

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

{

int x;

cin>>x;

v.push\_back(x);

}

cout<<endl<<"FCFS:-"<<endl;

fcfs(v,n,head);

cout<<endl<<"SCAN(direction=right):-"<<endl;

scan(v,n,head,low,high);

cout<<endl<<"CSCAN(direction=right):-"<<endl;

cscan(v,n,head,low,high);

cout<<endl<<"SSTF:-"<<endl;

sstf(v,n,head);

cout<<endl<<"LOOK(direction=right):-"<<endl;

look(v,n,head,low,high);

cout<<endl<<"CLOOK(direction=right):-"<<endl;

clook(v,n,head,low,high);

return 0;

}

------------------------------------------------------------------------------------------------------------------------------------------

Q.11) Write a program in C language (or the language in your choice) to simulate FCFS Algorithm?

Answer:-

#include<bits/stdc++.h>

using namespace std;

struct process

{

int id,arrival;

int cpu,completion;

int wait,turn;

};

int partition(struct process p[],int low,int high)

{

int loc=low,left=low,right=high;

while(true)

{

while(p[loc].arrival<=p[right].arrival && loc!=right)

right--;

if(loc==right)

break;

else//Swap

{

struct process temp=p[loc];

p[loc]=p[right];

p[right]=temp;

loc=right;

}

while(p[loc].arrival>=p[left].arrival && loc!=left)

left++;

if(loc==left)

break;

else//Swap

{

struct process temp=p[loc];

p[loc]=p[left];

p[left]=temp;

loc=left;

}

}

return loc;

}

void quicksort(struct process p[],int low,int high)

{

if(low<high)

{

int par=partition(p,low,high);

quicksort(p,low,par-1);

quicksort(p,par+1,high);

}

}

int main()

{

cout<<"Enter the Number of Processes:-"<<endl;

int n;

cin>>n;

cout<<"Enter < Process-id Arrival-Time CPU-Burst > :-"<<endl;

struct process p[n];

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

cin>>p[i].id>>p[i].arrival>>p[i].cpu;

quicksort(p,0,n-1);

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

cout<<p[i].id<<" ";

cout<<endl;

int ti=p[0].arrival,tf;

float avgT=0,avgW=0;

cout<<"GANTT CHART:-"<<endl;

cout<<"Id Arrival Completion TurnAT WaitT"<<endl;

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

{

if(p[i].arrival>ti)

{

ti++;

tf++;

i--;

continue;

}

tf=ti+p[i].cpu;

p[i].completion=tf;

p[i].turn=p[i].completion-p[i].arrival;

p[i].wait=p[i].turn-p[i].cpu;

cout<<"P"<<p[i].id<<" "<<p[i].arrival<<" "<<p[i].completion<<" "<<p[i].turn<<" "<<p[i].wait<<" "<<endl;

ti=tf;

avgT+=p[i].turn;

avgW+=p[i].wait;

}

avgT=avgT/n;

avgW=avgW/n;

cout<<"Average Turn Around Time="<<avgT<<endl;

cout<<"Average Waiting Time="<<avgW<<endl;

return 0;

}

------------------------------------------------------------------------------------------------------------------------------------------

Q12. Implement Bankers algorithm for deadlock avoidance. Assume three resources and 5 processes and take inputs as maximum need, current allocation, available resources?

Answer:-

#include<bits/stdc++.h>

using namespace std;

int main()

{

cout<<"Enter the Number of Processes:-"<<endl;

int n;

cin>>n;

int m;

cout<<"Enter the Number of Resources:-"<<endl;

cin>>m;

int max\_need[n][m];//Maximum Need of Resources

cout<<"Enter the Maximum Need of Resources to all Process:-"<<endl;

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

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

cin>>max\_need[i][j];

cout<<"Enter the Current Resource Allocation:-"<<endl;

int curr\_allocation[n][m];//Current Allocation of Resources

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

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

cin>>curr\_allocation[i][j];

cout<<"Current Available Resources:-"<<endl;

int available[m];

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

cin>>available[i];

int remaining\_need[n][m];//Remaining need of the Resources

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

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

remaining\_need[i][j]=max\_need[i][j]-curr\_allocation[i][j];

vector<int> order;

int status[n];

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

status[i]=0;

int it;

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

{

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

{

if(status[j]==0)//Not Full-Filled the Request

{

int flag=1;//Assuming at this instance all its need for resources can be fulfilled

for(int t=0;t<m;t++)

{

if(available[t]<remaining\_need[j][t])

{

flag=0;

break;

}

}

if(flag==1)//Execute this Process by FullFilling its need for Resources & Deallocate all the Resources

{

for(int t=0;t<m;t++)//Allocating the Resources

available[t]-=remaining\_need[j][t];

for(int t=0;t<m;t++)

available[t]+=max\_need[j][t];

status[j]=1;

order.push\_back(j);

break;

}

}

}

}

if(order.size()==n)//Safe Execution Order Found

{

cout<<"Safe Execution Order:-"<<endl;

for(int i=0;i<order.size();i++)

cout<<"P"<<order[i]<<" ";

cout<<endl;

}

else

cout<<"No Safe Execution Order Found"<<endl;

return 0;

}

------------------------------------------------------------------------------------------------------------------------------------------