**2018UCO1665**

**Rahul Agarwal**

**COE-3**

**Sem 4**

**OS tutorial file**

**CEC14**

**­­­­­­­**

**INDEX**

|  |  |
| --- | --- |
| 1. | Fork()-> To create a child process |
| 2. | Wait()-> To keep a process waiting until it child terminates |
| 3. | Status()-> To check the status with which child process ended |
| 4. | Getppid()-> To print a process id |
| 5. | CPU scheduling-> First Come First Serve |
| 6. | CPU scheduling-> Non-Preemptive Shortest Job First |
| 7. | CPU scheduling-> Preemptive Shortest Job First |
| 8. | CPU scheduling-> Priority First Come First Serve |
| 9. | CPU scheduling-> Round Robin |
| 10. | Reader’s Writer’s Problem |
| 11. | Dining Philosopher Problem |
| 12. | Banker Algorithm |
| 13. | Page Replacement-> First In First Out |
| 14. | Page Replacement-> Least Recently Used |
| 15. | Page Replacement-> Optimal |
| 16. | File Management |
| 17. | Disk Scheduling-> First Come First Serve |
| 18. | Disk Scheduling->Shortest Seek Time First |
| 19. | Disk Scheduling-> Scan |
| 20. | Disk Scheduling-> C-Scan |
| 21. | Disk Scheduling-> Look |
| 22. | Disk Scheduling-> C-Look |

**FORK**

#include<iostream>

#include<sys/types.h>

#include<unistd.h>

using namespace std;

int main()

{

int check=fork();

if(check==0)//child

{

cout<<"I am child process";

}

else if(check>0)

{

cout<<"\nI am the parent process";

}

else

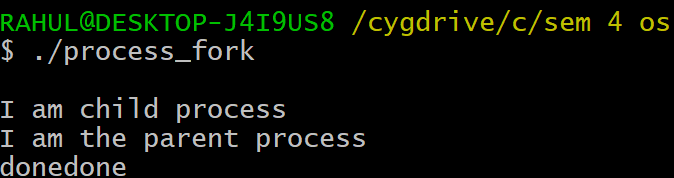
{

cout<<"Child cannot be created";

}

cout<<"\ndone";

}



**WAIT PROCESS**

#include<iostream>

#include<sys/wait.h>

#include<unistd.h>

using namespace std;

int main()

{

cout<<"Parent process creating child process\n";

if(fork()>0)

{

cout<<"I am parent process and will wait till my child completes the counting\n";

pid\_t id= wait(NULL);

cout<<"exiting parent after child whose id is:"<<id<<endl;

}

else

{

cout<<"I am child process and will count from 1 till 10 and then exit\n";

for(int i=1;i<=10;i++)

{

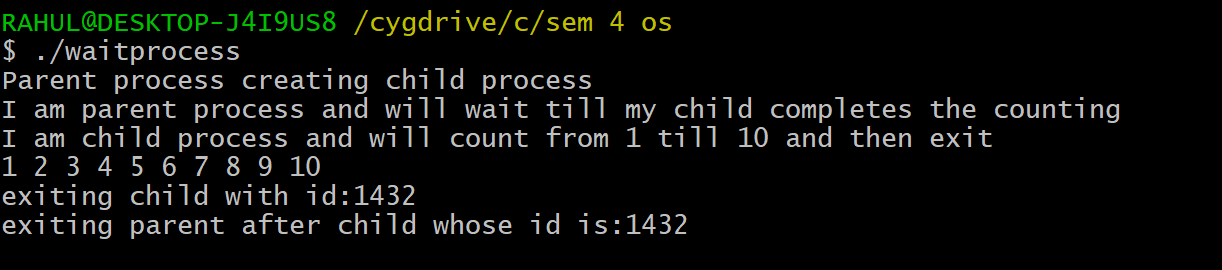
cout<<i<<" ";

}

cout<<"\nexiting child with id:"<<getpid()<<endl;

}

}



**STATUS**

#include<iostream>

#include<sys/wait.h>

#include<sys/types.h>

#include<unistd.h>

using namespace std;

int main()

{

if(fork()==0)

{

cout<<"Child process id:"<<getpid()<<" performing some commands"<<endl;

//execl("/bin/sh", "bin/sh", "-c", "./wrongpath", "NULL");

///////comment this line to see the status when child process works fine

}

else

{

int child\_id;

int status;

child\_id=wait(&status);

if(WIFEXITED(status)==true)

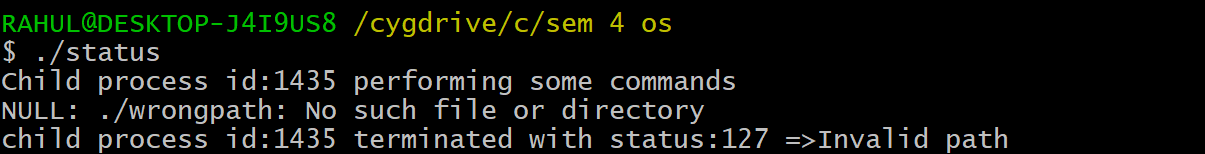
{cout<<"child process id:"<<child\_id<<" terminated with status:"<<WEXITSTATUS(status);

if(WEXITSTATUS(status)==0)

{cout<<" =>Child process commands successful";}

if(WEXITSTATUS(status)==127)

{cout<<" =>Invalid path";}}}}



**PRINT ID**

#include<iostream>

#include<sys/types.h>

#include<unistd.h>

using namespace std;

int main()

{

cout<<"Process id:"<<getpid()<<endl;

cout<<"Creating child process"<<endl;

if(fork()==0)

{

int p\_pid=getppid();

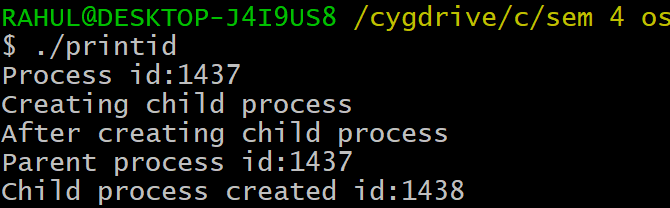
cout<<"After creating child process"<<endl;

cout<<"Parent process id:"<<p\_pid<<endl;

cout<<"Child process created id:"<<getpid()<<endl;

}

}



**FIRST COME FIRST SERVE**

#include<iostream>

using namespace std;

void findWaitingTime(int processes[], int n,

int bt[], int wt[])

{wt[0] = 0;

for (int i = 1; i < n ; i++ ) wt[i] = bt[i-1] + wt[i-1] ;}

void findTurnAroundTime( int processes[], int n,int bt[], int wt[], int tat[])

{for (int i = 0; i < n ; i++)tat[i] = bt[i] + wt[i];}

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

{

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

findWaitingTime(processes, n, bt, wt);

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

cout << "Processes "<< " Burst time "

<< " Waiting time " << " Turn around time\n";

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

{

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

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

cout << " " << i+1 << "\t\t" << bt[i] <<"\t "<< wt[i] <<"\t\t " << tat[i] <<endl;

}

cout << "Average waiting time = "<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "

<< (float)total\_tat / (float)n;

}

int main()

{ int n;

cout <<"enter the number of processes"<<endl;

cin>>n;

int processes[100];

cout<<"enter the processes"<<endl;

for(int i = 0 ; i<n ; i++){cin>>processes[i];}

cout<<"enter the bursts times"<<endl;

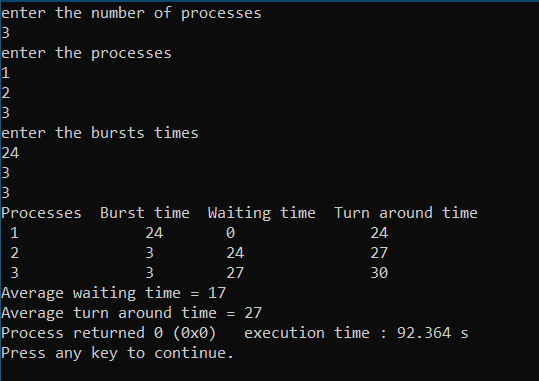
int burst\_time[100] ;

for(int i = 0 ; i<n ; i++){cin>>burst\_time[i];}

solution(processes, n, burst\_time);

return 0;

}



**NON-PREEMPTIVE-SJF**

#include <bits/stdc++.h>

using namespace std;

class process

{

public:

int burst\_time,arrival\_time,index;

};

bool cmp(process a,process b)

{

if(a.arrival\_time==b.arrival\_time)

{return a.burst\_time<b.burst\_time; }

else

{return a.arrival\_time<b.arrival\_time; }

}

int main()

{

cout<<"Enter the number of processes:";

int n;cin>>n;

process arr[n];

cout<<"Enter the arrival time and burst time\n";

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

{cin>>arr[i].arrival\_time>>arr[i].burst\_time; arr[i].index=i; }

sort(arr,arr+n,cmp);

int ans[n];

memset(ans,0,sizeof(int)\*n);

int time=0;

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

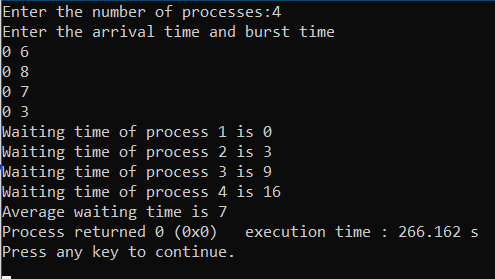
{ans[i]=time-arr[i].arrival\_time; time+=arr[i].burst\_time; }

int avg=0;

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

{cout<<"Waiting time of process "<<i+1<<" is "<<ans[i]<<endl; avg+=ans[i]; }

cout<<"Average waiting time is "<<avg/n;}

****

**PREEMPTIVE-SJF**

#include <bits/stdc++.h>

using namespace std;

struct Process {

int pid; // Process ID

int bt; // Burst Time

int art; // Arrival Time

};

void findWaitingTime(Process proc[], int n,int wt[])

{

int rt[n];

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

rt[i] = proc[i].bt;

int complete = 0, t = 0, minm = INT\_MAX;

int shortest = 0, finish\_time;

bool check = false;

while (complete != n) {

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

if ((proc[j].art <= t) &&(rt[j] < minm) && rt[j] > 0) {

minm = rt[j];shortest = j;check = true;}}

if (check == false) {

t++;

continue;

}

rt[shortest]--;

minm = rt[shortest];

if (minm == 0)minm = INT\_MAX;

if (rt[shortest] == 0) {

complete++;

check = false;

finish\_time = t + 1;

wt[shortest] = finish\_time -proc[shortest].bt -proc[shortest].art;

if (wt[shortest] < 0){wt[shortest] = 0;}

t++;}}

void findTurnAroundTime(Process proc[], int n,

int wt[], int tat[])

{

for (int i = 0; i < n; i++)tat[i] = proc[i].bt + wt[i];

}

void findavgTime(Process proc[], int n)

{

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

total\_tat = 0;

findWaitingTime(proc, n, wt);

findTurnAroundTime(proc, n, wt, tat);

cout << "Processes "<< " Burst time "<< " Waiting time "<< " Turn around time\n";

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

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

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

cout << " " << proc[i].pid << "\t\t"<< proc[i].bt << "\t\t " << wt[i]<< "\t\t " << tat[i] << endl;

}

cout << "\nAverage waiting time = "<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "<< (float)total\_tat / (float)n;

}

int main()

{ int n;

cout <<"enter the number of processes"<<endl;

cin>>n;

Process p [10];

cout<<"enter processes details"<<endl;

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

{

cout<<"enter process id"<<endl;

cin>>p[i].pid;

cout<<"enter the burst time"<<endl;

cin>>p[i].bt;

cout<<"enter the arrival time"<<endl;

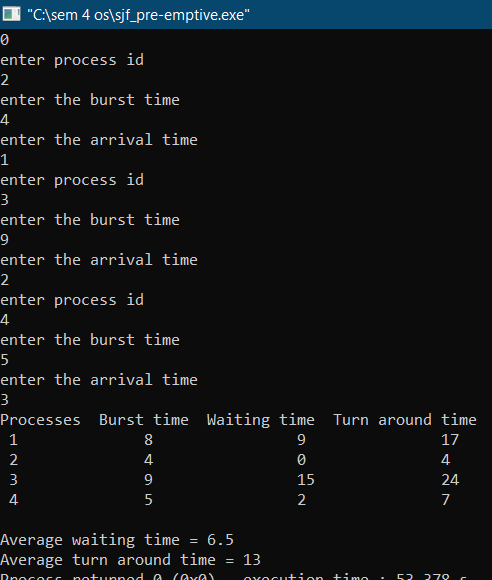
cin>>p[i].art;

}

findavgTime(p, n);

return 0;

}



**PRIORITY FCFS**

#include<bits/stdc++.h>

using namespace std;

struct Process

{

int pid; // Process ID

int bt; // CPU Burst time required

int priority; // Priority of this process

};

bool comparison(Process a, Process b)

{return (a.priority < b.priority);}

void findWaitingTime(Process proc[], int n,int wt[])

{

wt[0] = 0;

for (int i = 1; i < n ; i++ )wt[i] = proc[i-1].bt + wt[i-1] ;

}

void findTurnAroundTime( Process proc[], int n,int wt[], int tat[])

{for (int i = 0; i < n ; i++)tat[i] = proc[i].bt + wt[i];}

void findavgTime(Process proc[], int n)

{

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

findWaitingTime(proc, n, wt);

findTurnAroundTime(proc, n, wt, tat);

cout << "\nProcesses "<< " Burst time "<< " Waiting time " << " Turn around time\n";

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

{

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

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

cout << " " << proc[i].pid << "\t\t"

<< proc[i].bt << "\t " << wt[i]

<< "\t\t " << tat[i] <<endl;

}

cout << "\nAverage waiting time = "<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "<< (float)total\_tat / (float)n;

}

void priorityScheduling(Process proc[], int n)

{

sort(proc, proc + n, comparison);

cout<< "Order in which processes gets executed \n";

for (int i = 0 ; i < n; i++)cout << proc[i].pid <<" " ;

findavgTime(proc, n);

}

int main()

{

int n;

cout <<"enter the number of processes"<<endl;

cin>>n;

Process p [10];

cout<<"enter processes details"<<endl;

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

{

cout<<"enter process id:";

cin>>p[i].pid;

cout<<"enter the burst time:";

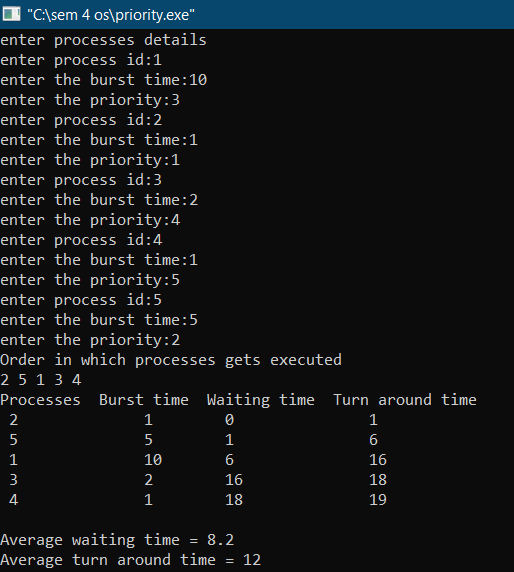
cin>>p[i].bt;

cout<<"enter the priority:";

cin>>p[i].priority;}

priorityScheduling(p, n);

return 0;}



**ROUND ROBIN**

#include<iostream>

using namespace std;

void findWaitingTime(int processes[], int n,int bt[], int wt[], int quantum)

{

int rem\_bt[n];

for (int i = 0 ; i < n ; i++)rem\_bt[i] = bt[i];

int t = 0;

while (1)

{

bool done = true;

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

{

if (rem\_bt[i] > 0)

{

done = false;

if (rem\_bt[i] > quantum)

{

t += quantum;

rem\_bt[i] -= quantum;

}

else

{

t = t + rem\_bt[i];

wt[i] = t - bt[i];

rem\_bt[i] = 0;

}

}

}

if (done == true)break;

}

}

void findTurnAroundTime(int processes[], int n,int bt[], int wt[], int tat[])

{for (int i = 0; i < n ; i++)tat[i] = bt[i] + wt[i];}

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

{

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

findWaitingTime(processes, n, bt, wt, quantum);

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

cout << "Processes "<< " Burst time "<< " Waiting time " << " Turn around time\n";

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

{

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

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

cout << " " << i+1 << "\t\t" << bt[i] <<"\t "<< wt[i] <<"\t\t " << tat[i] <<endl;

}

cout << "Average waiting time = "<< (float)total\_wt / (float)n;

cout << "\nAverage turn around time = "<< (float)total\_tat / (float)n;

}

int main()

{

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

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

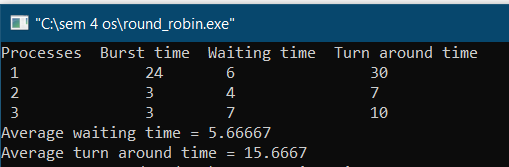
int burst\_time[] = {24,3,3};

int quantum = 4;

findavgTime(processes, n, burst\_time, quantum);

return 0;

}



**READER WRITER PROBLEM**

#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

sem\_t mutex,writeblock;

int data = 0,rcount = 0;

void \*reader(void \*arg)

{

int f;

f = ((int)arg);

sem\_wait(&mutex);

rcount = rcount + 1;

if(rcount==1)

sem\_wait(&writeblock);

sem\_post(&mutex);

printf("Data read by the reader%d is %d\n",f,data);

sem\_wait(&mutex);

rcount = rcount - 1;

if(rcount==0)

sem\_post(&writeblock);

sem\_post(&mutex);

}

void \*writer(void \*arg)

{

int f;

f = ((int) arg);

sem\_wait(&writeblock);

data++;

printf("Data writen by the writer%d is %d\n",f,data);

sem\_post(&writeblock);

}

int main()

{

int i,b;

pthread\_t rtid[5],wtid[5];

sem\_init(&mutex,0,1);

sem\_init(&writeblock,0,1);

for(i=0;i<=2;i++)

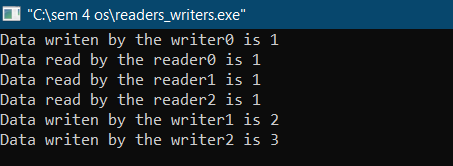
{pthread\_create(&wtid[i],NULL,writer,(void \*)i);pthread\_create(&rtid[i],NULL,reader,(void \*)i);}

for(i=0;i<=2;i++)

{pthread\_join(wtid[i],NULL);pthread\_join(rtid[i],NULL);}

return 0;

}



**DINING PHILOSOPHER PROBLEM**

#include<stdio.h>

#define n 4

int compltedPhilo = 0,i;

struct fork{

int taken;

}ForkAvil[n];

struct philosp{

int left;

int right;

}Philostatus[n];

void goForDinner(int philID){

if(Philostatus[philID].left==10 && Philostatus[philID].right==10)

printf("Philosopher %d completed his dinner\n",philID+1);

else if(Philostatus[philID].left==1 && Philostatus[philID].right==1){

printf("Philosopher %d completed his dinner\n",philID+1);

Philostatus[philID].left = Philostatus[philID].right = 10;

int otherFork = philID-1;

if(otherFork== -1)

otherFork=(n-1);

ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0;

printf("Philosopher %d released fork %d and fork %d\n",philID+1,philID+1,otherFork+1);

compltedPhilo++;

}

else if(Philostatus[philID].left==1 && Philostatus[philID].right==0){

if(philID==(n-1)){

if(ForkAvil[philID].taken==0){

ForkAvil[philID].taken = Philostatus[philID].right = 1;

printf("Fork %d taken by philosopher %d\n",philID+1,philID+1);

}else{

printf("Philosopher %d is waiting for fork %d\n",philID+1,philID+1);

}

}else{

int dupphilID = philID;

philID-=1;

if(philID== -1)

philID=(n-1);

if(ForkAvil[philID].taken == 0){

ForkAvil[philID].taken = Philostatus[dupphilID].right = 1;

printf("Fork %d taken by Philosopher %d\n",philID+1,dupphilID+1);

}else{

printf("Philosopher %d is waiting for Fork %d\n",dupphilID+1,philID+1);

}

}

}

else if(Philostatus[philID].left==0){

if(philID==(n-1)){

if(ForkAvil[philID-1].taken==0){

ForkAvil[philID-1].taken = Philostatus[philID].left = 1;

printf("Fork %d taken by philosopher %d\n",philID,philID+1);

}else{

printf("Philosopher %d is waiting for fork %d\n",philID+1,philID);

}

}else{

if(ForkAvil[philID].taken == 0){

ForkAvil[philID].taken = Philostatus[philID].left = 1;

printf("Fork %d taken by Philosopher %d\n",philID+1,philID+1);

}else{printf("Philosopher %d is waiting for Fork %d\n",philID+1,philID+1); }}}

}

int main(){

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

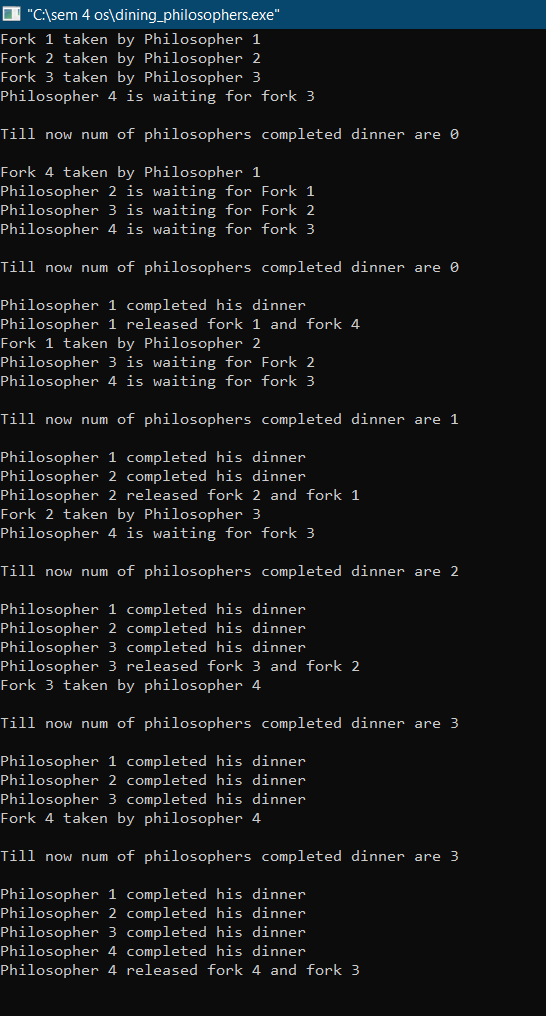
ForkAvil[i].taken=Philostatus[i].left=Philostatus[i].right=0;

while(compltedPhilo<n){ for(i=0;i<n;i++)goForDinner(i);

printf("\nTill now num of philosophers completed dinner are %d\n\n",compltedPhilo);}

return 0;

}



**BANKER ALGORITHM**

#include<iostream>

using namespace std;

bool banker(int available[3],int allocate[][3],int need[][3],int n)

{

bool finish[n];

for(int z=0;z<n;z++){finish[z]=false;}

int i=0;

bool again=true;

while(again==true)

{

again=false;

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

{

if(finish[i]==false && need[i][0]<=available[0] && need[i][1]<=available[1] && need[i][2]<=available[2])

{

again=true;

finish[i]=true;

available[0]+=allocate[i][0];available[1]+=allocate[i][1];available[2]+=allocate[i][2];

i=(i+1)%n;

break;

}

i=(i+1)%n;

}

}

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

{if(finish[z]==false){return false;} }

return true;

}

int main()

{

cout<<"Enter the instances of 3 resources:";

int available[3];

cin>>available[0]>>available[1]>>available[2];

cout<<"Enter the number of processes:";

int n;cin>>n;

cout<<"Enter max use of resources for each process:";

int maxr[n][3];

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

{cin>>maxr[i][0]>>maxr[i][1]>>maxr[i][2]; }

cout<<"Enter currently allocated instances of each process:";

int allocate[n][3];

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

{

cin>>allocate[i][0]>>allocate[i][1]>>allocate[i][2];

available[0]-=allocate[i][0];

available[1]-=allocate[i][1];

available[2]-=allocate[i][2];

}

int need[n][3];

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

{

need[i][0]=maxr[i][0]-allocate[i][0];

need[i][1]=maxr[i][1]-allocate[i][1];

need[i][2]=maxr[i][2]-allocate[i][2];

}

bool safe=banker(available,allocate,need,n);

if(safe==false)

{cout<<"Initial state is unsafe";return 0; }

cout<<"Enter the index of process and its request:";

int index,request[3];

cin>>index>>request[0]>>request[1]>>request[2];

if(need[index][0]<request[0] || need[index][1]<request[1] || need[index][2]<request[2])

{cout<<index<<" is requesting for instances more than it claimed so request denied";return 0; }

allocate[index][0]+=request[0];

allocate[index][1]+=request[1];

allocate[index][2]+=request[2];

need[index][0]-=request[0];

need[index][1]-=request[1];

need[index][2]-=request[2];

available[0]-=request[0];

available[1]-=request[1];

available[2]-=request[2];

safe=banker(available,allocate,need,n);

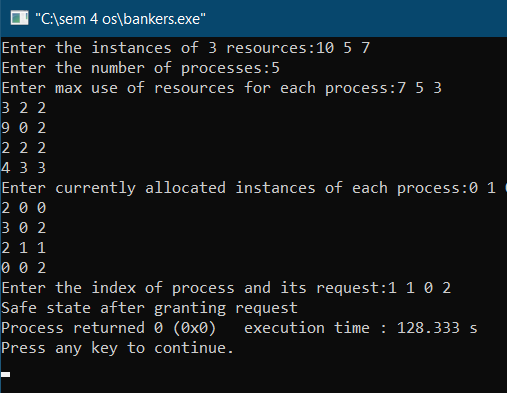
if(safe==true)

{cout<<"Safe state after granting request";}

else

{cout<<"Request cannot be granted";}

}



**PAGE REPLACEMENT FIFO**

#include<queue>

#include<iostream>

#include<map>

using namespace std;

void print(queue<int>q)

{

while(!q.empty())

{cout<<q.front()<<" ";q.pop();}

cout<<endl;

}

int fcfs(int n,int pages[])

{

int page\_fault=0;

queue<int>q;

map<int,bool>present;

for(int i=0;pages[i]!=-1;i++)

{

if(present[pages[i]]==false)

{

page\_fault++;

if(q.size()<n){q.push(pages[i]);present[pages[i]]=true;}

else

{

present[q.front()]=false;

q.pop();

q.push(pages[i]);

present[pages[i]]=true;

}

}

print(q);

}

return page\_fault;

}

int main()

{

int n;

cout<<"Enter the number of frames:";cin>>n;

int pages[10000];int npage=-1;

cout<<"Enter the pages and -1 to stop\n";

do{

npage++;

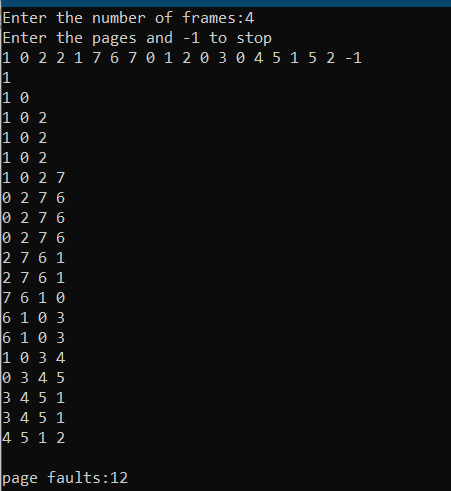
cin>>pages[npage];

}while(pages[npage]!=-1);

cout<<endl<<"page faults:"<<fcfs(n,pages);

return 0;

}



**PAGE REPLACEMENT LRU**

#include<queue>

#include<iostream>

#include<map>

using namespace std;

void print(map<int,int>q)

{

map<int,int>::iterator it=q.begin();

while(it!=q.end())

{cout<<it->first<<" ";it++;}

cout<<endl;

}

int LRU(int n,int pages[])

{

int page\_fault=0;

map<int,int>present;

for(int i=0;pages[i]!=-1;i++)

{

if(present.find(pages[i])==present.end())

{

page\_fault++;

if(present.size()<n)

{present[pages[i]]=i;}

else

{

int temp=present.begin()->first;

map<int,int>::iterator x=present.begin();

x++;

for(;x!=present.end();x++)

{if(x->second<present[temp]){temp=x->first;} }

present.erase(temp);

present[pages[i]]=i;

}

}

else

{present[pages[i]]=i; }

print(present);

}

return page\_fault;

}

int main()

{

int n;

cout<<"Enter the number of frames:";cin>>n;

int pages[10000];int npage=-1;

cout<<"Enter the pages and -1 to stop\n";

do{

npage++;

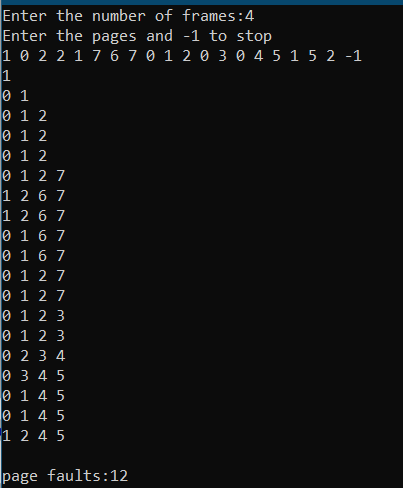
cin>>pages[npage];

}while(pages[npage]!=-1);

cout<<endl<<"page faults:"<<LRU(n,pages);

return 0;

}



**PAGE REPLACEMENT OPTIMAL**

#include<queue>

#include<iostream>

#include<map>

using namespace std;

void print(map<int,int>q)

{

map<int,int>::iterator it=q.begin();

while(it!=q.end())

{cout<<it->first<<" ";it++;}

cout<<endl;

}

int optimal(int n,int pages[])

{

int page\_fault=0;

map<int,int>present;

for(int i=0;pages[i]!=-1;i++)

{

if(present.find(pages[i])==present.end())

{

page\_fault++;

if(present.size()<n)

{

present[pages[i]]=INT\_MAX;

}

else

{

for(int j=i+1;pages[j]!=-1;j++)

{

if(present.find(pages[j])!=present.end())

{

if(present[pages[j]]==INT\_MAX)

{present[pages[j]]=j;}

}

}

int temp=present.begin()->first;int ans=present[temp];

present[temp]=INT\_MAX;

map<int,int>::iterator it=present.begin();it++;

for(;it!=present.end();it++)

{

if(it->second>ans){temp=it->first;ans=it->second;}

present[it->first]=INT\_MAX;

}

present.erase(temp);

present[pages[i]]=INT\_MAX;

}

}

print(present);

}

return page\_fault;

}

int main()

{

int n;

cout<<"Enter the number of frames:";cin>>n;

int pages[10000];int npage=-1;

cout<<"Enter the pages and -1 to stop\n";

do{

npage++;

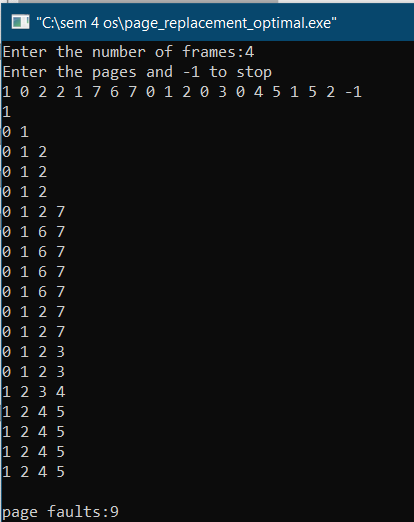
cin>>pages[npage];

}while(pages[npage]!=-1);

cout<<endl<<"page faults:"<<optimal(n,pages);

return 0;

}



**FILE MANAGEMENT**

#include<fstream>

#include<iostream>

#include<string>

using namespace std;

class stud

{

int age;string name;

public:

stud(int age,string name)

{

this->age=age;

this->name=name;

}

void print()

{cout<<" | "<<this->age<<" "<<this->name<<" | ";}};

int main()

{

cout<<"Binary files\n";

fstream a;

cout<<"Appending into the file name temp.dat\n";

a.open("temp.dat",ios::binary|ios::app);

cout<<"Enter the age and name of the student:";int age;string name;cin>>age>>name;

stud ex(age,name);

a.write((char\*)&ex,sizeof(ex));

cout<<"Closing file after appending\n";

a.close();

cout<<"Reading from file temp.dat\n\n";

a.open("temp.dat",ios::binary|ios::in);

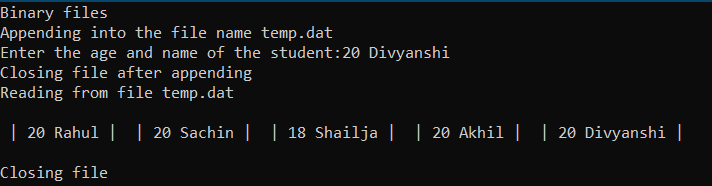
while(a.read((char\*)&ex,sizeof(ex)))

{ex.print();}

cout<<"\n\nClosing file";

a.close();

return 0;}



**DISK SCHEDULING FCFS**

#include<cmath>

#include<iostream>

using namespace std;

int main()

{

int cur,s,ans=0;

cout<<"Enter size of queue:";cin>>s;

int arr[s];

cout<<"Enter the requests values\n";

for(int i=0;i<s;i++){cin>>arr[i];}

cout<<"Enter the current pointer position:";cin>>cur;

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

{

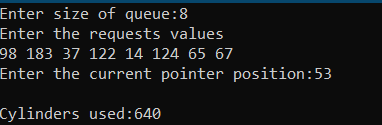
ans+=abs(arr[i]-cur);

cur=arr[i];

}

cout<<"\nCylinders used:"<<ans;

}



**DISK SCHEDULING SSTF**

#include<cmath>

#include<iostream>

using namespace std;

int main()

{

int cur,s,ans=0;

cout<<"Enter size of queue:";cin>>s;

int arr[s];

cout<<"Enter the requests values\n";

for(int i=0;i<s;i++){cin>>arr[i];}

cout<<"Enter the current pointer position:";cin>>cur;

bool next=true;

while(next)

{

next=false;int mini=-1;

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

{

if(arr[i]==-1){continue;}

if((mini==-1) || (abs(cur-arr[mini])>abs(cur-arr[i])))

{mini=i; next=true; }

}

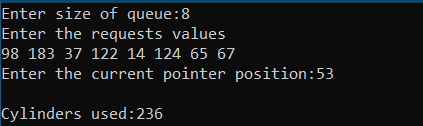
if(next==true)

{ans+=abs(cur-arr[mini]);cur=arr[mini];arr[mini]=-1;}

}

cout<<"\nCylinders used:"<<ans;

}



**DISK SCHEDULING SCAN**

#include<cmath>

#include<iostream>

#include<map>

using namespace std;

int main()

{

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*ASSUSMPTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nhead of disk is at 0 and end is at 199\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

int cur,s,ans=0;

cout<<"Enter size of queue:";cin>>s;

map<int,bool>arr;

cout<<"Enter the requests values\n";

for(int i=0;i<s;i++){int temp;cin>>temp;arr[temp]=true;}

cout<<"Enter the current pointer position:";cin>>cur;

while(arr.size()>0)

{

for(;cur>=0 && arr.size()>0;cur--)

{

if(arr.find(cur)!=arr.end()){arr.erase(cur);}

if(arr.size()==0){break;}

if(cur>0)

{ans++;}

}

cur=0;

if(arr.size()==0){break;}

for(;cur<=199 && arr.size()>0;cur++)

{

if(arr.find(cur)!=arr.end()){arr.erase(cur);}

if(arr.size()==0){break;}

if(cur<199)

{ans++;}

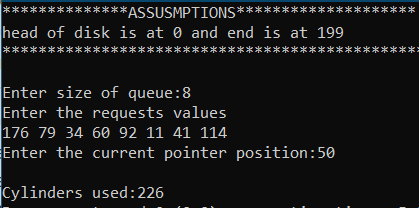
}

cur=199;

}

cout<<"\nCylinders used:"<<ans;

}



**DISK SCHEDULING C-SCAN**

#include<cmath>

#include<iostream>

#include<map>

using namespace std;

int main()

{

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*ASSUSMPTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nhead of disk is at 0 and end is at 199\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

int cur,s,ans=0;

cout<<"Enter size of queue:";cin>>s;

map<int,bool>arr;

cout<<"Enter the requests values\n";

for(int i=0;i<s;i++){int temp;cin>>temp;arr[temp]=true;}

cout<<"Enter the current pointer position:";cin>>cur;

while(arr.size()>0)

{

for(;cur<=199 && arr.size()>0;cur++)

{

if(arr.find(cur)!=arr.end()){arr.erase(cur);}

if(arr.size()==0){break;}

if(cur<199)

{ans++;}

}

if(arr.size()==0){break;}

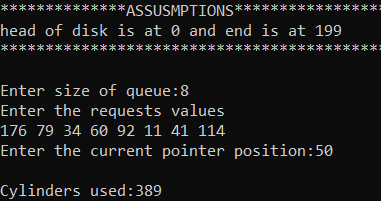
ans+=199;

cur=0;

}

cout<<"\nCylinders used:"<<ans;

}



**DISK SCHEDULING LOOK**

#include<cmath>

#include<iostream>

#include<map>

using namespace std;

int main()

{

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*ASSUSMPTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nhead of disk is at 0 and end is at 199\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

int cur,s,ans=0,big=INT\_MIN,small=INT\_MAX;

cout<<"Enter size of queue:";cin>>s;

map<int,bool>arr;

cout<<"Enter the requests values\n";

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

{

int temp;cin>>temp;arr[temp]=true;

if(temp>big){big=temp;}

if(temp<small){small=temp;}

}

cout<<"Enter the current pointer position:";cin>>cur;

while(arr.size()>0)

{

for(;cur<=big && arr.size()>0;cur++)

{

if(arr.find(cur)!=arr.end()){arr.erase(cur);}

if(arr.size()==0){break;}

if(cur<big)

{ans++;}

}

cur=big;

if(arr.size()==0){break;}

for(;cur>=small && arr.size()>0;cur--)

{

if(arr.find(cur)!=arr.end()){arr.erase(cur);}

if(arr.size()==0){break;}

if(cur>small)

{ans++;}

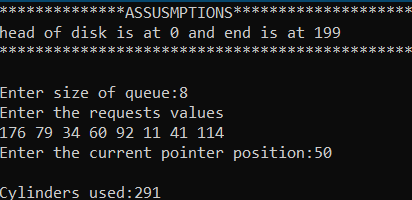
}

cur=small;

}

cout<<"\nCylinders used:"<<ans;

}



**DISK SCHEDULING C-LOOK**

#include<cmath>

#include<iostream>

#include<map>

using namespace std;

int main()

{

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*ASSUSMPTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\nhead of disk is at 0 and end is at 199\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

int cur,s,ans=0,big=INT\_MIN,small=INT\_MAX;

cout<<"Enter size of queue:";cin>>s;

map<int,bool>arr;

cout<<"Enter the requests values\n";

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

{

int temp;cin>>temp;arr[temp]=true;

if(temp>big){big=temp;}

if(temp<small){small=temp;}

}

cout<<"Enter the current pointer position:";cin>>cur;

while(arr.size()>0)

{

for(;cur<=big && arr.size()>0;cur++)

{

if(arr.find(cur)!=arr.end()){arr.erase(cur);}

if(arr.size()==0){break;}

if(cur<big)

{ans++;}

}

if(arr.size()==0){break;}

cur=small;

ans+=(big-small);

}

cout<<"\nCylinders used:"<<ans;

}

