OPERATING SYSTEMS

Practical Lab File

CECSC09 - 1

Submitted by :-

NAME: Harshit Gupta
ROLL NO: 2019UCO1580

INDEX

Program 1

Program to show creation of process

- Fork system call is used for creating a new process, which is called the child process, which runs concurrently with the process that makes the **fork() call.**
- The parent and the fork both execute the code following the line where **fork()** is called. Child process uses identical resources as allocated to the parent process.
- **Returns** negative, zero and positive values corresponding to the conditions that creation of the process was unsuccessful, return to the created child process and return to the parent or caller respectively.
- A new execution context is created for the child which is depicted by the following code

RECURSION FOR n Forks: F(n) = 2.F(n-1) which evaluates to F(n) = 2^n where F(n) denotes the number of processes created for n fork() calls.

CODE

- **Single**

```
#include<bits/stdc++.h>
#include <sys/types.h>
#include <unistd.h>
using namespace std;
int main(){
   ios::sync_with_stdio(0);
   cout<<"Executing the fork system call...\n";
   int x = 10;// created to show that variable value
              // changes for the child as a new execution
              // context is created for it
   // process ID
   pid_t ID = fork();
   if(ID==0)
       {cout<<"In child process, ID :"<<ID<<endl;
       cout<<"Value of x is :"<<++x<<endl;}
   else if(ID>0){
       cout<<"In parent process, ID :"<<ID<<endl;
```

```
        cout<<"Value of x is :"<<x<<endl;
    } return 0;}
```

- **Multiple**

```
#include<bits/stdc++.h>
#include <sys/types.h>
#include <unistd.h>
using namespace std;
int main(){
    ios::sync_with_stdio(0);
    cout<<"Executing the fork system call...\n";
    int x = 10;// created to show that variable value
               // change for the child as a new execution
               // context is created for it

    pid_t ID1 = fork();
    pid_t ID2 = fork();
    if(ID1==0 && ID2>0)
        {cout<<"In child process 1, ID :"<<ID1<<endl;
        cout<<"Value of x is :"<<++x<<endl;}
    else if(ID2==0 && ID1>0){cout<<"In child process 2, ID :"<<ID2<<endl;
        cout<<"Value of x is :"<<--x<<endl;}
    else if(ID1>0 && ID2>0){
        cout<<"In parent process, ID1:"<<ID1<<" ID2:"<<ID2<<endl;
        cout<<"Value of x is :"<<x<<endl;
    }
    else{
        cout<<"In child 1's child , both IDs are ID1:"<<ID1<<"
ID2:"<<ID2<<endl;
    }
    return 0;
}
```

SCREENSHOT 1

```
harshit@harshit-Aspire-A315-55G:~/Documents/COLLEGE/O
S Lab Work/Create process$ ./fork
Executing the fork system call...
In parent process, ID :5126
Value of x is :10
Executing the fork system call...
In child process, ID :0
Value of x is :11
harshit@harshit-Aspire-A315-55G:~/Documents/COLLEGE/O
S Lab Work/Create process$
```

SCREENSHOT 2

```
harshit@harshit-Aspire-A315-55G:~/Documents/COLLEGE/O
S Lab Work/Create process$ ./multiple
Executing the fork system call...
In parent process, ID1:5880 ID2:5881
Value of x is :10
Executing the fork system call...
In child process 2, ID :0
Value of x is :9
Executing the fork system call...
In child process 1, ID :0
Value of x is :11
Executing the fork system call...
In child 1's child , both IDs are ID1:0 ID2:0
harshit@harshit-Aspire-A315-55G:~/Documents/COLLEGE/O
S Lab Work/Create process$
```

Program 2

Program to show the concept of multithreading

- Threads are the basic unit of computation for a CPU executing a program
- A Process can be thought of as analogous to a building construction being done and Threads can be thought of as the numerous workers executing different tasks like painting, electricity fittings, laying bricks etc.
- Just the difference in a computer system is that the process actually gets converted to viz. A web browser being opened and threads are the workers which execute loading of graphics, fetching of text content, processing server requests, etc.
- **PThreads** or POSIX threads is an implementation of the threading features in different operating systems and its use in Windows operating system is depicted by the following program
- The objective of this experiment is to copy the contents of one file into another through multiple threads where it creates 3 different threads for **input names, copying** and **termination** of the program
- Note that **thread2** is actually dependent on **thread1** for the data values and thus can't be executed parallely with it

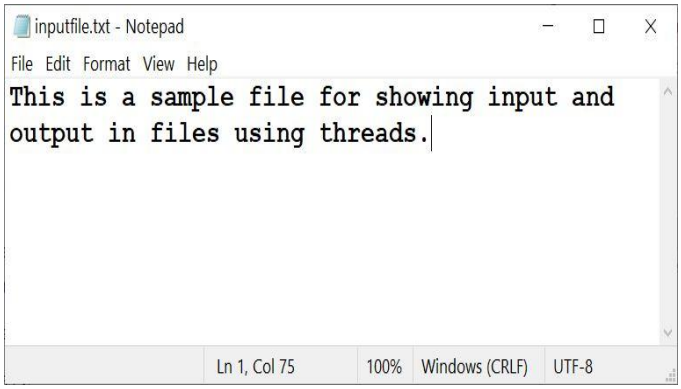
CODE - parallel execution achieved in the context of **pre opening the output file in a different thread.**

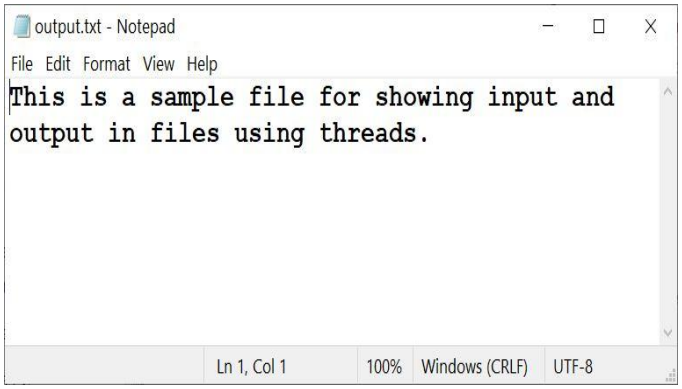
```
#include<bits/stdc++.h>
#include<thread>
#include<chrono>
#include<fstream>
using namespace std;
char source[20];
char destination[20];
FILE *fs, *ft;
int res;
string output;
int main(){
```

```
// making three functions for
// 1. input
// 2. copying
// 3. termination message
// for input names
output="";
res = 0;
auto f1 = [](){
cout<<"\nEnter the Name of Source File: ";
cin>>source;
fs = fopen(source,"r"); // read pointer
if(fs==NULL){
    cout<<"\nCould not open source file!";
    res = -1;
}
else
    cout<<"***INPUT OPENED SUCCESSFULLY***\n";
};
auto f2 = [](){
     ft = fopen("output.txt","w"); // write pointer
     cout<<"***OUTPUT OPENED SUCCESSFULLY***\n";
     return;
};
// for termination
auto f3 = [](){
    char ch = fgetc(fs); // get character
    while(ch != EOF)
    {
        fputc(ch, ft); // put character
        ch = fgetc(fs);
    }
    cout<<"\nFile copied successfully.";
    fclose(fs);
    fclose(ft);
    cout<<endl;
};
thread t2(f2);
thread::id t2_id = t2.get_id();
thread t1(f1);
```

```
    thread::id t1_id = t1.get_id();
    t1.join();
    t2.join();
    cout<<"\nInput Thread with id "<<t1_id<<" completed";
    cout<<"\nOutput Thread with id "<<t2_id<<" completed";
    if(res!=-1)
    { thread t3(f3);
    thread::id t3_id = t3.get_id();
    t3.join();
    cout<<"\nCopy & Termination Thread with id "<<t3_id<<" completed";
    }
    else {cout<<"\nCould not open file name "<<source<<endl;
         cout<<"\nIncomplete execution.";}
    return 0;
}
```

FILES





TERMINAL : contains successful and unsuccessful execution

The following screenshot contains the output of the file read-write being executed as a multi-threaded process in the program mentioned above.

```
PS D:\IV Semester\OS\LAB\Lab2> g++ -std=c++11 lab2.cpp -pthread -o lab2
PS D:\IV Semester\OS\LAB\Lab2> ./lab2
***OUTPUT OPENED SUCCESSFULLY***

Enter the Name of Source File: inputfile.txt
***INPUT OPENED SUCCESSFULLY***

Input Thread with id 3 completed
Output Thread with id 2 completed
File copied successfully.

Copy & Termination Thread with id 4 completed
PS D:\IV Semester\OS\LAB\Lab2> ./lab2
***OUTPUT OPENED SUCCESSFULLY***

Enter the Name of Source File: input.txt

Could not open source file!
Input Thread with id 3 completed
Output Thread with id 2 completed
Could not open file name input.txt

Incomplete execution.
PS D:\IV Semester\OS\LAB\Lab2>
```

Program 3

To show the concept of First Come First Serve job scheduling

- FCFS algorithm is one of the simplest job scheduling algorithms and schedules the jobs on the CPU as and when they come inside the ready queue of the system

CODE

```
#include<bits/stdc++.h>
#define all(x) x.begin(),x.end()
using namespace std;
int n,bur,arr,ind;

struct Process{
    int pid;
    int arrival_time;
    int burst_time;
};
vector< Process > process;
map <int,int> waiting, turn_around;

bool my_sorter(Process const &a, Process const &b){
    return a.arrival_time < b.arrival_time;
}


float get_awt(){
    int time = 0;
    float awt = 0.0;
    for(int i=0; i<n; i++){
        ind = process[i].pid;
        arr = process[i].arrival_time;
        bur = process[i].burst_time;

        if(i==0)
            waiting[ind] = 0;
        else
```

```
            waiting[ind] = max(time - arr,0);

        time = max(time + bur, arr + bur);
    }

    for(auto k:waiting)
        awt+= k.second;

    return awt/n;
}

float get_tat(){
    float tat = 0.0;
    for(int i=0;i<n;i++){
       ind = process[i].pid;
       bur = process[i].burst_time;
       turn_around[ind] = waiting[ind] + bur;
    }

    for(auto k:turn_around)
        tat+= k.second;

    return tat/n;
}
int main(){

    cout<<"Enter the number of processes :";
    cin>>n;
    int arrival,burst;
    Process P;
    for(int i=0;i<n;i++){
        cout<<"Arrival time "<<i+1<<":";
        cin>>arrival;
        cout<<"Burst time "<<i+1<<":";
        cin>>burst;
        P.pid = i;
        P.arrival_time = arrival;
        P.burst_time = burst;
        process.push_back(P);
```

```
    }


    sort(all(process),my_sorter);
    cout<<"PROCESS\t\t ARRIVAL TIME\t\tBURST TIME\n";

    for(auto k: process)
        cout<<"   "<<k.pid<<"\t\t
"<<k.arrival_time<<"\t\t\t"<<k.burst_time<<endl;


    float awt = get_awt();
    float tat = get_tat();

    cout<<"AVERAGE WAITING TIME FOR FIFO  :"<<awt;
    cout<<"\nAVERAGE TURN AROUND TIME FOR FIFO :"<<tat;
    return 0;

}
```

TERMINAL

```
PS D:\IV Semester\OS\LAB\lab3> g++ fifo.cpp
PS D:\IV Semester\OS\LAB\lab3> ./a
Enter the number of processes :4
Arrival time 1:2
Burst time 1:8
Arrival time 2:0
Burst time 2:4
Arrival time 3:3
Burst time 3:9
Arrival time 4:1
Burst time 4:2
PROCESS          ARRIVAL TIME          BURST TIME
  1              0                     4
  3              1                     2
  0              2                     8
  2              3                     9
AVERAGE WAITING TIME FOR FIFO  :4.5
AVERAGE TURN AROUND TIME FOR FIFO :10.25
```

Program 4

To show the concept of Shortest Job First job scheduling

- This algorithm is considered to be one of the optimal algorithm for job scheduling
- It takes the smallest available job in the ready queue as the next job to be executed on the CPU

CODE

```
#include<bits/stdc++.h>
#define all(x) x.begin(),x.end()
using namespace std;
int n,bur,ind,arrived;

struct Process{
    int pid;
    int arrival_time;
    int burst_time;
};
map <int,int> waiting, turn_around;
vector<Process> process;

bool my_sorter(Process const &a, Process const &b){
    if(a.arrival_time != b.arrival_time )
        return a.arrival_time < b.arrival_time;
    else{
        return a.burst_time < b.burst_time ;
    }
}
// to get the current process
int get_curr_process(int time,set<int> done){
    if(time == 0)
        return 0;
    else{
        int min_arr,best_ind;
        bool found1 = false;
```

```
        bool found2 = false;
        int min_bur = 1e6;
        for(int i=0;i<n;i++){

            // get the best left over process
            if(done.find(i) == done.end()){

                arrived = process[i].arrival_time;
                bur = process[i].burst_time;

                // min_arrival process
                if(found1 == false){
                    min_arr = i;
                    found1 = true;
                }

                // if process present
                if(arrived <= time ){
                    if(bur <= min_bur){
                        min_bur = bur;
                        best_ind = i;
                        found2 = true;
                    }
                }

            }
        }
        // if all processes have greater arrivals
        if(found2 == false)
            return min_arr;

        // return shortest job within time
        return best_ind;
    }
}

float get_AWT(){
    // first sort the job according to arrival time
    int time = 0;
```

```
    float awt = 0.0;

    set<int> done;
    while(done.size() != n){

        int curr = get_curr_process(time,done); // get the process

        // then select the job with min arrival and min burst time
        // such that the arrival time of the job is < current time stamp
        ind = process[curr].pid;
        bur = process[curr].burst_time;
        arrived = process[curr].arrival_time;

        // now this process's waiting time is
        waiting[ind] = max(time - arrived,0);

        // increment the time stamp
        time  = max(time + bur, arrived + bur);

        // process is done
        done.insert(curr);
    }


    for(auto k:waiting)
        awt+= k.second;
    return awt/n;
    // execute it and then again do the same till you have no processes
left
}

float get_TAT(){
    float tat = 0.0;

    for (int i=0;i<n;i++)
        tat+= process[i].burst_time + waiting[i];

    return tat/n;
}
```

```
int main(){
    ios::sync_with_stdio(0);


    cout<<"Enter the number of processes :";
    cin>>n;
    int arrival,burst;
    Process P;

    for(int i=0;i<n;i++){
        cout<<"Arrival time "<<i+1<<":";
        cin>>arrival;
        cout<<"Burst time "<<i+1<<":";
        cin>>burst;
        P.pid = i;
        P.arrival_time = arrival;
        P.burst_time = burst;
        process.push_back(P);
    }

    sort(all(process),my_sorter);

    cout<<"PROCESS\t\t ARRIVAL TIME\t\tBURST TIME\n";

    for(auto k: process)
        cout<<"  "<<k.pid<<"\t\t
"<<k.arrival_time<<"\t\t\t"<<k.burst_time<<endl;

    float awt = get_AWT();
    float tat = get_TAT();
    cout<<"AVERAGE WAITING TIME FOR SHORTEST JOB FIRST :"<<awt;
    cout<<"\nAVERAGE TURN AROUND TIME FOR SHORTEST JOB FIRST :"<<tat;


    return 0;
}
```

TERMINAL

```
PS D:\IV Semester\OS\LAB\lab3> g++ shortest_job.cpp
PS D:\IV Semester\OS\LAB\lab3> ./a
Enter the number of processes :4
Arrival time 1:2
Burst time 1:3
Arrival time 2:0
Burst time 2:4
Arrival time 3:4
Burst time 3:2
Arrival time 4:5
Burst time 4:4
PROCESS          ARRIVAL TIME          BURST TIME
  1              0                     4
  0              2                     3
  2              4                     2
  3              5                     4
AVERAGE WAITING TIME FOR SHORTEST JOB FIRST :2
AVERAGE TURN AROUND TIME FOR SHORTEST JOB FIRST :5.25
```

Program 5

To show the concept of Round Robin job scheduling

- Round robin scheduling is derived from the FCFS job scheduling but has a change in the times allotted to each job
- In RR scheduling, there is a fixed **time quantum** allotted to each job irrespective of the size of job and this algorithm executes preemptively.

CODE

```
#include<bits/stdc++.h>
#define all(x) x.begin(),x.end()
using namespace std;
int n,bur;

struct Process{
    int pid;
    int arrival_time;
    int burst_time;
};

bool my_sorter(Process const &a, Process const &b){
    return a.arrival_time < b.arrival_time;
}
map <int,int> waiting, turn_around;

vector<Process> process;
float get_awt(int time_slice){
    // variables
    int time = 0;
    float awt = 0;
    int left = n;
    int remaining[n] = {0};

    // copy burst time
    for(int i=0;i<n;i++){
        remaining[i] = process[i].burst_time;
    }
```

```
    // iterate till atleast one process left
    while(left > 0){

        for(int i=0;i<n;i++){
            bur = remaining[i] ;

            if(bur == 0) continue;

            if(bur > time_slice){
                time+= time_slice;
                remaining[i] -= time_slice;
            }
            else{
                // total time counter
                time += remaining[i];
                waiting[i] = time - process[i].burst_time;// because this
is the time for which process waited
                remaining[i] = 0;
                left--;
            }
        }
    }

    for(auto k:waiting)
        awt+= k.second;

    return awt/n;
}
float get_tat(){
    float tat = 0;
    for(int i=0;i<n;i++){
        turn_around[i] = waiting[i] + process[i].burst_time;
    }

    for(auto k:turn_around)
        tat+= k.second;

    return tat/n;
```

```
}
int main(){

    cout<<"Enter the number of processes :";
    cin>>n;
    int arrival,burst;
    Process P;
    for(int i=0;i<n;i++){
        cout<<"Arrival time "<<i+1<<":";
        cin>>arrival;
        cout<<"Burst time "<<i+1<<":";
        cin>>burst;
        P.pid = i;
        P.arrival_time = arrival;
        P.burst_time = burst;
        process.push_back(P);
    }

    int time_slice;
    cout<<"Enter the time slice :";
    cin>>time_slice;

    sort(all(process),my_sorter);
    cout<<"PROCESS\t\t ARRIVAL TIME\t\tBURST TIME\n";

    for(auto k: process)
        cout<<"  "<<k.pid<<"\t\t
"<<k.arrival_time<<"\t\t\t"<<k.burst_time<<endl;


    float awt = get_awt(time_slice);
    float tat = get_tat();
    cout<<"AVERAGE WAITING TIME FOR ROUND ROBIN :"<<awt;
    cout<<"\nAVERAGE TURN AROUND TIME FOR ROUND ROBIN :"<<tat;
    return 0;

}
```

TERMINAL

```
PS D:\IV Semester\OS\LAB\lab3> g++ round_robin.cpp
PS D:\IV Semester\OS\LAB\lab3> ./a
Enter the number of processes :3
Arrival time 1:0
Burst time 1:10
Arrival time 2:0
Burst time 2:5
Arrival time 3:0
Burst time 3:8
Enter the time slice :2
PROCESS          ARRIVAL TIME          BURST TIME
  0              0                     10
  1              0                     5
  2              0                     8
AVERAGE WAITING TIME FOR ROUND ROBIN :12
AVERAGE TURN AROUND TIME FOR ROUND ROBIN :19.6667
```

Program 6

To show the concept of priority job scheduling

- This algorithm is used for the scheduling of jobs with respect to the priorities of execution of the jobs
- These priorities may be pre-decided or maybe determined dynamically like in the Completely Fair Scheduler of Linux kernel.
- Here the basic priority scheduling algorithm is implemented, assuming **higher priority** for lower value

CODE

```
#include<bits/stdc++.h>
#define all(x) x.begin(),x.end()
using namespace std;
int n,burst,priority,arrival,ind;

struct Process{
    int pid;
    int burst_time;
    int priority;
};

bool my_sorter(Process const &a, Process const &b){
    return a.priority < b.priority;
}
map <int,int> waiting, turn_around;

vector<Process> process;
float get_awt(){
    // variables
    int time = 0;
    float awt = 0;
    for(int i=0;i<n;i++){
        ind = process[i].pid;
        priority = process[i].priority;
        burst = process[i].burst_time;
        // waiting time for process
        waiting[ind] = time;
```

```
        time+= burst;
    }

    for(auto k:waiting)
        awt+= k.second;
    return awt/n;
}
float get_tat(){
    float tat = 0;

    for(int i=0;i<n;i++){
        ind = process[i].pid;
        burst = process[i].burst_time;

        turn_around[ind] = waiting[ind] + burst;
    }
    for(auto k:turn_around)
        tat+= k.second;
    return tat/n;
}
int main(){

    cout<<"Enter the number of processes :";
    cin>>n;
    Process P;
    for(int i=0;i<n;i++){

        cout<<"Burst time "<<i+1<<":";
        cin>>burst;
        cout<<"Priority "<<i+1<<":";
        cin>>priority;
        P.pid = i;
        P.priority = priority;
        P.burst_time = burst;
        process.push_back(P);
    }

    // ASSUMPTION - lower absolute value of priority is having higher
actual priority
```

```
    // priority 1 -> highest
    // priority n -> lowest
    cout<<"ORDER OF PROCESSES IN PRIOIRITY SCHEDULING:\n";
    sort(all(process),my_sorter);
    cout<<"PROCESS\t\t PRIORITY\t\tBURST TIME\n";

    for(auto k: process)
        cout<<"  "<<k.pid<<"\t\t
"<<k.priority<<"\t\t\t"<<k.burst_time<<endl;

    float awt = get_awt();
    float tat = get_tat();
    cout<<"AVERAGE WAITING TIME FOR PRIORITY SCHEDULING:"<<awt;
    cout<<"\nAVERAGE TURN AROUND TIME FOR PRIORITY SCHEDULING :"<<tat;
    return 0;

}
```

TERMINAL

```
PS D:\IV Semester\OS\LAB\lab3> g++ priority.cpp
PS D:\IV Semester\OS\LAB\lab3> ./a
Enter the number of processes :5
Burst time 1:4
Priority 1:1
Burst time 2:8
Priority 2:5
Burst time 3:11
Priority 3:3
Burst time 4:9
Priority 4:4
Burst time 5:5
Priority 5:2
ORDER OF PROCESSES IN PRIOIRITY SCHEDULING:
PROCESS          PRIORITY                BURST TIME
  0              1                       4
  4              2                       5
  2              3                       11
  3              4                       9
  1              5                       8
AVERAGE WAITING TIME FOR PRIORITY SCHEDULING:12.4
AVERAGE TURN AROUND TIME FOR PRIORITY SCHEDULING :19.8
```

Program 7

To show the concept of Multilevel feedback queue scheduling

CODE

```
#include<bits/stdc++.h>
using namespace std;
struct process
{
    char name;
    int AT,BT,WT,TAT,RT,CT;
}
Q1[10],Q2[10],Q3[10];/*Three queues*/

int n;
void sortByArrival()
{
struct process temp;
int i,j;
for(i=0;i<n;i++)
    {
        for(j=i+1;j<n;j++)
            {
                if(Q1[i].AT>Q1[j].AT)
                    {
                            temp=Q1[i];
                            Q1[i]=Q1[j];
                            Q1[j]=temp;
                    }
            }
    }
}

int main()
{
     int i,j,k=0,r=0,time=0,tq1=5,tq2=8,flag=0;
     char c;
```

```
     cout<<"Enter no of processes:";
     cin>>n;
     for(i=0,c='A';i<n;i++,c++)
     {

         Q1[i].name=c;
         cout<<"Arrival time and burst time of process
"<<Q1[i].name<<endl;
         Q1[i].AT = rand()%10 + 1;
         Q1[i].BT = rand()%10 + 1;
         cout<<"Arrival :"<<Q1[i].AT<<" Burst :"<<Q1[i].BT<<endl;
         Q1[i].RT=Q1[i].BT;/*save burst time in remaining time for each
process*/

    }
    sortByArrival();
    time=Q1[0].AT;
    cout<<"\nProcess in first queue following RR with qt=5";
    cout<<"\nProcess\t\tRT\t\tWT\t\tTAT\t\t";
    for(i=0;i<n;i++)
    {

    if(Q1[i].RT<=tq1)
    {

        time+=Q1[i].RT;/*from arrival time of first process to completion
of this process*/
        Q1[i].RT=0;
        Q1[i].WT=time-Q1[i].AT-Q1[i].BT;/*amount of time process has been
waiting in the first queue*/
        Q1[i].TAT=time-Q1[i].AT;/*amount of time to execute the process*/

printf("\n%c\t\t%d\t\t%d\t\t%d",Q1[i].name,Q1[i].BT,Q1[i].WT,Q1[i].TAT);

    }
    else/*process moves to queue 2 with qt=8*/
    {
        Q2[k].WT=time;
        time+=tq1;
```

```
        Q1[i].RT-=tq1;
        Q2[k].BT=Q1[i].RT;
        Q2[k].RT=Q2[k].BT;
        Q2[k].name=Q1[i].name;
        k=k+1;
        flag=1;
    }
    }
    if(flag==1)
    {
        cout<<"\nProcess in second queue following RR with qt=8";
        cout<<"\nProcess\t\tRT\t\tWT\t\tTAT\t\t";
    }for(i=0;i<k;i++)
    {
        if(Q2[i].RT<=tq2)
        {
        time+=Q2[i].RT;/*from arrival time of first process +BT of this
process*/
        Q2[i].RT=0;
        Q2[i].WT=time-tq1-Q2[i].BT;/*amount of time process has been
waiting in the ready queue*/
        Q2[i].TAT=time-Q2[i].AT;/*amount of time to execute the process*/
printf("\n%c\t\t%d\t\t%d\t\t%d",Q2[i].name,Q2[i].BT,Q2[i].WT,Q2[i].TAT);
        }
        else/*process moves to queue 3 with FCFS*/
        {
        Q3[r].AT=time;
        time+=tq2;
        Q2[i].RT-=tq2;
        Q3[r].BT=Q2[i].RT;
        Q3[r].RT=Q3[r].BT;
        Q3[r].name=Q2[i].name;
        r=r+1;
        flag=2;
        }
    }

    {if(flag==2)
    cout<<"\nProcess in third queue following FCFS ";
```

```
    }
    for(i=0;i<r;i++)
    {
        if(i==0)
                Q3[i].CT=Q3[i].BT+time-tq1-tq2;
            else
                Q3[i].CT=Q3[i-1].CT+Q3[i].BT;
    }
    for(i=0;i<r;i++)
        {
            Q3[i].TAT=Q3[i].CT;
            Q3[i].WT=Q3[i].TAT-Q3[i].BT;

printf("\n%c\t\t%d\t\t%d\t\t%d\t\t",Q3[i].name,Q3[i].BT,Q3[i].WT,Q3[i].TAT
);
        }
    return 0;
}
```

SCREENSHOT

```
PS D:\IV Semester\OS\LAB\Lab3\codes> g++ multi_level.cpp
PS D:\IV Semester\OS\LAB\Lab3\codes> ./a
Enter no of processes:6
Arrival time and burst time of process A
Arrival :2 Burst :8
Arrival time and burst time of process B
Arrival :5 Burst :1
Arrival time and burst time of process C
Arrival :10 Burst :5
Arrival time and burst time of process D
Arrival :9 Burst :9
Arrival time and burst time of process E
Arrival :3 Burst :5
Arrival time and burst time of process F
Arrival :6 Burst :6

Process in first queue following RR with qt=5
Process         RT              WT              TAT
E               5               4               9
B               1               7               8
C               5               13              18
Process in second queue following RR with qt=8
Process         RT              WT              TAT
A               3               23              31
F               1               26              32
D               4               27              36
```

Program 8

To show the working of deadlock avoidance and deadlock detection through Banker's algorithm

- Banker's algorithm is the algorithm used for the avoidance of deadlocks in system
- It is used for avoiding deadlocks in system which contains multiple instances of single resource types
- It has a worst case run time complexity of $O(mn^2)$ where m is the number of different resource types available in the system and n is the number of processes that the system contains at the present instant.

SYSTEM STATE

- There are 5 processes in the system and 3 resources in the system with 10, 5 and 7 instances.

	Allocated	*Max Required*	*Available*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

CODE

```
#include<bits/stdc++.h>
using namespace std;
typedef vector<int> vi;
typedef vector<vector<int> > vvi;

// declare variables
```

```
int n,m;
vvi max_p(100,vi(100));
vvi allocated(100,vi(100));
vvi needs(100,vi(100));
vector <int> max_resources(100);
vector <int> available(100);

// returns the safe sequence, if present...
list<int> get_safe_sequence(){

     list<int> result; // first element denotes whether we found or not

    // start with safety algorithm

    // generate finish array
    bool finish[n] = {false};
    int count = n; // all unfinished

    // generate work array
    vector<int> work;
    work.reserve(m);
    for(int i=0;i<m;i++)
        work.push_back(available[i]);

    int less;
    bool found;
    while(count > 0){
        found = false;
        for(int i=0;i<n;i++){
            if(finish[i] == false){
                // found unfinished process
                less = 0;

                // compare Needi and work
                for(int j=0;j<m;j++){
                    if(needs[i][j] <= work[j]) less++;
                }
                // if all of them are lesser
                if(less == m){
```

```
                    // reclaim resources
                    found = true;
                    for(int j=0;j<m;j++)
                        work[j]+= allocated[i][j];

                    //process is finished
                    result.push_back(i);
                    finish[i] = true;
                    count--;
                }

                else {
                    continue;
                }
            }
        }
        if(found == false)
            break;
    }

    // if result is consisting of all processes
    if(result.size() == n){
        result.push_front(1);
        return result;
    }

    // if result does not contain all processes
    else{
        result.push_front(-1);
        return result;
    }

}

int main(){

    ios::sync_with_stdio(0);
    cout<<"Simulating Banker's algorithm...\n";
    cout<<"Enter the total number of processes in the system :";
```

```
    cin>>n;
    cout<<"Enter the total number of resources in the system :";
    cin>>m;


    int rj;

    cout<<"Enter the MAXIMUM availability of the resources :\n";
    for(int i=0;i<m;i++){
        cout<<"R"<<i<<":";
        cin>>rj;
        max_resources[i] = rj;
    }
    cout<<"\nEnter the CURRENT availability of the resources :\n";
    for(int i=0;i<m;i++){
        cout<<"R"<<i<<":";
        cin>>rj;
        available[i] = rj;
    }

    cout<<"PROCESSES' INFORMATION \n\n";

    cout<<"Enter the MAXIMUM possible resources for processes\n";
    for(int i=0;i<n;i++){
        cout<<"For process P"<<i<<":\n";
        for(int j=0;j<m;j++){
            cout<<"R"<<j<<":";
            cin>>rj;
            max_p[i][j] = rj;
        }
    }

    cout<<"Enter the CURRENT allocated resources";
    for(int i=0;i<n;i++){
        cout<<"For process P"<<i<<":\n";
        for(int j=0;j<m;j++){
            cout<<"R"<<j<<":";
            cin>>rj;
            allocated[i][j] = rj;
```

```
        }
    }

    // calculate the need matrix..

    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            needs[i][j] = max_p[i][j] - allocated[i][j];
        }
    }

    // show processes in system...
    cout<<"PROCESSES \t ALLOCATED \t MAXIMUM \t AVAILABLE RESOURCES \n";
    for(int i=0;i<n;i++){
        cout<<"     P"<<i<<"\t\t ";
        for(int j=0;j<m;j++){
            cout<<allocated[i][j]<<" ";
        }
        cout<<"\t\t";
        for(int j=0;j<m;j++){
            cout<<max_p[i][j]<<" ";
        }
        cout<<"\t\t";
        if(i==0)
        {for(int j=0;j<m;j++){
            cout<<available[j]<<" ";
        }}

        cout<<endl;
    }

    // now calculate a safe sequence
    list<int> safe = get_safe_sequence();

    // first element would be 1 or -1

    // 1 means -> got a safe sequence
    // -1 means -> do not have a safe sequence in this state
    auto it = safe.begin();
```

```
    if(*it == 1){
        cout<<"Safe sequence exists!\n";
        cout<<"SAFE SEQUENCE : ";

        // move to second element
        it++;

        while(it!=safe.end())
            {cout<<"P"<<*it<<" ";
            it++;}
        cout<<endl;
    }
    else
        cout<<"Sorry, safe sequence does not exist.\n";
    return 0;}
```

SCREENSHOTS

```
PS D:\IV Semester\OS\LAB\Lab4> ./a
Simulating Banker's algorithm...
Enter the total number of processes in the system :5
Enter the total number of resources in the system :3
Enter the MAXIMUM availability of the resources :
R0:10
R1:5
R2:7

Enter the CURRENT availability of the resources :
R0:3
R1:3
R2:2
PROCESSES' INFORMATION

Enter the MAXIMUM possible resources for processes
For process P0:
R0:7
R1:5
R2:3
For process P1:
R0:3
R1:2
R2:2
```

```
For process P2:
R0:9
R1:0
R2:2
For process P3:
R0:2
R1:2
R2:2
For process P4:
R0:4
R1:3
R2:3
Enter the CURRENT allocated resourcesFor process P0:
R0:0
R1:1
R2:0
For process P1:
R0:2
R1:0
R2:0
For process P2:
R0:3
R1:0
R2:2
```

```
For process P3:
R0:2
R1:1
R2:1
For process P4:
R0:0
R1:0
R2:2
PROCESSES        ALLOCATED       MAXIMUM         AVAILABLE RESOURCES
     P0          0 1 0          7 5 3           3 3 2
     P1          2 0 0          3 2 2
     P2          3 0 2          9 0 2
     P3          2 1 1          2 2 2
     P4          0 0 2          4 3 3
Safe sequence exists!
SAFE SEQUENCE : P1 P3 P4 P0 P2
```

Program 9

To show the working of the Best Fit memory allocation in contiguous memory.

- The following program highlights how the memory is allocated inside a system when the allocation strategy is best-fit for contiguous memory
- This is one of the best contiguous memory allocation strategies and looks to minimize the internal fragmentation caused due to the memory space allocated.

CODE

```
#include<bits/stdc++.h>
#define all(c) c.begin(),c.end()
using namespace std;
int main(){
    ios::sync_with_stdio(0);

    cout<<"Program to simulate BEST FIT CONTIGUOUS MEMORY ALLOCATION\n";
    cout<<"Enter the number of memory blocks available :";
    int n,mem;
    cin>>n;
    // define memory block
    // {int : memory size }

    multiset <int> mem_blocks;
    multiset <int> used;
    cout<<"Enter available memory blocks : ";
    for(int i=0;i<n;i++){
        cin>>mem;
        mem_blocks.insert(mem);
    }

    // get the memory allocation array
```

```
int internal_frag  = 0;
cout<<"\nEnter the memory requirement queue (-1 to exit):";
while(true){
    cout<<"Available locations :";

    for(auto k:mem_blocks)
        cout<<k<<" ";
    cout<<endl;
    cout<<"Required :";

    cin>>mem;
    if(mem == -1) break;

    else{
        // find upper & lower bound in the not used elements
        auto it1 = mem_blocks.upper_bound(mem);
        auto it2 = mem_blocks.lower_bound(mem);
        if(it1 == mem_blocks.end()){
            cout<<"Request too large! Please enter valid request\n";
        }
        else{
            if(*it2 == mem){
                // best fit.
                cout<<*it2<<" was allocated to the block.\n";
                mem_blocks.erase(it2);
                used.insert(*it2);
                cout<<"Fragmentation :"<<0<<endl;
            }
            else {
                cout<<*it1<<" was allocated to the block.\n";
                mem_blocks.erase(it1);
                used.insert(*it1);
                cout<<"Fragmentation :"<<*it1-mem<<endl;
                internal_frag += (*it1 - mem);
            }
        }
    }
}
cout<<"Total internal fragmentation caused :"<<internal_frag<<endl;
```

```
    return 0;
}
```

SCREENSHOTS

```
PS D:\IV Semester\OS\LAB> g++ .\best_fit_allocation.cpp
PS D:\IV Semester\OS\LAB> ./a
Program to simulate BEST FIT CONTIGUOUS MEMORY ALLOCATION
Enter the number of memory blocks available :5
Enter available memory blocks : 100 500 200 300 600

Enter the memory requirement queue (-1 to exit):Available locations :100 200 300 500 600
Required :112
200 was allocated to the block.
Fragmentation :88
Available locations :100 300 500 600
Required :417
500 was allocated to the block.
Fragmentation :83
Available locations :100 300 600
Required :600
600 was allocated to the block.
Fragmentation :0
Available locations :100 300
Required :112
300 was allocated to the block.
Fragmentation :188
Available locations :100
Required :-1
Total internal fragmentation caused :359
PS D:\IV Semester\OS\LAB>
```

Program 10

To show the working of the FIFO page replacement algorithm

- The following program highlights how the page frames are allocated inside a system when the replacement strategy is *first in first out*
- This is one of the simplest page replacement strategies used in the demand paging type systems but suffers from the belady's anomaly and is not very efficient.

CODE

```
#include<bits/stdc++.h>
#define all(x) x.begin(),x.end()
using namespace std;

int main(){
    ios::sync_with_stdio(0);
    cout<<"Program to simulate First in First out page replacement
algorithm\n";
    int frame_size;
    cout<<"Enter the frame size in your system :";
    cin>>frame_size;

    set<int> frame;
    map<int,int> indices;
    int hits,misses;
    hits = misses = 0;
    // Page queue
    cout<<"Enter the page request queue (-1 to exit ):\n";
    int c = 0,min,to_replace,page;
    vector<int> pages;
    pages.reserve(100);

    while(true){
```

```
        cin>>page;
        if(page == -1) break;
        pages.push_back(page);
    }

    // Evaluate
    for(auto page : pages){

        // associate earliest time stamps
        if(indices.find(page) == indices.end()){
            indices[page] = c;
        }

        // check if the page is in the frame or not
        if(frame.find(page)!= frame.end()){
            // page is found
            hits++;
            cout<<"Hit! "<<page<<" found in frame\n";
        }
        else{
            misses++;
            if(frame.size()!= frame_size){
                // no replacement required , just insert
                frame.insert(page);
                cout<<"Free frame available!\n";
            }
            else{
                // replacement required
                min = 1e5;
                for(auto k:frame){
                    if(indices[k] < min){
                        min = indices[k];
                        to_replace = k;
                    }
                }
                cout<<to_replace<<" is replaced by "<<page<<endl;

                // page is removed
                frame.erase(to_replace);
```

```
                indices.erase(to_replace);
                frame.insert(page);

            }
        }

        cout<<"Current Page Frame :\n";
        for(auto k:frame) cout<<k<<" ";

        cout<<endl;

        c++;

    }
    // Hit and miss ratio
    cout<<"Total numbers of hits :"<<hits<<endl;
    cout<<"Total number of misses :"<<misses<<endl;

    float hr,mr;
    hr = float(hits)/(c);
    mr = 1.0 - hr;
    cout<<"Hit ratio :"<<hr<<endl;
    cout<<"Miss ratio :"<<mr<<endl;
    return 0;
}
```

SCREENSHOTS

```
PS D:\IV Semester\OS\LAB\Page Replacement> g++ .\fcfs_pr.cpp
PS D:\IV Semester\OS\LAB\Page Replacement> ./a
Program to simulate First in First out page replacement algorithm
Enter the frame size in your system :3
Enter the page request queue (-1 to exit ):
3 2 1 3 4 1 6 2 4 3 4 2 1 4 5 2 1 -1
```

```
Free frame available!
Current Page Frame :
3
Free frame available!
Current Page Frame :
2 3
Free frame available!
Current Page Frame :
1 2 3
Hit! 3 found in frame
Current Page Frame :
1 2 3
3 is replaced by 4
Current Page Frame :
1 2 4
Hit! 1 found in frame
Current Page Frame :
1 2 4
2 is replaced by 6
Current Page Frame :
1 4 6
1 is replaced by 2
Current Page Frame :
2 4 6
Hit! 4 found in frame
Current Page Frame :
```

```
Current Page Frame :
2 4 6
4 is replaced by 3
Current Page Frame :
2 3 6
6 is replaced by 4
Current Page Frame :
2 3 4
Hit! 2 found in frame
Current Page Frame :
2 3 4
2 is replaced by 1
Current Page Frame :
1 3 4
Hit! 4 found in frame
Current Page Frame :
1 3 4
3 is replaced by 5
Current Page Frame :
1 4 5
4 is replaced by 2
Current Page Frame :
1 2 5
6 is replaced by 4
Current Page Frame :
2 3 4
```

```
1 2 5
6 is replaced by 4
Current Page Frame :
2 3 4
Hit! 2 found in frame
Current Page Frame :
2 3 4
2 is replaced by 1
Current Page Frame :
1 3 4
Hit! 4 found in frame
Current Page Frame :
1 3 4
3 is replaced by 5
Current Page Frame :
1 4 5
4 is replaced by 2
Current Page Frame :
1 2 5
Hit! 1 found in frame
Current Page Frame :
1 2 5
Total numbers of hits :6
Total number of misses :11
Hit ratio :0.352941
Miss ratio :0.647059
```

Program 11

To show the working of the LRU page replacement algorithm

- The following program highlights how the page frames are allocated inside a system when the replacement strategy is *least recently used*
- This strategy looks for the page ‘use’ in the request queue and replaces the page which was used the earliest and is present in the current frame

CODE

```
#include<bits/stdc++.h>
#define all(x) x.begin(),x.end()
#define pb(x) push_back(x)
using namespace std;
typedef vector<int> vi;
int main(){
   cout<<"LEAST RECENTLY USED PAGE REPLACEMENT ....\n";
   cout<<"Please enter the page request queue ( -1 to exit ) :\n";
    int miss,hits;
    hits = miss = 0;
    vi pages;
    int f_size;
    set<int> frame;

    cout<<"Enter the frame size of the system :";
    cin>>f_size;
    int ele;
    while(ele!=-1){
        cin>>ele;
        if(ele == -1) break;

        pages.pb(ele);
    }
    // GOALS
    // 1.Find the number of page faults encountered
    // 2.Find the hit rate and miss rate
```

```
    int c = 0;
    map< int,int > indices;
    int i,min_ind;
    for(auto k:pages){

        indices[k] = c;
        // if frame contains element

        // hit
        if(frame.find(k) != frame.end()){
            // got a hit
            cout<<"\nHit! Page "<<k<<" found in the frame.";
            hits++;
            c++;
            continue;
        }

//

        //else
        miss++;
        // frame does not have the page
        if(frame.size()!= f_size){
            frame.insert(k);

        }
        else{
            // Frame is full

                min_ind = 1e6;
                for(auto j:frame){
                    i = indices[j];
                    if(i < min_ind){

                        ele = j;
                        min_ind = i;
                    }
                }
```

```
                // at this point I have the minimum index element
                cout<<endl<<ele<<" is replaced with "<<k;
                // replace it
                frame.erase(ele);
                indices.erase(ele);
                // insert the current page
                frame.insert(k);
            }


    c++;
    cout<<endl;
    cout<<"Current Frame :\n";
    for(auto k:frame) cout<<k<<" ";

    }

    cout<<"\nNumber of misses :"<<miss<<endl;
    cout<<"Number of hits :"<<hits<<endl;
    float hr = float(hits)/(hits + miss);
    cout<<"The hit ratio of the algorithm :"<<hr<<endl;
    cout<<"Miss ratio of the algorithm :"<<1.0 - hr;

    return 0;}
```

SCREENSHOTS

```
PS D:\IV Semester\OS\LAB\Page Replacement> g++ lru.cpp
PS D:\IV Semester\OS\LAB\Page Replacement> ./a
LEAST RECENTLY USED PAGE REPLACEMENT ....
Please enter the page request queue ( -1 to exit ) :
Enter the frame size of the system :3
3 2 1 3 4 1 6 2 4 3 2 4 1 4 5 2 1 -1
```

```
Current Frame :
3
Current Frame :
2 3
Current Frame :
1 2 3
Hit! Page 3 found in the frame.
2 is replaced with 4
Current Frame :
1 3 4
Hit! Page 1 found in the frame.
3 is replaced with 6
Current Frame :
1 4 6
4 is replaced with 2
Current Frame :
1 2 6
1 is replaced with 4
Current Frame :
2 4 6
6 is replaced with 3
Current Frame :
2 3 4
Hit! Page 2 found in the frame.
Hit! Page 4 found in the frame.
3 is replaced with 1
```

```
Current Frame :
1 2 4
Hit! Page 4 found in the frame.
2 is replaced with 5
Current Frame :
1 4 5
1 is replaced with 2
Current Frame :
2 4 5
4 is replaced with 1
Current Frame :
1 2 5
Number of misses :12
Number of hits :5
The hit ratio of the algorithm :0.294118
Miss ratio of the algorithm :0.705882
```

Program 12

To show the working of the Second Chance page replacement algorithm

- In the Second Chance page replacement policy, the candidate pages for removal are considered in a round robin matter, and a page that has been accessed between consecutive considerations will not be replaced.
- The page replaced is the one that, when considered in a round robin matter, has not been accessed since its last consideration.

CODE

```
#include<bits/stdc++.h>
#include<windows.h>

using namespace std;
int main(){
    ios::sync_with_stdio(0);

    cout<<"Program to simulate SECOND CHANCE ALGORITHM\n";
    int f_size,len;
    cout<<"Enter frame size : ";
    cin>>f_size;
    list <int> frame;
    set <int> frame_copy;
    map <int,bool> reference;
    cout<<"Enter the number of page references : ";
    cin>>len;
    cout<<"Simulating pages...\n";
    int c;
    list<int>::iterator it2;

    bool found;
    int hit,miss,count,page,to_replace;
    hit = count = miss = 0;
    while(len--){

        page = rand()%9 + 1;
```

```
        // page was referenced
        reference[page] = 1;

        if(frame_copy.find(page) != frame_copy.end()){
            cout<<"Hit! "<<page<<" found in frame\n";
            hit++;
        }
        else{
            miss++;


            if(frame.size() != f_size){
                // fifo insertion
                frame.push_back(page);
                frame_copy.insert(page);
            }

            else{

                // replacement needs to be done
                found = false;
                auto it = frame.begin();
                while(found == false){

                    if(reference[*it] == 1){
                        reference[*it] = 0;
                        //second chance given
                        it++;
                        if(it==frame.end()) // loop back
                            it = frame.begin();
                    }
                    // found a page
                    else{
                        found = true;
                        to_replace = *it;

                        //it2 is the iterator pointing to the
                        // element before which we want to insert
                        // new page
```

```
                        it2 = frame.erase(it);
                        frame.insert(it2,page);

                        // update the copy
                        frame_copy.erase(to_replace);
                        frame_copy.insert(page);
                    }
                }

                cout<<to_replace<<" was replaced by "<<page<<endl;
            }
        }

    cout<<"Current frame : ";
    for(auto k:frame) cout<<k<<" ";
    cout<<endl;

    cout<<"Reference : ";
    for(auto k:frame) cout<<reference[k]<<" ";
    cout<<endl;
    count++;
    _sleep(400);
    }


    cout<<"Total hits :"<<hit<<endl;
    cout<<"Total missed :"<<miss<<endl;
    float hr = float(hit)/count;
    cout<<"Hit ratio :"<<hr<<endl;
    cout<<"Miss ratio :"<<1.0 - hr;

    return 0;
}
```

SCREENSHOTS

```
PS D:\IV Semester\OS\LAB\Page Replacement> ./a
Program to simulate SECOND CHANCE ALGORITHM
Enter frame size : 4
Enter the number of page references : 12
Simulating pages...
Current frame : 6
Reference : 1
Current frame : 6 9
Reference : 1 1
Current frame : 6 9 8
Reference : 1 1 1
Current frame : 6 9 8 5
Reference : 1 1 1 1
Hit! 9 found in frame
Current frame : 6 9 8 5
Reference : 1 1 1 1
6 was replaced by 2
Current frame : 2 9 8 5
Reference : 1 0 0 0
9 was replaced by 4
```

```
Current frame : 2 4 8 5
Reference : 0 1 0 0
2 was replaced by 1
Current frame : 1 4 8 5
Reference : 1 1 0 0
Hit! 8 found in frame
Current frame : 1 4 8 5
Reference : 1 1 1 0
5 was replaced by 3
Current frame : 1 4 8 3
Reference : 0 0 0 1
1 was replaced by 9
Current frame : 9 4 8 3
Reference : 1 0 0 1
Hit! 3 found in frame
Current frame : 9 4 8 3
Reference : 1 0 0 1
Total hits :3
Total missed :9
Hit ratio :0.25
Miss ratio :0.75
```

Program 13

To show the working of the Enhanced Second Chance page replacement algorithm

- In the Enhanced Second Chance page replacement policy, the candidate pages for removal are considered with a <reference,modify> bit pair.
- The highest priority is given to the page with <0,0> as its bit pair followed by <0,1> pair. This is because of the fact that if the page has *modify* bit as 0, there is no requirement of disk access and thus lesser disk I/O

CODE

```
#include<bits/stdc++.h>
#define tr(v,it) for(auto it = v.begin();it!=v.end();it++)
using namespace std;
int main(){
    ios::sync_with_stdio(0);

    cout<<"Program to simulate ENHANCED SECOND CHANCE ALGORITHM\n";
    int f_size,len;
    cout<<"Enter frame size : ";
    cin>>f_size;
    list <int> frame;
    set <int> frame_copy;
    map <int,bool> reference;
    map <int,bool> modify;
    cout<<"Enter the number of page references : ";
    cin>>len;
    cout<<"Simulating pages...\n";
    int c;
    list<int>::iterator it2;

    bool found;
    int hit,miss,count,page,to_replace,write,bit;
    hit = count = miss = 0;
    while(len--){

        page = rand()%9 + 1;
        cout<<"Page is :"<<page;
```

```
        write = rand()%2;

        //change modify bit
        if(write == 0) cout<<", Reading done...\n";
        else cout<<", Writing done...\n";
        modify[page] = write;

        // page was referenced
        reference[page] = 1;

        // page found
        if(frame_copy.find(page) != frame_copy.end()){
            cout<<"Hit! "<<page<<" found in frame\n";
            hit++;
        }

        //page not found
        else{
            miss++;

            if(frame.size() != f_size){
                // fifo insertion
                frame.push_back(page);
                frame_copy.insert(page);
            }

            else{

                // replacement needs to be done
                found = false;
                 // 0,0 , then 0,1
                int c = 0;
                bit = 0;
                while(found == false)
                {
                    // need to loop twice to for a bit
                    // to incorporate the second chance
                        if(c == 2) {
```

```
                        bit = 1-bit;
                        c = 0;
                    }
                    c++;
                    // now iterate the list
                    tr(frame,it){
                        if(reference[*it] == 1){
                            // give a second chance
                            reference[*it] = 0;
                        }
                        else{

                            if(modify[*it] == bit){
                                // found a target
                                found = true;
                                to_replace = *it;

                                it2 = frame.erase(it);
                                frame.insert(it2,page);

                                // update the copy
                                frame_copy.erase(to_replace);
                                frame_copy.insert(page);

                                // update references
                                reference[to_replace] = 0;
                                modify[to_replace] = 0;

                                break;
                            }

                        }
                    }

            }

        cout<<to_replace<<" was replaced by "<<page<<endl;

    }
```

```
        }

    cout<<"Current frame : ";
    for(auto k:frame) cout<<k<<" ";
    cout<<endl;

    cout<<"Reference : ";
    for(auto k:frame) cout<<reference[k]<<" ";
    cout<<endl;
    cout<<"Modify : ";
    for(auto k:frame) cout<<modify[k]<<" ";
    cout<<endl;
    count++;
    _sleep(400);
    }

    cout<<"Total hits :"<<hit<<endl;
    cout<<"Total missed :"<<miss<<endl;
    float hr = float(hit)/count;
    cout<<"Hit ratio :"<<hr<<endl;
    cout<<"Miss ratio :"<<1.0 - hr;



    return 0;
}
```

SCREENSHOTS

```
PS D:\IV Semester\OS\LAB\Page Replacement\codes> ./a
Program to simulate ENHANCED SECOND CHANCE ALGORITHM
Enter frame size : 3
Enter the number of page references : 12
Simulating pages...
```

1

```
Page is :6, Writing done...
Current frame : 6
Reference : 1
Modify : 1
Page is :8, Reading done...
Current frame : 6 8
Reference : 1 1
Modify : 1 0
Page is :9, Reading done...
Current frame : 6 8 9
Reference : 1 1 1
Modify : 1 0 0
Page is :4, Reading done...
8 was replaced by 4
Current frame : 6 4 9
Reference : 0 1 0
Modify : 1 0 0
Page is :8, Reading done...
9 was replaced by 8
Current frame : 6 4 8
Reference : 0 0 1
Modify : 1 0 0
Page is :9, Writing done...
4 was replaced by 9
Current frame : 6 9 8
Reference : 0 1 1
Modify : 1 1 0
Page is :8, Writing done...
Hit! 8 found in frame
Current frame : 6 9 8
Reference : 0 1 1
Modify : 1 1 1
Page is :8, Writing done...
```

2

```
Page is :8, Writing done...
Hit! 8 found in frame
Current frame : 6 9 8
Reference : 0 1 1
Modify : 1 1 1
Page is :8, Reading done...
Hit! 8 found in frame
Current frame : 6 9 8
Reference : 0 1 1
Modify : 1 1 0
Page is :4, Reading done...
8 was replaced by 4
Current frame : 6 9 4
Reference : 0 0 1
Modify : 1 1 0
Page is :1, Reading done...
4 was replaced by 1
Current frame : 6 9 1
Reference : 0 0 1
Modify : 1 1 0
Current frame : 6 9 8
Reference : 0 1 1
Modify : 1 1 1
Page is :8, Writing done...
Hit! 8 found in frame
Current frame : 6 9 8
Reference : 0 1 1
Modify : 1 1 1
Page is :8, Reading done...
Hit! 8 found in frame
Current frame : 6 9 8
Reference : 0 1 1
```

3

```
Current frame : 6 9 8
Reference : 0 1 1
Modify : 1 1 0
Page is :4, Reading done...
8 was replaced by 4
Current frame : 6 9 4
Reference : 0 0 1
Modify : 1 1 0
Page is :1, Reading done...
4 was replaced by 1
Current frame : 6 9 1
Reference : 0 0 1
Modify : 1 1 0
Page is :6, Writing done...
Hit! 6 found in frame
Current frame : 6 9 1
Reference : 1 0 1
Modify : 1 1 0
Total hits :4
Total missed :8
Hit ratio :0.333333
Miss ratio :0.666667
```

Program 14

Program to show the working of the LFU page replacement algorithm

- In this algorithm, the operating system keeps track of all pages in the memory in a queue.
- When a page needs to be replaced, the operating system chooses the page which is least frequently used for the replacement with the incoming page.
- *Hash Table* data structure is used to keep the frequency array for a page and *set* is used to simulate a frame in C++.

CODE

```
#include<bits/stdc++.h>
#define for0(i,n) for(int i=0;i<n;i++)
#include<windows.h>
using namespace std;
int main(){
    ios::sync_with_stdio(0);

    //Declare frame
    set<int> frame;

    //Declare  hash table for frequencies
    unordered_map <int,int> freq;

    cout<<"Program to simulate LEAST FREQUENTLY USED page replacement\n";
    int f_size,len,min,to_replace;

    //SIMULATION
    cout<<"Enter the frame size :";
    cin>>f_size;
    cout<<"Enter how many pages you want to simulate :";
    cin>>len;
    cout<<"Simulating page request queue :\n";

    int hit,miss,count,page;
    hit = count = miss = 0;
```

```
for0(i,len){
    page = rand()%9 + 1;
    cout<<"Page :"<<page<<endl;

    // referenced
    freq[page]++;

    if(frame.find(page) != frame.end()){
        // hit
        cout<<"Hit! "<<page<<" found in frame\n";
        hit++;
    }
    else{
        //miss
        miss++;
        cout<<"Miss! ";
        if(frame.size() != f_size){
            // some empty space available
            cout<<page<<" inserted\n";
            frame.insert(page);
        }

        else{
            // no empty space, replacement required
            min = 1e7;

            // choose minimum used page
            for(auto k:frame){
                if(freq[k] <= min){
                    min = freq[k];
                    to_replace = k;
                }
            }
            cout<<to_replace<<" replace with "<<page<<endl;
            // replace the page
            frame.erase(to_replace);
            frame.insert(page);
            freq[to_replace] = 0; // erased page
        }
```

```
        }

    cout<<"Current frame : ";
    for(auto k:frame) cout<<k<<" ";

    cout<<endl;
    count++;
    _sleep(400);
    }

    cout<<"Total hits :"<<hit<<endl;
    cout<<"Total missed :"<<miss<<endl;
    float hr = float(hit)/count;
    cout<<"Hit ratio :"<<hr<<endl;
    cout<<"Miss ratio :"<<1.0 - hr;
    return 0;
}
```

SCREENSHOTS

```
PS D:\IV Semester\OS\LAB\Page Replacement> ./a
Program to simulate LEAST FREQUENTLY USED page replacement
Enter the frame size :4
Enter how many pages you want to simulate :12
Simulating page request queue :
```

```
Page :6
Miss! 6 inserted
Current frame : 6
Page :9
Miss! 9 inserted
Current frame : 6 9
Page :8
Miss! 8 inserted
Current frame : 6 8 9
Page :5
Miss! 5 inserted
Current frame : 5 6 8 9
Page :9
Hit! 9 found in frame
Current frame : 5 6 8 9
Page :2
Miss! 8 replace with 2
Current frame : 2 5 6 9
Page :4
Miss! 6 replace with 4
Current frame : 2 4 5 9
```

```
Page :1
Miss! 5 replace with 1
Current frame : 1 2 4 9
Page :8
Miss! 4 replace with 8
Current frame : 1 2 8 9
Page :3
Miss! 8 replace with 3
Current frame : 1 2 3 9
Page :9
Hit! 9 found in frame
Current frame : 1 2 3 9
Page :3
Hit! 3 found in frame
Current frame : 1 2 3 9
Total hits :3
Total missed :9
Hit ratio :0.25
Miss ratio :0.75
```

Program 15

Program to show the working of the FCFS disk scheduling algorithm

- This is the simplest Disk Scheduling algorithm used for scheduling the disk access queue.
- The Disk Requests are served, as they come in, by the disk head

Advantages

- Every request gets a fair chance
- No indefinite postponement

Disadvantages

- Does not try to optimize seek time
- May not provide the best possible service

CODE

```
#include<bits/stdc++.h>
using namespace std;
int get_movement(list<int> &Q, int curr){
    int ans = 0;
    for(auto k:Q){
        ans+= abs(k-curr);
        curr = k;
    }
    return ans;
}
int main(){
    ios::sync_with_stdio(0);
    int curr_head,size_disk;
    cout<<"FIRST COME FIRST SERVE disk scheduling...\n";
    cout<<"Enter current head position :";
    cin>>curr_head;
    cout<<"Enter the disk size :";
    cin>>size_disk;

    cout<<"Enter the ready queue for DISK ACCESS :(-1 to end)";
```

```
    list<int> disk_queue;
    int d;
    while(true){
        cin>>d;
        if(d == -1)
            break;
        if(d < 0 || d>=size_disk){
            cout<<"Invalid value\nEnter current element again\n";
            continue;
        }
        disk_queue.push_back(d);
    }
    // calculate the total head movement and average seek time
    int head_move = get_movement(disk_queue,curr_head);

    float avg_head = (float(head_move)/disk_queue.size());

    cout<<"TOTAL HEAD MOVEMENT : "<<head_move<<endl;
    cout<<"AVERAGE HEAD MOVEMENT : "<<avg_head<<endl;

    return 0;
}
```

SCREENSHOTS

```
PS D:\IV Semester\OS\LAB\Disk Scheduling> g++ .\fifo_sched
uling.cpp
PS D:\IV Semester\OS\LAB\Disk Scheduling> ./a
FIRST COME FIRST SERVE disk scheduling...
Enter current head position :50
Enter the disk size :200
Enter the ready queue for DISK ACCESS :(-1 to end)82
170
43
140
24
16
190
-1
TOTAL HEAD MOVEMENT : 642
AVERAGE HEAD MOVEMENT : 91.7143
```

Program 16

Program to show the working of the SSTF disk scheduling algorithm

- In SSTF (Shortest Seek Time First), requests having the shortest seek time are executed first.
- So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time

Advantages:

- Average Response Time decreases
- Throughput increases

Disadvantages:

- Overhead to calculate seek time in advance
- Can cause Starvation for a request if it has higher seek time as compared to incoming requests

CODE

```
#include<bits/stdc++.h>
#define all(x) x.begin(),x.end()
using namespace std;
int get_movement(vector<int> &Q, int curr){

    int ind = upper_bound(all(Q),curr) - Q.begin();

    // the first element > curr_head_pos

    // just look at immediate left and immediate right
    int n = Q.size();
    if(n == 0)
        return -1;
    bool visited[n] = {false};
    bool done = false;
    int ans,left,right;
```

```
    if(ind == n)  right = 1e6; // if the value is very big for current
head
    else right = abs(curr - Q[ind]); // ind is the first index > current
head

    if(ind == 0) left = 1e6;
    else left = abs(curr - Q[ind-1]);

    if(left <= right) // left has shorter seek time
        {ans = left;
        ind--;
        curr = Q[ind];
        visited[ind] = true;}

    else
        {ans = right; // no need to increment as already there...
        visited[ind] = true;
        curr = Q[ind];}

    int templ,tempr;

    while(done == false){

        templ = ind-1;
        while(visited[templ] == true && templ >=0)
            templ--;

        if(templ>=0)
            left = abs(curr - Q[templ]);
        else
            left = 1e6;
        tempr = ind+1;
        while(visited[tempr] == true && tempr <n)
            tempr++;

        if(tempr != n)
            right = abs(curr - Q[tempr]);
        else
            right = 1e6;
```

```
        if(left < right ){
            ans+= left;
            cout<<Q[templ]<<" chosen for head at "<<curr<<endl;
            ind = templ;
            curr = Q[ind];
            visited[ind] = true;
        }
        else if(left > right){
            ans+= right;
            cout<<Q[tempr]<<" chosen for head at "<<curr<<endl;
            ind = tempr;
            curr = Q[ind];
            visited[ind] = true;
        }
        else if(left == right && left!= 1e6){
            ans+= left;
            ind= templ;
            curr = Q[ind];
            visited[ind] = true;
        }
        else
            done = true;
    }

    return ans;
}
int main(){
    ios::sync_with_stdio(0);
    int curr_head,size_disk;
    cout<<"SHORTEST SEEK TIME FIRST disk scheduling...\n";
    cout<<"Enter current head position :";
    cin>>curr_head;
    cout<<"Enter the disk size :";
    cin>>size_disk;

    cout<<"Enter the ready queue for DISK ACCESS :(-1 to end)\n";
    vector<int> disk_queue;
    int d;
```

```
    while(true){
        cin>>d;
        if(d == -1)
            break;
        if(d < 0 || d>=size_disk){
            cout<<"Invalid value\nEnter current element again\n";
            continue;
        }
        disk_queue.push_back(d);
    }
    sort(disk_queue.begin(),disk_queue.end());

    // calculate the total head movement and average seek time

    int head_move = get_movement(disk_queue,curr_head);

    float avg_head = (float(head_move)/disk_queue.size());

    cout<<"TOTAL HEAD MOVEMENT : "<<head_move<<endl;
    cout<<"AVERAGE HEAD MOVEMENT : "<<avg_head<<endl;

    return 0;
}
```

SCREENSHOTS

```
PS D:\IV Semester\OS\LAB\Disk Scheduling> ./a
SHORTEST SEEK TIME FIRST disk scheduling...
Enter current head position :50
Enter the disk size :200
Enter the ready queue for DISK ACCESS :(-1 to end)
82 170 43 140 24 16 190 -1
24 chosen for head at 43
16 chosen for head at 24
82 chosen for head at 16
140 chosen for head at 82
170 chosen for head at 140
190 chosen for head at 170
TOTAL HEAD MOVEMENT : 208
AVERAGE HEAD MOVEMENT : 29.7143
```

Program 17

Program to show the working of the C-SCAN disk scheduling algorithm

- In the CSCAN algorithm, the disk arm, instead of reversing its direction, goes to the other end of the disk and starts servicing the requests from there.
- So, the disk arm moves in a circular fashion and since this algorithm is similar to the SCAN algorithm, it is known as C-SCAN (Circular SCAN).

CODE

```
#include<bits/stdc++.h>
#define all(x) x.begin(),x.end()
using namespace std;
int get_movement(vector<int> &Q, int curr,char move,int disk_size){
    int ans = 0;

    if(move == 'R'){
        ans+= disk_size - 1 - curr;
        // if it moved right initially,
        // would stop at the last index or
        // LOWER BOUND OF curr after moving right again

        auto it = lower_bound(all(Q),curr) - Q.begin();
        it--;
        // if the smaller element exists
        // cout<<"Lower bound :"<<Q[it]<<endl;
        if(it>=0)
            ans+= Q[it];
    }

    else{
        // if it moved left initially,
        // would stop at first element > curr
        // or the UPPER BOUND
        ans+= curr;
```

```
        auto it = upper_bound(all(Q),curr);
        // if the bigger element exists
        if(it!=Q.end())
             ans+= disk_size - 1 - (*it);
    }

    ans+= (disk_size-1);

    return ans;
}
int main(){
    ios::sync_with_stdio(0);
    int curr_head,size_disk;
    cout<<"C-SCAN disk scheduling...\n";
    // get current head pointer
    cout<<"Enter current head position :";
    cin>>curr_head;

    // get the disk size
    cout<<"Enter the disk size :";

    cin>>size_disk;

    // get the head movement
    char move;
    cout<<"Enter the initial head movement(L/R) :";
    cin>>move;
    cout<<"Enter the ready queue for DISK ACCESS :(-1 to end)";
    vector<int> disk_queue;
    int d;
    while(true){
        cin>>d;
        if(d == -1)
            break;
        if(d < 0 || d>=size_disk){
            cout<<"Invalid value\nEnter current element again\n";
            continue;
        }
        disk_queue.push_back(d);
```

```
    }

    // calculate the total head movement and average seek time
    sort(all(disk_queue));
    int head_move = get_movement(disk_queue,curr_head,move,size_disk);

    float avg_head = (float(head_move)/disk_queue.size());

    cout<<"TOTAL HEAD MOVEMENT : "<<head_move<<endl;
    cout<<"AVERAGE HEAD MOVEMENT : "<<avg_head<<endl;

    return 0;
}
```

SCREENSHOTS

```
PS D:\IV Semester\OS\LAB\Disk Scheduling> ./a
C-SCAN disk scheduling...
Enter current head position :50
Enter the disk size :200
Enter the initial head movement(L/R) :L
Enter the ready queue for DISK ACCESS :(-1 to end)24
16
43
82
100
142
170
190
-1
TOTAL HEAD MOVEMENT : 366
AVERAGE HEAD MOVEMENT : 45.75
```

Program 18

Program to show the working of the LOOK disk scheduling algorithm

- The disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only.
- It prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

CODE

```
#include<bits/stdc++.h>
#define all(x) x.begin(),x.end()
using namespace std;
int get_movement(vector<int> &Q, int curr,char move,int disk_size){

    int min = Q[0];
    int max = Q.back();
    int ans = 0;
    if(move == 'R'){
        if(curr<min){
            ans = max - curr;
        }
        else if(curr > max){
            ans = curr - min;
        }
        else{
            ans+= abs(max-min) + abs(max - curr);
        }
    }
    // go left
    else{
        if(curr < min){
            ans+= max - curr;
        }
        else if(curr > max){
```

```
            ans+= curr - min;
        }
        else{
            ans+= abs(curr-min) + abs(max-min);
        }
    }
    return ans;
}
int main(){
    ios::sync_with_stdio(0);
    int curr_head,size_disk;
    cout<<"LOOK disk scheduling...\n";
    // get current head pointer
    cout<<"Enter current head position :";
    cin>>curr_head;

    // get the disk size
    cout<<"Enter the disk size :";

    cin>>size_disk;

    // get the head movement
    char move;
    cout<<"Enter the initial head movement(L/R) :";
    cin>>move;
    cout<<"Enter the ready queue for DISK ACCESS :(-1 to end)";
    vector<int> disk_queue;
    int d;
    while(true){
        cin>>d;
        if(d == -1)
            break;
        if(d < 0 || d>=size_disk){
            cout<<"Invalid value\nEnter current element again\n";
            continue;
        }
        disk_queue.push_back(d);
    }
```

```
    // calculate the total head movement and average seek time
    sort(all(disk_queue));
    int head_move = get_movement(disk_queue,curr_head,move,size_disk);

    float avg_head = (float(head_move)/disk_queue.size());

    cout<<"TOTAL HEAD MOVEMENT : "<<head_move<<endl;
    cout<<"AVERAGE HEAD MOVEMENT : "<<avg_head<<endl;

    return 0;
}
```

SCREENSHOTS

```
PS D:\IV Semester\OS\LAB\Disk Scheduling> ./a
LOOK disk scheduling...
Enter current head position :50
Enter the disk size :200
Enter the initial head movement(L/R) :R
Enter the ready queue for DISK ACCESS :(-1 to end)82
170
43
140
24
16
190
-1
TOTAL HEAD MOVEMENT : 314
AVERAGE HEAD MOVEMENT : 44.8571
```