**Medicaps University, Indore**

**Department of Computer Science & Engineering**

***Name-Dhananjay Porwal***

***Enrollment No.-EN19CS301110***

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.no.** | **Practical Performed** | **Date** | **Page no.** |
| 1. | Study of latest Operating System and their latest features | 19/01/2021 | 2 |
| 2. | WAP to implement FCFS Scheduling Algorithm | 26/01/2021 | 3 |
| 3. | WAP to implement Shortest Job First Scheduling Algorithm | 26/01/2021 | 5 |
| 4. | WAP to implement Round Robin Scheduling Algorithm | 02/02/2021 | 8 |
| 5. | WAP to implement Priority Scheduling Algorithm | 09/02/2021 | 10 |
| 6. | WAP to implement the Producer Consumer Problem | 16/02/2021 | 13 |
| 7. | WAP to implement the Dining Philosopher’s Problem | 16/02/2021 | 14 |
| 8. | WAP to implement the Banker’s Algorithm | 23/02/2021 | 17 |
| 9. | WAP to implement the concept of First Fit memory allocation | 02/03/2021 | 20 |
| 10. | WAP to implement the concept of Best fit memory allocation. | 02/03/2021 | 22 |
| 11. | WAP to implement the concept of Worst Fit memory allocation. | 02/03/2021 | 24 |
| 12. | WAP a program to implement FIFO Page Replacement algorithm. | 09/03/2021 | 25 |
| 13. | WAP a program to implement LRU Page Replacement algorithms | 16/03/2021 | 26 |
| 14. | WAP a program to implement Optimal Page Replacement algorithm. | 23/03/2021 | 27 |
| 15. | WAP to implement FCFS Disk Scheduling | 30/03/2021 | 28 |
| 16. | WAP to implement SCAN Disk Scheduling | 13/04/2021 | 29 |
| 17. | WAP to implement C-SCAN Disk Scheduling | 13/04/2021 | 31 |
| 18. | WAP to implement LOOK Disk Scheduling | 20/04/2021 | 33 |
| 19. | WAP to implement C-LOOK Disk Scheduling | 20/04/2021 | 35 |

**PRACTICAL-01**

**THE STUDY OF THE DIFFERENT OS USED IN DIFFERENT OBJECTS.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Serial**  **No.** | **Name of the operating system** | | **Latest Release** | | **Key features** |
| **1.** | Ms window | | Oct-20-2020 | | Snap enhancement ,Multiple desktops,Enterprise features. |
| **2.** | Ubuntu 20.10 | | Oct-22-2020 | | Theme variants and aubergine touch,visual and performance improvements, DND button, fractional scaling. |
| **3.** | Mac OS 11.3 | | Dec-14-2020 | | Updated menu bar,floating dock,redesigned sheets,full heights side bar and interactive notifaction. |
| **4.** | Fedora 33 | | Oct-27-2020 | | Best power management , better end-user software, dynamic fire-wall,virtual desktop support,package manager. |
| **5.** | Solaris 11.4 | | Aug-28-2018 | | Man command enhancements,paps print filter,iconv framework modernization and locale name fallback mechanism. |
| **6.** | Free BSD | | OCT-27-2020 | | Documentation and support ,OS compatibility layers , virtualization,hardware compatibility , portability and security. |
| **7.** | Chrome OS | | Jan-07-2021 | | 100% interface web ,free, open source , not android, eliminates time sucks. |
| **8.** | Cent OS | | Nov-12-2020 | | Desktop background , networking arena ,software management , virtualization , installation and image creation. |
| **9.** | Debian 10.9 | | Dec-05-2020 | | Contains free software, desktop environments , localization and multimedia support. |
| **10.** | Deepin 20.2 | | July-19-2019 | | Unified style DDE, dual kernel system installation,personalized notification management , optimized fingerprint recognition. |
| **11.** | Android OS | | Sep-08-2020 | | NFC , Alternate keyboards , infrared transmission , no touch control , automation , storage and battery swap , custom ROMs. |
|  |  | |  | |
| **12.** | Amiga OS | | Jan-12-2021 | | Virtual memory,fully skinnable GUI , limited memory protection ,integrated debugger CD/DVD writer support, improved TCP/IP networking. |
| **13.** | BeOS | | DEC-01-2009 | | High performance , reliability , integrated services and compatibility. |
| **14.** | Darwin | | Dec-16-2020 | | User level memory pagers , ports , threads. |
| **15.** | Morph OS | | DEC-31-2020 | | AmigaDOS commands, local and global variables, command substitution, command redirection, named and unnamed pipes, programmable menus, multiple shells in a window, ANSI compatibility, colour selection. |

**PRACTICAL-02**

**WAP to implement FCFS Scheduling Algorithm.**

**NON-PREEMPTIVE**

#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 findavgTime( 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 processes[] = { 1, 2, 3};

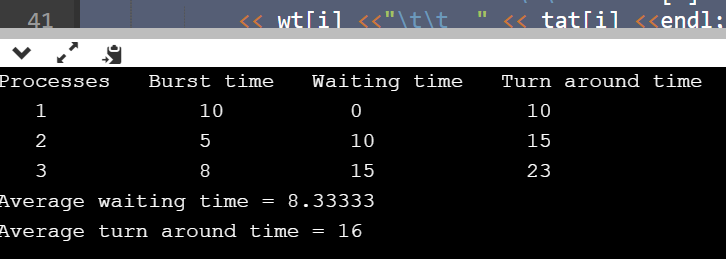
int n = sizeof processes / sizeof processes[0];

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

findavgTime(processes, n, burst\_time);

return 0;

}



**PRACTICAL-03**

**WAP to implement Shortest Job First Scheduling Algorithm.**

**NON-PREEMPTIVE**

#include<iostream>

using namespace std;

int mat[10][6];

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

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void arrangeArrival(int num, int mat[][6])

{

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

{

for(int j=0; j<num-i-1; j++)

{

if(mat[j][1] > mat[j+1][1])

{

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

{

swap(mat[j][k], mat[j+1][k]);

} } } } }

void completionTime(int num, int mat[][6])

{

int temp, val;

mat[0][3] = mat[0][1] + mat[0][2];

mat[0][5] = mat[0][3] - mat[0][1];

mat[0][4] = mat[0][5] - mat[0][2];

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

{

temp = mat[i-1][3];

int low = mat[i][2];

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

{

if(temp >= mat[j][1] && low >= mat[j][2])

{

low = mat[j][2];

val = j;

}

}

mat[val][3] = temp + mat[val][2];

mat[val][5] = mat[val][3] - mat[val][1];

mat[val][4] = mat[val][5] - mat[val][2];

for(int k=0; k<6; k++)

{

swap(mat[val][k], mat[i][k]);

} } }

int main()

{

int num, temp;

cout<<"Enter number of Process: ";

cin>>num;

cout<<"...Enter the process ID...\n";

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

{

cout<<"...Process "<<i+1<<"...\n";

cout<<"Enter Process Id: ";

cin>>mat[i][0];

cout<<"Enter Arrival Time: ";

cin>>mat[i][1];

cout<<"Enter Burst Time: ";

cin>>mat[i][2];

}

cout<<"Before Arrange...\n";

cout<<"Process ID\tArrival Time\tBurst Time\n";

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

{

cout<<mat[i][0]<<"\t\t"<<mat[i][1]<<"\t\t"<<mat[i][2]<<"\n";

}

arrangeArrival(num, mat);

completionTime(num, mat);

cout<<"Final Result...\n";

cout<<"Process ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n";

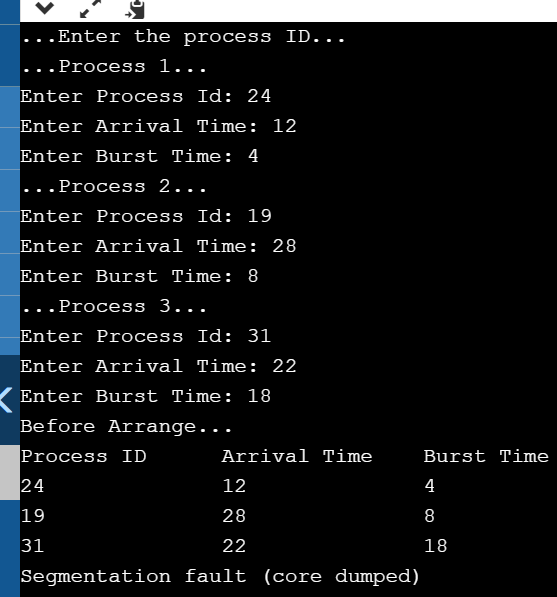
for(int i=0; i<num; i++)

{

cout<<mat[i][0]<<"\t\t"<<mat[i][1]<<"\t\t"<<mat[i][2]<<"\t\t"<<mat[i][4]<<"\t\t"<<mat[i][5]<<"\n";

}

}



**PRACTICAL-04**

**WAP to implement Round Robin Scheduling Algorithm.**

#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];

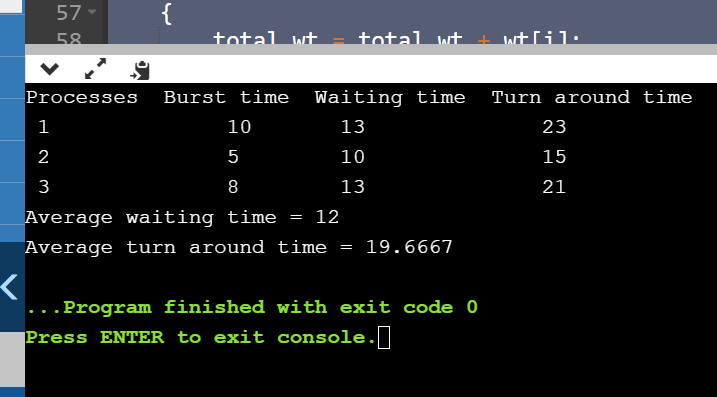
int burst\_time[] = {10, 5, 8};

int quantum = 2;

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

return 0;

}



**PRACTICAL-05**

**WAP to implement Priority Scheduling Algorithm.**

**PREEMPTIVE**

#include <iostream>

#include <algorithm>

#include <iomanip>

#include <string.h>

using namespace std;

struct process {

int pid;

int arrival\_time;

int burst\_time;

int priority;

int start\_time;

int completion\_time;

int turnaround\_time;

int waiting\_time;

int response\_time;

};

int main() {

int n;

struct process p[100];

float avg\_turnaround\_time;

float avg\_waiting\_time;

float avg\_response\_time;

float cpu\_utilisation;

int total\_turnaround\_time = 0;

int total\_waiting\_time = 0;

int total\_response\_time = 0;

int total\_idle\_time = 0;

float throughput;

int burst\_remaining[100];

int is\_completed[100];

memset(is\_completed,0,sizeof(is\_completed));

cout << setprecision(2) << fixed;

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

cin>>n;

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

cout<<"Enter arrival time of process "<<i+1<<": ";

cin>>p[i].arrival\_time;

cout<<"Enter burst time of process "<<i+1<<": ";

cin>>p[i].burst\_time;

cout<<"Enter priority of the process "<<i+1<<": ";

cin>>p[i].priority;

p[i].pid = i+1;

burst\_remaining[i] = p[i].burst\_time;

cout<<endl;

}

int current\_time = 0;

int completed = 0;

int prev = 0;

while(completed != n) {

int idx = -1;

int mx = -1;

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

if(p[i].arrival\_time <= current\_time && is\_completed[i] == 0) {

if(p[i].priority > mx) {

mx = p[i].priority;

idx = i;

}

if(p[i].priority == mx) {

if(p[i].arrival\_time < p[idx].arrival\_time) {

mx = p[i].priority;

idx = i;

} } } }

if(idx != -1) {

if(burst\_remaining[idx] == p[idx].burst\_time) {

p[idx].start\_time = current\_time;

total\_idle\_time += p[idx].start\_time - prev; }

burst\_remaining[idx] -= 1;

current\_time++;

prev = current\_time;

if(burst\_remaining[idx] == 0) {

p[idx].completion\_time = current\_time;

p[idx].turnaround\_time = p[idx].completion\_time - p[idx].arrival\_time;

p[idx].waiting\_time = p[idx].turnaround\_time - p[idx].burst\_time;

p[idx].response\_time = p[idx].start\_time - p[idx].arrival\_time;

total\_turnaround\_time += p[idx].turnaround\_time;

total\_waiting\_time += p[idx].waiting\_time;

total\_response\_time += p[idx].response\_time;

is\_completed[idx] = 1;

completed++; } }

else {

current\_time++; } }

int min\_arrival\_time = 10000000;

int max\_completion\_time = -1;

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

min\_arrival\_time = min(min\_arrival\_time,p[i].arrival\_time);

max\_completion\_time = max(max\_completion\_time,p[i].completion\_time);

}

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

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

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

cpu\_utilisation = ((max\_completion\_time - total\_idle\_time) / (float) max\_completion\_time )\*100;

throughput = float(n) / (max\_completion\_time - min\_arrival\_time);

cout<<endl<<endl;

cout<<"#P\t"<<"AT\t"<<"BT\t"<<"PRI\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<endl;

for(int i = 0; i < n; i++) { cout<<p[i].pid<<"\t"<<p[i].arrival\_time<<"\t"<<p[i].burst\_time<<"\t"<<p[i].priority<<"\t"<<p[i].start\_time<<"\t"<<p[i].completion\_time<<"\t"<<p[i].turnaround\_time<<"\t"<<p[i].waiting\_time<<"\t"<<p[i].response\_time<<"\t"<<"\n"<<endl;

} cout<<"Average Turnaround Time = "<<avg\_turnaround\_time<<endl;

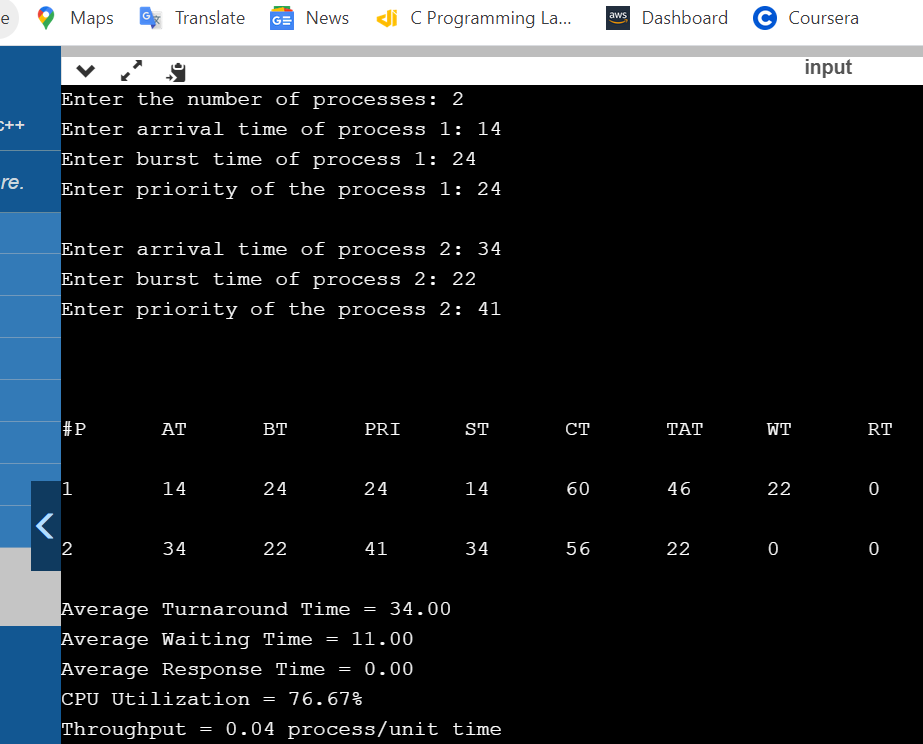
cout<<"Average Waiting Time = "<<avg\_waiting\_time<<endl;

cout<<"Average Response Time = "<<avg\_response\_time<<endl;

cout<<"CPU Utilization = "<<cpu\_utilisation<<"%"<<endl;

cout<<"Throughput = "<<throughput<<" process/unit time"<<endl;

}



**PRACTICAL-06**

**WAP to implement the Producer Consumer Problem.**

#include <iostream>

#include <pthread.h>

#include <semaphore.h>

#include <random>

#include <unistd.h>

using namespace std;

#define BUFFER\_SIZE 10

int buffer[BUFFER\_SIZE];

int index=0;

sem\_t full,empty;

pthread\_mutex\_t mutex;

void\* produce(void\* arg){

while(1){

sleep(1);

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

int item = rand()%100;

buffer[index++] = item;

cout<<"Produced "<<item<<endl;

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

} }

void\* consume(void\* arg){

while(1){

sleep(1);

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

int item = buffer[--index];

cout<<"Consumed "<<item<<endl;

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

} }

int main(){

pthread\_t producer,consumer;

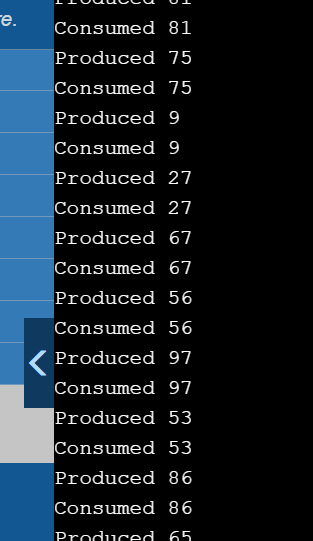
sem\_init(&empty,0,BUFFER\_SIZE);

sem\_init(&full,0,0);

pthread\_mutex\_init(&mutex,NULL);

pthread\_create(&producer,NULL,produce,NULL);

pthread\_create(&consumer,NULL,consume,NULL);

pthread\_exit(NULL); } 

**PRACTICAL-07**

**WAP to implement the Dining Philosopher’s Problem.**

#include<iostream>

#define n 4

using namespace std;

int compltedPhilo = 0,i;

struct fork{

int taken;

}ForkAvil[n];

struct philosp{

int left;

int right;

}Philostatus[n];

void goForDinner(int philID){ //same like threads concept here cases implemented

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

cout<<"Philosopher "<<philID+1<<" completed his dinner\n";

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

cout<<"Philosopher "<<philID+1<<" completed his dinner\n";

Philostatus[philID].left = Philostatus[philID].right = 10; //remembering that he completed dinner by assigning value 10

int otherFork = philID-1;

if(otherFork== -1)

otherFork=(n-1);

ForkAvil[philID].taken = ForkAvil[otherFork].taken = 0; //releasing forks

cout<<"Philosopher "<<philID+1<<" released fork "<<philID+1<<" and fork

"<<otherFork+1<<"\n";

compltedPhilo++;

}

else if(Philostatus[philID].left==1 && Philostatus[philID].right==0){ //left already taken, trying for right fork

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

if(ForkAvil[philID].taken==0){ //KEY POINT OF THIS PROBLEM, THAT LAST PHILOSOPHER TRYING IN reverse DIRECTION

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

cout<<"Fork "<<philID+1<<" taken by philosopher "<<philID+1<<"\n";

}else{

cout<<"Philosopher "<<philID+1<<" is waiting for fork "<<philID+1<<"\n";

}

}else{

int dupphilID = philID;

philID-=1;

if(philID== -1)

philID=(n-1);

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

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

cout<<"Fork "<<philID+1<<" taken by Philosopher "<<dupphilID+1<<"\n";

}else{

cout<<"Philosopher "<<dupphilID+1<<" is waiting for Fork "<<philID+1<<"\n";

}

}

}

else if(Philostatus[philID].left==0){ //nothing taken yet

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

if(ForkAvil[philID-1].taken==0){ //KEY POINT OF THIS PROBLEM, THAT LAST PHILOSOPHER TRYING IN reverse DIRECTION

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

cout<<"Fork "<<philID<<" taken by philosopher "<<philID+1<<"\n";

}else{

cout<<"Philosopher "<<philID+1<<" is waiting for fork "<<philID<<"\n";

}

}else{ //except last philosopher case

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

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

cout<<"Fork "<<philID+1<<" taken by Philosopher "<<philID+1<<"\n";

} else {

cout<<"Philosopher "<<philID+1<<" is waiting for Fork "<<philID+1<<"\n";

} }

}else{}

}

int main(){

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

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

while(compltedPhilo<n){

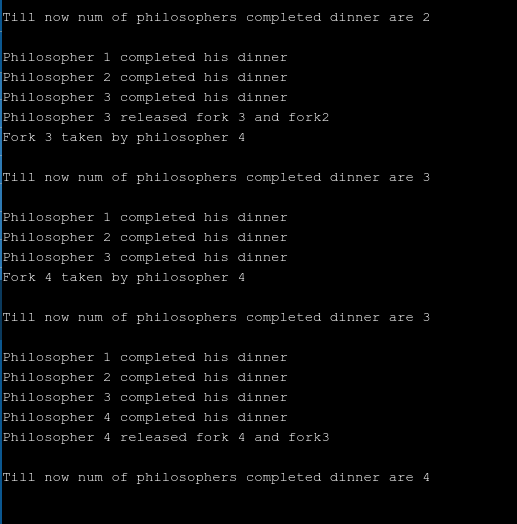
for(i=0;i<n;i++)

goForDinner(i);

cout<<"\nTill now num of philosophers completed dinner are "<<compltedPhilo<<"\n\n";

}

return 0; }

****

**PRACTICAL-08**

**WAP to implement the Banker’s Algorithm.**

#include<bits/stdc++.h>

using namespace std;

int allocation[10][3],need[10][3],Max[10][3],available[10][3];

int p,current[3];

bool executed[10],come;

void IMP(){

come=false;

for (int i = 0; i < 10; ++i)

{

executed[i]=false;

}

}

void Calculate()

{

IMP();

int i,j;

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

{

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

{

while(executed[j] && j<p-1)

{

j++;

}

if (need[j][0]<=current[0]&&need[j][1]<=current[1]&&need[j][2]<=current[2])

{

if (!executed[j])

{

executed[j]=true; current[0]+=allocation[j][0];current[1]+=allocation[j][1];current[2]+=allocation[j][2];

cout<<"\nProcess P"<<j+1;

cout<<"\nCurrent: "<<current[0]<<" "<<current[1]<<" "<<current[2]<<"\n";

cout<<"\nProcess executed without deadlock";

come=true;

break;

}

}

}

if (!come)

{

cout<<"\n Dead lock\n\n";

break;

}else{

come=false;

}

}

}

int main ()

{

int keepon = 1;

cout<<"Enter No. of processes: ";

cin>>p;

cout<<"\n";

cout<<"Enter the current resources: ";

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

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

{

cout<<"\n\n Process P"<<i+1<<" Details\n";

cout<<"Enter Allocation : ";

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

cout<<"Enter Max :";

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

need[i][0]=Max[i][0]-allocation[i][0];need[i][1]=Max[i][1]-

allocation[i][1];need[i][2]=Max[i][2]-allocation[i][2];

}

cout<<"\n\n Table for Bankers Algo\n\n";

cout<<"Initial Resources: "<<current[0]<<" "<<current[1]<<" "<<current[2]<<"\n\n";

cout<<"Process Max Allocation Need\n";

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

{

cout<<" P"<<i+1<<" ";

cout<<" "<<Max[i][0]<<" "<<Max[i][1]<<" "<<Max[i][2]<<" ";

cout<<" "<<allocation[i][0]<<" "<<allocation[i][1]<<" "<<allocation[i][2]<<" ";

cout<<" "<<need[i][0]<<" "<<need[i][1]<<" "<<need[i][2];

cout<<"\n";

}

cout<<"\n\n";

Calculate();

while(keepon)

{

int val,pro;

cout<<"\n\nSelect Below oprations:\n\n";

cout<<"1.Change Max of process: \n";

cout<<"2.Change Allocation of process\n";

cout<<"3.Change Initial Resources\n";

cout<<"4.Exit\n\n";

cin>>val;

if (val==1)

{

cout<<"\n\nEnter Process No: ";

cin>>pro;

cout<<"\nEnter New Max: ";

cin>>Max[pro-1][0]>>Max[pro-1][1]>>Max[pro-1][2];

}

else if (val==2)

{

cout<<"\n\nEnter Process No: ";

cin>>pro;

cout<<"\nEnter New Allocation: ";

cin>>allocation[pro-1][0]>>allocation[pro-1][1]>>allocation[pro-1][2];

}

else if (val==3)

{

cout<<"\nEnter Initial Resources: ";

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

}

Else

{

break;

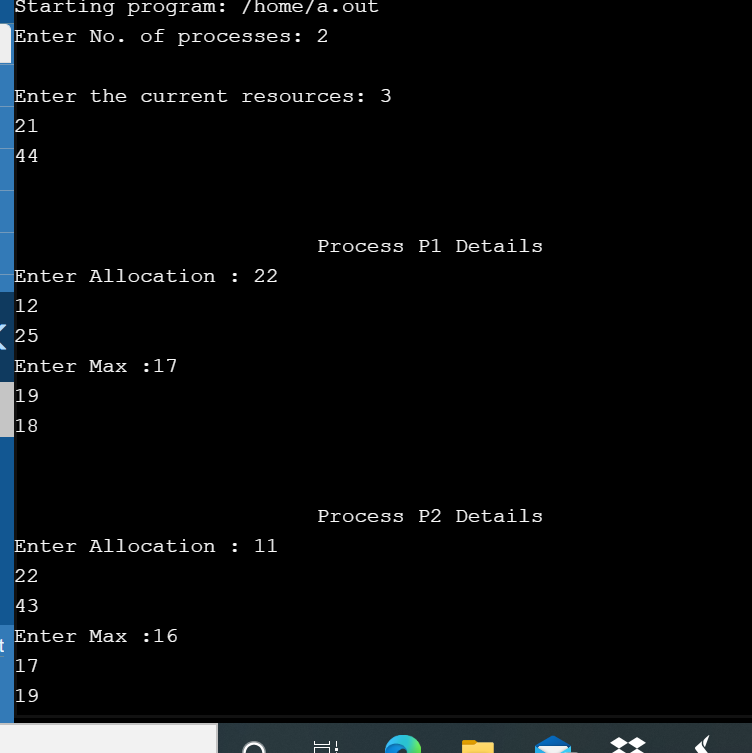
}

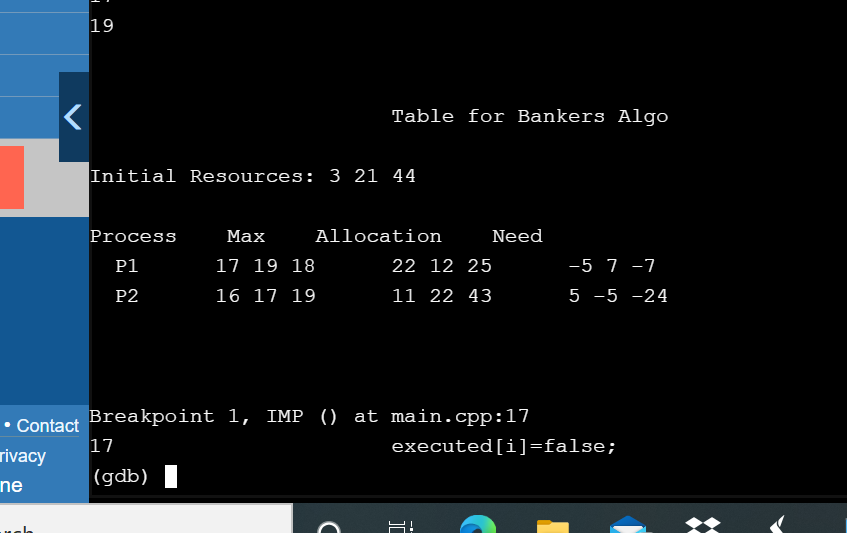
Calculate();

}

return 0;

}





**PRACTICAL-09**

**WAP to implement the concept of First Fit memory allocation.**

#include<iostream>

using namespace std;

int main()

{

int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;

for(i = 0; i < 10; i++)

{

flags[i] = 0;

allocation[i] = -1;

}

cout<<"Enter no. of blocks: ";

cin>>bno;

cout<<"\nEnter size of each block: ";

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

cin>>bsize[i];

cout<<"\nEnter no. of processes: ";

cin>>pno;

cout<<"\nEnter size of each process: ";

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

cin>>psize[i];

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

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

if(flags[j] == 0 && bsize[j] >= psize[i])

{

allocation[j] = i;

flags[j] = 1;

break;

}

cout<<"\nBlock no.\tsize\t\tprocess no.\t\tsize";

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

{ cout<<"\n"<< i+1<<"\t\t"<<bsize[i]<<"\t\t";

if(flags[i] == 1)

cout<<allocation[i]+1<<"\t\t\t"<<psize[allocation[i]];

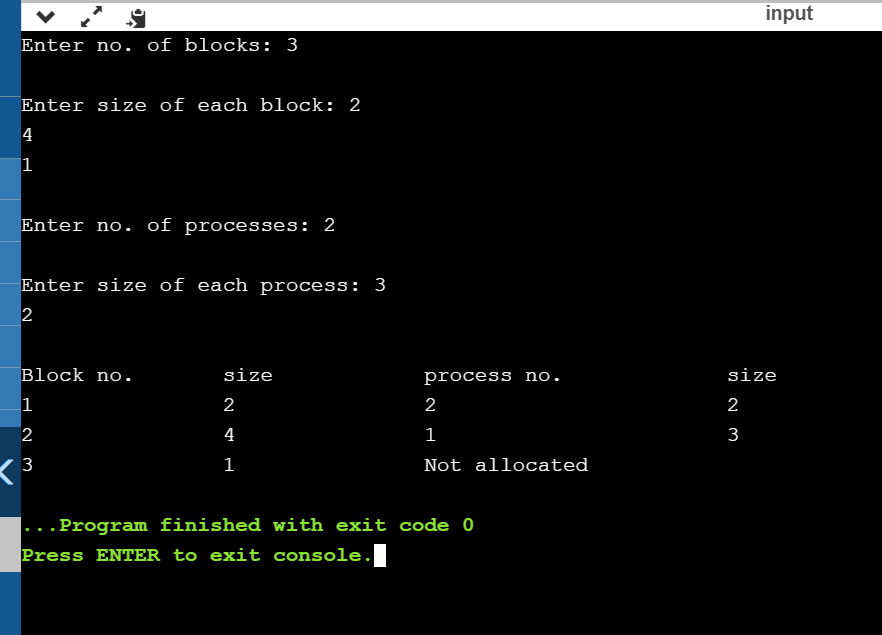
else

cout<<"Not allocated";

}

return 0;

}



**PRACTICAL-10**

**WAP to implement the concept of Best fit memory allocation.**

#include<iostream>

using namespace std;

int main()

{ int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;

static int barray[20],parray[20];

cout<<"\n\t\t\tMemory Management Scheme - Best Fit";

cout<<"\nEnter the number of blocks:";

cin>>nb;

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

cin>>np;

cout<<"\nEnter the size of the blocks:-\n";

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

{ cout<<"Block no."<<i<<":";

cin>>b[i]; }

cout<<"\nEnter the size of the processes :-\n";

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

{ cout<<"Process no. "<<i<<":";

cin>>p[i]; }

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

{ for(j=1;j<=nb;j++)

{ if(barray[j]!=1)

{ temp=b[j]-p[i];

if(temp>=0)

if(lowest>temp)

{ parray[i]=j;

lowest=temp; } } }

fragment[i]=lowest;

barray[parray[i]]=1;

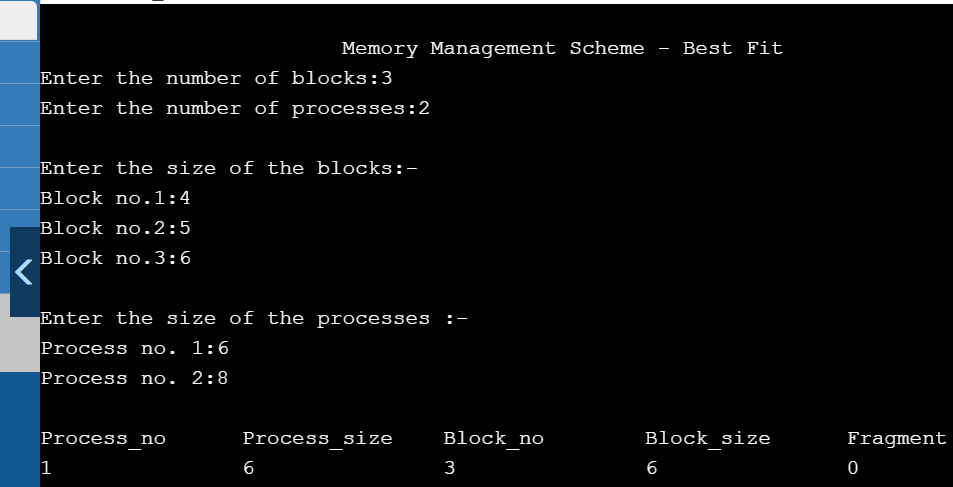
lowest=10000; }

cout<<"\nProcess\_no\tProcess\_size\tBlock\_no\tBlock\_size\tFragment";

for(i=1;i<=np && parray[i]!=0;i++)

cout<<"\n"<<i<<"\t\t"<<p[i]<<"\t\t"<<parray[i]<<"\t\t"<<b[parray[i]]<<"\t\t"<<fragment[i];

return 0; }



**PRACTICAL-11**

**WAP to implement the concept of Worst Fit memory allocation.**

#include<bits/stdc++.h>

using namespace std;

void worstFit(int blockSize[], int m, int processSize[], int n)

{

int allocation[n];

memset(allocation, -1, sizeof(allocation));

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

{

int wstIdx = -1;

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

{

if (blockSize[j] >= processSize[i])

{

if (wstIdx == -1)

wstIdx = j;

else if (blockSize[wstIdx] < blockSize[j])

wstIdx = j;

} }

if (wstIdx != -1)

{ allocation[i] = wstIdx;

blockSize[wstIdx] -= processSize[i]; } }

cout << "\nProcess No.\tProcess Size\tBlock no.\n";

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

{

cout << " " << i+1 << "\t\t" << processSize[i] << "\t\t";

if (allocation[i] != -1)

cout << allocation[i] + 1;

else

cout << "Not Allocated";

cout << endl;

} }

int main()

{

int blockSize[] = {100, 500, 200, 300, 600};

int processSize[] = {212, 417, 112, 426};

int m = sizeof(blockSize)/sizeof(blockSize[0]);

int n = sizeof(processSize)/sizeof(processSize[0]);

worstFit(blockSize, m, processSize, n);

return 0 ;

}



**PRACTICAL-12**

**WAP a program to implement FIFO Page Replacement algorithm.**

#include<bits/stdc++.h>

using namespace std;

int pageFaults(int pages[], int n, int capacity)

{

unordered\_set<int> s;

queue<int> indexes;

int page\_faults = 0;

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

{

if (s.size() < capacity)

{

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

{

s.insert(pages[i]);

page\_faults++;

indexes.push(pages[i]);

}

}

else

{

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

{

int val = indexes.front();

indexes.pop();

s.erase(val);

s.insert(pages[i]);

indexes.push(pages[i]);

page\_faults++;

}

}

}

return page\_faults;

}

int main()

{

int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

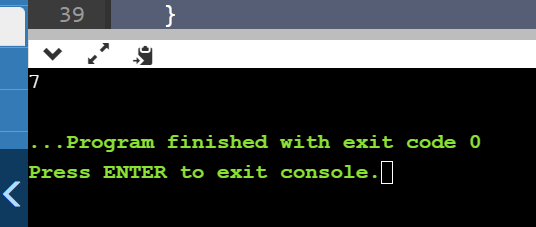
int n = sizeof(pages)/sizeof(pages[0]);

int capacity = 4;

cout << pageFaults(pages, n, capacity);

return 0;

}



**PRACTICAL-13**

**WAP a program to implement LRU Page Replacement algorithms.**

#include<bits/stdc++.h>

using namespace std;

int pageFaults(int pages[], int n, int capacity)

{

unordered\_set<int> s;

unordered\_map<int, int> indexes;

int page\_faults = 0;

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

{

if (s.size() < capacity)

{

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

{

s.insert(pages[i]);

page\_faults++;

}

indexes[pages[i]] = i;

}

else

{

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

{

int lru = INT\_MAX, val;

for (auto it=s.begin(); it!=s.end(); it++)

{ if (indexes[\*it] < lru)

{ lru = indexes[\*it];

val = \*it;

} }

s.erase(val);

s.insert(pages[i]);

page\_faults++;

}

indexes[pages[i]] = i;

} }

return page\_faults;

}

int main()

{

int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

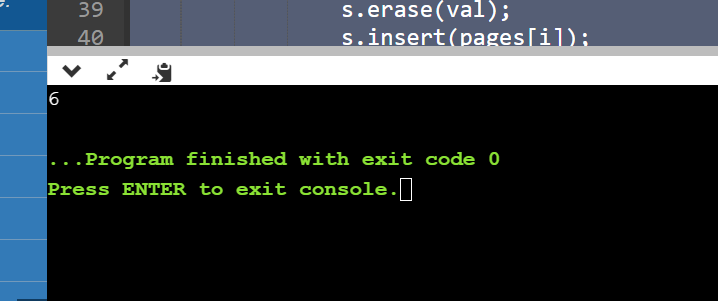
int n = sizeof(pages)/sizeof(pages[0]);

int capacity = 4;

cout << pageFaults(pages, n, capacity);

return 0;

}



**PRACTICAL-14**

**WAP a program to implement Optimal Page Replacement algorithm.**

#include <bits/stdc++.h>

using namespace std;

bool search(int key, vector<int>& fr)

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

if (fr[i] == key)

return true;

return false; }

int predict(int pg[], vector<int>& fr, int pn, int index)

{ int res = -1, farthest = index;

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

int j;

for (j = index; j < pn; j++) {

if (fr[i] == pg[j]) {

if (j > farthest) {

farthest = j;

res = i; }

break; } }

if (j == pn)

return i; }

return (res == -1) ? 0 : res; }

void optimalPage(int pg[], int pn, int fn)

{

vector<int> fr;

int hit = 0;

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

if (search(pg[i], fr)) {

hit++;

continue; }

if (fr.size() < fn)

fr.push\_back(pg[i]);

else {

int j = predict(pg, fr, pn, i + 1);

fr[j] = pg[i]; } }

cout << "No. of hits = " << hit << endl;

cout << "No. of misses = " << pn - hit << endl;

}

int main()

{

int pg[] = { 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 };

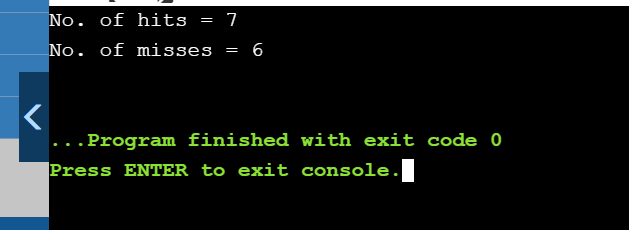
int pn = sizeof(pg) / sizeof(pg[0]);

int fn = 4;

optimalPage(pg, pn, fn);

return 0;

}



**PRACTICAL-15**

**WAP to implement FCFS Disk Scheduling.**

#include <bits/stdc++.h>

using namespace std;

int size = 8;

void FCFS(int arr[], int head)

{

int seek\_count = 0;

int distance, cur\_track;

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

cur\_track = arr[i];

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

}

cout << "Total number of seek operations = "

<< seek\_count << endl;

cout << "Seek Sequence is" << endl;

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

cout << arr[i] << endl;

}

}

int main()

{

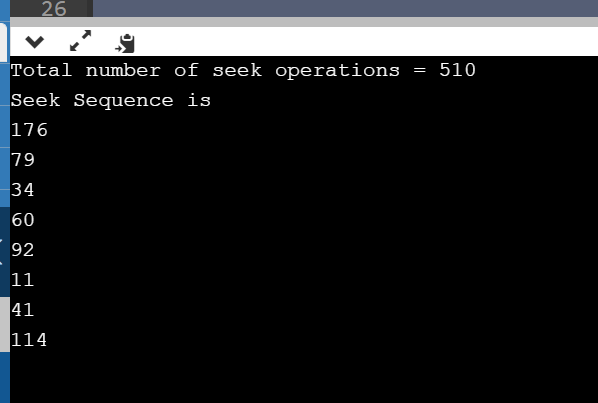
int arr[size] = { 176, 79, 34, 60, 92, 11, 41, 114 };

int head = 50;

FCFS(arr, head);

return 0;

}



**PRACTICAL-16**

**WAP to implement SCAN Disk Scheduling.**

#include <bits/stdc++.h>

using namespace std;

int size = 8;

int disk\_size = 200;

void SCAN(int arr[], int head, string direction)

{

int seek\_count = 0;

int distance, cur\_track;

vector<int> left, right;

vector<int> seek\_sequence;

if (direction == "left")

left.push\_back(0);

else if (direction == "right")

right.push\_back(disk\_size - 1);

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

if (arr[i] < head)

left.push\_back(arr[i]);

if (arr[i] > head)

right.push\_back(arr[i]);

}

std::sort(left.begin(), left.end());

std::sort(right.begin(), right.end());

int run = 2;

while (run--) {

if (direction == "left") {

for (int i = left.size() - 1; i >= 0; i--) {

cur\_track = left[i];

seek\_sequence.push\_back(cur\_track);

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

}

direction = "right";

}

else if (direction == "right") {

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

cur\_track = right[i];

seek\_sequence.push\_back(cur\_track);

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

}

direction = "left";

}

}

cout << "Total number of seek operations = "

<< seek\_count << endl;

cout << "Seek Sequence is" << endl;

for (int i = 0; i < seek\_sequence.size(); i++) {

cout << seek\_sequence[i] << endl;

}

}

int main()

{

int arr[size] = { 176, 79, 34, 60,92, 11, 41, 114 };

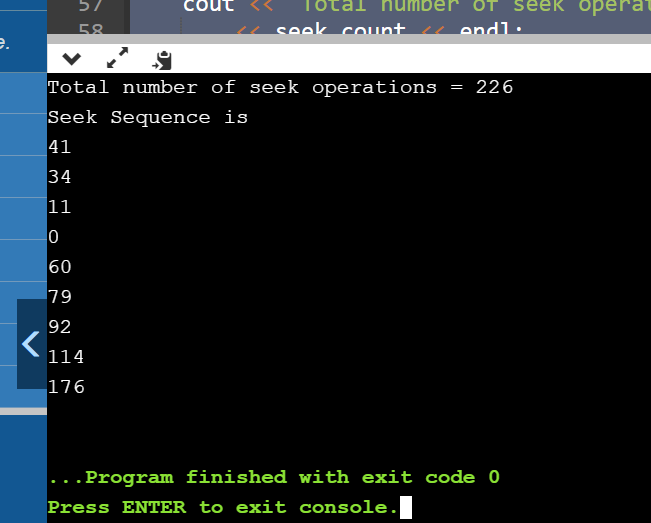
int head = 50;

string direction = "left";

SCAN(arr, head, direction);

return 0;

}



**PRACTICAL-17**

**WAP to implement C-SCAN Disk Scheduling.**

#include <bits/stdc++.h>

using namespace std;

int size = 8;

int disk\_size = 200;

void CSCAN(int arr[], int head)

{

int seek\_count = 0;

int distance, cur\_track;

vector<int> left, right;

vector<int> seek\_sequence;

left.push\_back(0);

right.push\_back(disk\_size - 1);

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

if (arr[i] < head)

left.push\_back(arr[i]);

if (arr[i] > head)

right.push\_back(arr[i]);

}

std::sort(left.begin(), left.end());

std::sort(right.begin(), right.end());

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

cur\_track = right[i];

seek\_sequence.push\_back(cur\_track);

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

}

head = 0;

seek\_count += (disk\_size - 1);

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

cur\_track = left[i];

seek\_sequence.push\_back(cur\_track);

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

}

cout << "Total number of seek operations = "

<< seek\_count << endl;

cout << "Seek Sequence is" << endl;

for (int i = 0; i < seek\_sequence.size(); i++) {

cout << seek\_sequence[i] << endl;

}

}

int main()

{

int arr[size] = { 176, 79, 34, 60, 92, 11, 41, 114 };

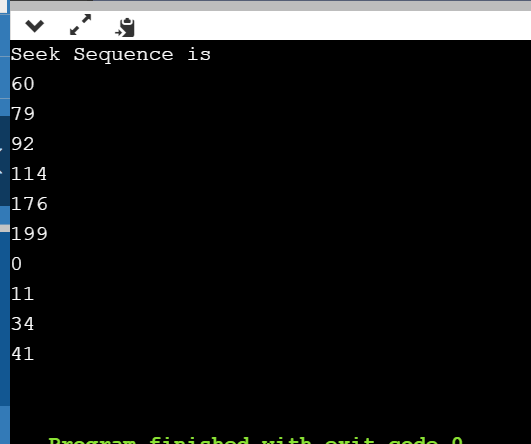
int head = 50;

cout << "Initial position of head: " << head << endl;

CSCAN(arr, head);

return 0;

}



**PRACTICAL-18**

**WAP to implement LOOK Disk Scheduling.**

#include <bits/stdc++.h>

using namespace std;

int size = 8;

int disk\_size = 200;

void LOOK(int arr[], int head, string direction)

{

int seek\_count = 0;

int distance, cur\_track;

vector<int> left, right;

vector<int> seek\_sequence;

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

if (arr[i] < head)

left.push\_back(arr[i]);

if (arr[i] > head)

right.push\_back(arr[i]);

}

std::sort(left.begin(), left.end());

std::sort(right.begin(), right.end());

int run = 2;

while (run--) {

if (direction == "left") {

for (int i = left.size() - 1; i >= 0; i--) {

cur\_track = left[i];

seek\_sequence.push\_back(cur\_track);

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

}

direction = "right";

}

else if (direction == "right") {

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

cur\_track = right[i];

seek\_sequence.push\_back(cur\_track);

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

}

direction = "left";

}

}

cout << "Total number of seek operations = "

<< seek\_count << endl;

cout << "Seek Sequence is" << endl;

for (int i = 0; i < seek\_sequence.size(); i++) {

cout << seek\_sequence[i] << endl;

}

}

int main()

{

int arr[size] = { 176, 79, 34, 60,92, 11, 41, 114 };

int head = 50;

string direction = "right";

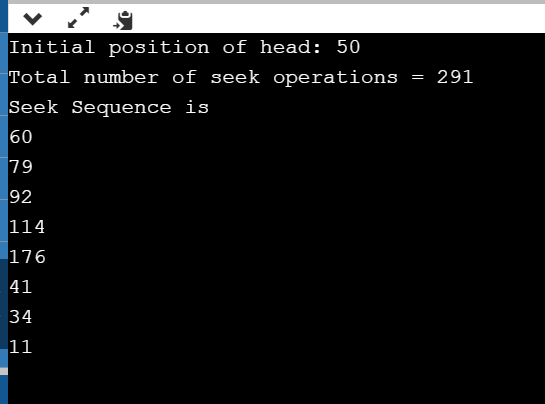
cout << "Initial position of head: "

<< head << endl;

LOOK(arr, head, direction);

return 0;

}



**PRACTICAL-19**

**WAP to implement C-LOOK Disk Scheduling.**

#include <bits/stdc++.h>

using namespace std;

int size = 8;

int disk\_size = 200;

void CLOOK(int arr[], int head)

{

int seek\_count = 0;

int distance, cur\_track;

vector<int> left, right;

vector<int> seek\_sequence;

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

if (arr[i] < head)

left.push\_back(arr[i]);

if (arr[i] > head)

right.push\_back(arr[i]);

}

std::sort(left.begin(), left.end());

std::sort(right.begin(), right.end());

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

cur\_track = right[i];

seek\_sequence.push\_back(cur\_track);

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

}

seek\_count += abs(head - left[0]);

head = left[0];

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

cur\_track = left[i];

seek\_sequence.push\_back(cur\_track);

distance = abs(cur\_track - head);

seek\_count += distance;

head = cur\_track;

}

cout << "Total number of seek operations = " << seek\_count << endl;

cout << "Seek Sequence is" << endl;

for (int i = 0; i < seek\_sequence.size(); i++) {

cout << seek\_sequence[i] << endl;

}

}

int main()

{

int arr[size] = { 176, 79, 34, 60, 92, 11, 41, 114 };

int head = 50;

cout << "Initial position of head: " << head << endl;

CLOOK(arr, head);

return 0;

}

