

## **File handling utilities:**

**mkdir** : used to make directories

Syn: mkdir Directoryname

eg. mkdir D1

**cd** : changes directories

Syn: cd Directoryname

eg. cd D1

**cat**: this command is used to create /display files.

Syn: cat >filename.txt //to create a file

Eg: cat >file1.txt

```
#include<stdio.h>
```

```
void main()
```

```
{
```

```
printf("hi");
```

```
}
```

Syn:cat filename.txt //to display a file

Eg:cat file1.txt

```
#include<stdio.h>
```

```
void main()
```

```
{
```

```
printf("hi");
```

```
}
```

**gedit**:is used to create txt files;

Syn:gedit filename.txt

Eg:gedit file1.txt //a text file

**vim**: Vi Improved a programmers text editor

Syn: vim filename.txt

eg. vim file1.txt

To exit from vim

Press esc and !wq

**ls** : Displays list of all system files

Syn: ls

Display list of directory contents

Syn: ls Directoryname

eg. ls D1

**pwd**: print name of current working directory

Syn: pwd

**cp** :copy files and directories

syn: cp sourcefile destinationfile

eg. cp file1.txt file2.txt

**mv** : move (rename) files

syn: mv sourcefilename destinationfilename

eg. mv file1.txt file2.txt

To rename files

syn: mv oldfilename.txt newfilename.txt

**rm**: remove files

Syn: rm filename

eg. rm file1.txt

**find**:search for files in a directory hierarchy

Syn: find path pattern

eg. find file1.txt, find name

file1.txt

**history**:prints recently used commands

Syn: history

**Security filespermissions:**

**chmod**:change file access permissions

Syn: chmod mode filename

eg. chmod 744 file1.txt

**chown**: change file owner and group

Syn: chown owner/group filename.

eg. chown student1 file1.txt

**who:** show who is logged on

Syn: who

**Process utilities:**

**ps** : report a snapshot of the current processes

Syn: ps

**Zip:** package and compress (archive) files

Syn: zip filename.zip filename.txt

eg. zip file1.zip file1.txt

**unzip:** list, test and extract compressed files in a ZIP archive

Syn: unzip filename

eg. unzip original.zip

## **Disk utilities:**

**Du** :used to estimate file space usage

**Du -a**:display an entry for each file contained in the current directory

**Df** : used to check free disk space

**Syn: du**

**df -h or df -k** :command is used to list free disk space

Syn:df -h/df -k

## **Networking commands:**

**hostname**:displays the machines host name

**hostname -d**:displays the domain name the machine belongs to

**hostname -f**: displays the fully qualified host and domain name

**hostname -i**:displays the IP address for the current machine

**ping**: It sends packets of information to the user-defined source. If the packets are received, the destination device sends packets back.

Eg:If you do ping www.yahoo.com it will display its IP address. Use ctrl+C to stop the test.

**Netstat**: used to find out all the multicast groups

**netstat -a or netstat -all** :will display all connections including TCP and UDP

**netstat --tcp or netstat -t**:will display only TCP connection

**netstat --udp or netstat -u**:will display only UDP connection

**nslookup**:To find all the IP addresses for a given domain name.

Eg. nslookup blogger.com

## **finger**

View user information, displays a user's login name, real name, terminal name and write status.

## **telnet**

Connects destination host via telnet protocol, if telnet connection establish on any port means connectivity between two hosts is working fine.

Syn:telnet

## **Filters:**

**Head:** command displays the top of the file, when used without any option it will display first 10lines of the file.

Syn:head filename.txt

Ex:head sample1.txt

/\*display first 10 lines\*/

\$ head -n 20 sample1.txt

/\* will display first 20 lines\*/



**tail**:command displays the end of the file. By default it will display last 10 lines of thefile.

Syn:tail filename.txt

Eg:\$ tail sample1.txt

/\*display last 10 lines\*/

Eg:\$ tail -n 15 sample1.txt

/\* will display last 15 lines \*/

**cut** : cut command can be used to cut the columns from a file with -c option

Eg:cat file1.txt

```
#includt<stdio.h>
```

```
voidmain()
```

```
{
```

```
printf(“hello”);
```

```
}
```

```
$ cut -c 1,2,3file1.txt
```

```
#in
```

```
voi
```

```
{
```

```
pri
```

```
}
```

**grep** :used to searching for a pattern

Syn:grep pattern filename.txt

Eg:grep main file1.txt //prints the line that containing  
main

```
main()
```

**grep -c**:counts number of line that have given pattern.

Syn:grep -c pattern filename.txt

Eg:grep -c main file1.txt

1

**grep -i**:prints the line that containing main by ignoring case sensitive.

Syn:grep -i pattern filename.txt

Eg:grep main file1.txt //prints the line that containing main

main()

**b)NAME OF THE EXPERIMENT:**Linux commands cp , mv using Linux system calls

AIM:Implement the Linux commands (a) cp (b) mv using Linux system calls.

ALGORITHM:

Step 1:Start

Step 2: open Terminal

Step 3:give the cp and mv command and check with the different type of the options

Step 4:observe the result of the each options

Step 5: Stop

## **Program:**

cp :copy files and directories

Usage: cp [OPTION]... SOURCE DEST

eg. cp sample.txt sample\_copy.txt

cp sample\_copy.txt target\_dir

mv : move (rename) files

Usage: mv [OPTION]... SOURCE DEST

eg. mv source.txt target\_dir

mv old.txt new.txt

## **VIVA-VOCE**

1. How would you kill a process?

The killall command is used to kill processes by name.

2. What are the different file types available ?

In Linux, everything is considered as a file. In UNIX, seven standard file types are regular,

directory, symbolic link, FIFO special, block special, character special, and socket.

### 3. Explain the method of changing file access permission?

To change the file or the directory permissions, you use the `chmod` (change mode) command.

There are two ways to use `chmod` — the symbolic mode and the absolute mode.

### 4. Which are the Linux Directory Commands?

DirectoryCommand	Description
<code>Pwd</code> :	The <code>pwd</code> command stands for (print working directory). It displays the current working location or directory of the user. It displays the whole working path starting with <code>/</code> . It is a built-in command.
<code>ls</code> :	The <code>ls</code> command is used to show the list of a folder. It will list out all the files in the directed folder.
<code>cd</code> :	The <code>cd</code> command stands for (change directory). It is used to change to the directory you want to work from the present directory.

`mkdir`: With `mkdir` command you can create your own directory.

`rmdir`: The `rmdir` command is used to remove a directory from your system.

### 5. How to rename a file in Linux using `cp` command?

If we want to rename and copy at the same time, then we use the following command.

```
cp program3.cpp homework6.cpp
```

### 6. How would you delete a directory in Linux

1. To remove an empty directory, use either `rmdir` or `rm -d` followed by the

directory name: `rm -d dirname rmdir dirname.`

2. To remove non-empty directories and all the files within them, use the `rm` command with the `-r` (recursive) option: `rm -r dirname.`

**WEEK 9: NAME OF THE EXPERIMENT: SHELL SCRIPT**

AIM:a) Write a shell script to find factorial of a given integer.

ALGORITHM:

Step 1:Start

Step 2:Read any number to find factorial

Step 3: initialize fact=1 and i=1

Step 4:while i less than do

fact=fact\*i

i=i+1

Step 4:print fact

Step 5: stop

**Source code :**

```
echo enter a number
```

```
read a
```

```
i=2
```

```
fact=1
```

```
if [ $a -ge 2 ]
```

```
then
```

```
while [ $i -le $a ]
```

```
do
```

```
fact=`expr $fact \* $i`
```

```
i=`expr $i + 1`
```

```
done
```

```
fi
```

```
echo factorial of $a=$fact
```

output:

```
[latha@localhost ~]$ sh fact.sh
```

enter a number

5

factorial of 5=120

**b)NAME OF THE EXPERIMENT:Creating a child process and allow the parent to display ‘parent’.**

AIM:Write a C program to create a child process and allow the parent to display ‘parent’

And the child to display ‘child’ on the screen.

**ALGORITHM:**

Step 1: Start the main function

Step 2: call the fork() function to create a child process fork function returns 2 values

Step 3:which returns 0 to child process

Step 4:which returns process id to the parent process

Step 5:stop

**Source Code:**

```
Include<stdio.h>
```

```
#include<unistd.h>
```

```
#include<stdlib.h>
```

```
#include<string.h>
```

```
int global=10;
```

```
int main()
```

```
{
```

```

int local=20;

pid_t pid;

printf("before fork\n");

printf("pid=%d,global=%d,local=%d\n",getpid(),global,local);

pid=fork();

if(pid<0)

printf("failed to create child");

else if(pid==0)

{printf("after fork\n");

global++;

local++;

}

else if(pid>0)

sleep(2);

printf("cid=%d,global=%d,local=%d\n",getpid(),global,local);

exit(0);

}

```

### **Output:**

```
[latha@localhost ~]$ gcc week16.c
```

```
[latha@localhost ~]$ ./a.out
```

```
before fork
```

```
pid=3005,global=10,local=20
```

```
after fork
```

```
cid=3006,global=11,local=21
```

```
cid=3005,global=10,local=20
```



c)

**NAME OF THE EXPERIMENT: Parent writes a message to a pipe and the child reads**

**The message.**

AIM: Write a C program in which a parent writes a message to a pipe and the child reads the message.

SOURCE CODE:

```
#include <stdio.h>

#include <stdlib.h>

#include <sys/wait.h> /* contains prototype for wait */

int main(void)

{

int pid;

int status;

printf(&quot;Hello World!\n&quot;);

pid = fork( );

if(pid == -1) /* check for error in fork */

{

perror(&quot;bad fork&quot;);

exit(1);

}

if (pid == 0)

printf(&quot;I am the child process.\n&quot;);

else {

wait(&status); /* parent waits for child to finish */

printf(&quot;I am the parent process.&quot;);
```

```
return 0;
```

```
}
```

```
}
```

OUTPUT:

```
student@202.sys14:~$ ./a.out
```

Hello World!

I am the child process

I am the parent process.

VIVA-VOCE

1. What is Shell Script?

A Shell Script is a text file that contains one or more commands.

2. What are the different type of variables used in a shell Script?

In Linux shell script we can use two types of variables :

- o System defined variables

- o User defined variables

3. Why are we using fork() function call?

System call fork() is used to create processes. It takes no arguments and returns a process ID. The purpose of fork() is to create a new process, which becomes the child process of the caller. After a new child process is created, both processes will execute the next instruction following the fork() system call.

4. Why do we create a child process?

Sometimes there is a need for a program to perform more than one function simultaneously. Since these jobs may be interrelated so two different programs to perform them cannot be created.

5. Why do we use piping?

A pipe is a form of redirection (transfer of standard output to some other destination) that is used in Linux and other Unix-like operating systems to send the output of one command/program/process to another command/program/process for further processing.

6. what is IPC?

Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them.

### **Week-10:**

#### **NAME OF THE EXPERIMENT:CPU Scheduling Techniques FCFS , Priority**

AIM:Write C Programs to simulate the following CPU scheduling algorithms:

a)FCFS b) Priority

#### **SOURCE CODE:**

```
#include<stdio.h>

void main()
{
int pid[10],bt[10],wt[10],tat[10],n,twt=0,ttat=0,i;

float awt,atat;

printf("Enter no.of processes");

scanf("%d",&n);

printf("\n Enter burst times:");

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

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

wt[0]=0;

tat[0]=bt[0];
```

```

for(i=1;i<n;i++)
{
wt[i]=tat[i-1];
tat[i]=bt[i]+wt[i];
}
for(i=0;i<n;i++)
{
ttat= ttat+tat[i];
twt=twt+wt[i];
}
printf("\n PID \t BT \t WT \t TAT");
for(i=0;i<n;i++)
printf("\n %d\t%d\t%d\t%d",i+1,bt[i],wt[i],tat[i]);
awt=(float)twt/n;
atat=(float)ttat/n;
printf("\nAvg. Waiting Time=%f",awt);
printf("\nAvg. Turn around time=%f",atat);
}

```

### **Output:**

Enter no.of processes:3

Enter burst times:4

13

25

Enter pid 1 2 3

PID	BT	WT	TAT
1	4	0	4
2	13	4	17
3	25	17	42

Avg.waiting time=7.00000

Avg turnaround time=21.00000

Process returned 32(0X2) execution time=20.112s

Pres any key to continue.

#### **b) PRIORITY ALGORITHM:**

##### **SOURCE CODE:**

```
#include<stdio.h>
```

```
void main()
```

```
{
```

```
int pid[10],bt[10],pr[10],wt[10],tat[10],n,twt=0,ttat=0,i,j,t;
```

```
float awt,atat;
```

```
printf("Enter no.of processes:");
```

```
scanf("%d",&n);
```

```
printf("\n Enter burst times:");
```

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

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

```
printf("\n Enter PID:");
```

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

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

```
printf("\n Enter Priorities:");
```

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

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

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

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

```
if(pr[i]>pr[j]){
```

```
t=pr[i];
```

```
pr[i]=pr[j];
```

```
pr[j]=t;
```

```
t=bt[i];
```

```
bt[i]=bt[j];
```

```
bt[j]=t;
```

```
t=pid[i];
```

```
pid[i]=pid[j];
```

```
pid[j]=t;
```

```
}}}
```

```
wt[0]=0;
```

```
tat[0]=bt[0];
```

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

```

wt[i]=tat[i-1];

tat[i]=bt[i]+wt[i];

}

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

ttat= ttat+tat[i];

twt=twt+wt[i];

}

printf("\n PID PRIORITY \t BT \t WT \t TAT");

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

printf("\n %d\t%d\t%d\t%d\t%d",pid[i],pr[i],bt[i],wt[i],tat[i]);

awt=(float)twt/n;

atat=(float)ttat/n;

printf("\nAvg. Waiting Time=%f",awt);

printf("\nAvg. Turn around time=%f",atat);

```

### Output:

```

Enter  noof Processes :4
Enter  Burst Times:2
6
4
5
Enter Priorities :3
2
6
1
PID      PRIORITY  BT    WT    TT
5         1         5     0     5
6         2         6     5    11
3         3         2    11    13
4         6         4     3    17

```

```
Average Waiting Time: 7.250000  
AvG Turnaround Time: 11.500000  
Process returned 32 (0X20) execution time:86.108s
```

## **VIVA-VOCE**

### **1.What is an Operating system?**

**Operating System (OS), program that manages a computer's resources, especially the allocation of those resources among other programs. Typical resources include the central processing unit (CPU), computer memory, file storage, input/output (I/O) devices, and network connections.**

### **2.What is a process ?**

**A process is an 'active' entity, instead of a program, which is considered a 'passive' entity. A single program can create many processes when run multiple times; for example, when we open a .exe or binary file multiple times, multiple instances begin (multiple processes are created)**

### **3.What Are The Advantages of A Multiprocessor System?**

**The advantages of the multiprocessing system are: Increased Throughput – By increasing the number of processors, more work can be completed in a unit time. Cost Saving – Parallel system shares the memory, buses, peripherals etc. Multiprocessor system thus saves money as compared to multiple single systems.**

### **4. Explain starvation and Aging**

**Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time. In heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.**



Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time. For example, if priority range from 127(low) to 0(high), we could increase the priority of a waiting process by 1 Every 15 minutes.

#### **5.What are the functions of operating system?**

An operating system has three main functions: (1) manage the computer's resources, such as the central processing unit, memory, disk drives, and printers, (2) establish a user interface, and (3) execute and provide services for applications software.

### **Week-11:**

**NAME OF THE EXPERIMENT:CPU Scheduling Techniques SJF, Round Robin**

**AIM:Write C Programs to simulate the following CPU scheduling algorithms:**

**a)SJF   b) Round Robin**

**a) SJF ALGORITHM:**

**SOURCE CODE:**

```
#include<stdio.h>

void main(){

int pid[10],bt[10],wt[10],tat[10],n,twt=0,ttat=0,i,j,t;

float awt,atat;

printf("Enter no.of processes:");

scanf("%d",&n);

printf("\n Enter burst times:");
```

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

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

printf("\n Enter PID:");

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

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

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

{

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

{

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

{

t=bt[i];

bt[i]=bt[j];

bt[j]=t;

t=pid[i];

pid[i]=pid[j];

pid[j]=t;

}}}

wt[0]=0;

tat[0]=bt[0];
```

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

{

wt[i]=tat[i-1];

tat[i]=bt[i]+wt[i];

}

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

{

ttat= ttat+tat[i];

twt=twt+wt[i];

}

printf("\n PID \t BT \t WT \t TAT");

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

printf("\n %d\t%d\t%d\t%d",pid[i],bt[i],wt[i],tat[i]);

awt=(float)twt/n;

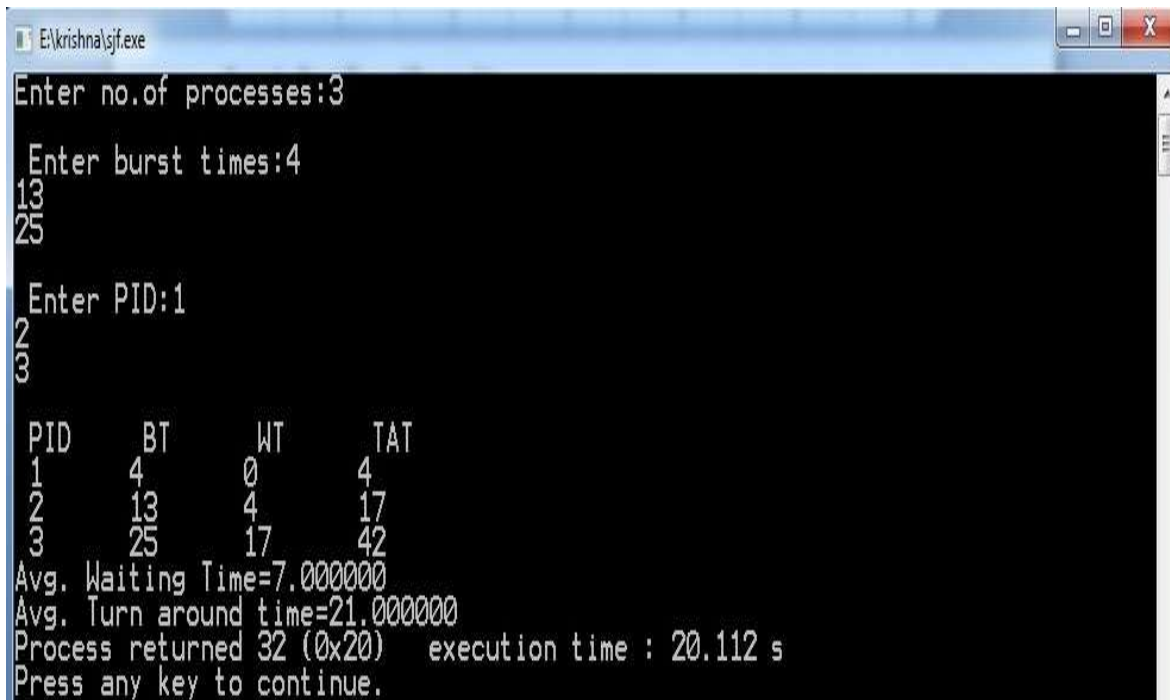
atat=(float)ttat/n;

printf("\nAvg. Waiting Time=%f",awt);

printf("\nAvg. Turn around time=%f",atat);

}
```

## OUTPUT :



```
E:\krishna\sjf.exe
Enter no.of processes:3
Enter burst times:4
13
25
Enter PID:1
2
3
PID    BT    WT    TAT
1       4     0     4
2       13    4     17
3       25    17    42
Avg. Waiting Time=7.000000
Avg. Turn around time=21.000000
Process returned 32 (0x20)   execution time : 20.112 s
Press any key to continue.
```

## b) ROUND ROBIN ALGORITHM

### SOURCE CODE:

```
#include<stdio.h>
```

```
void main()
```

```
{
```

```
int ts, bt[10], wt[10], tat[10], i, j=0, n, bt[10], ttat=0, twt=0, tot=0;
```

```
float awt, atat;
```

```
printf("Enter the number of Processes \n");
```

```
scanf("%d", &n);
```

```
printf("\n Enter the Timeslice \n");
```

```
scanf("%d",&ts);

printf("\n Enter the Burst Time for the process");

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

scanf("%d",&bt1[i]);

bt[i]=bt1[i];

}

while(j<n){

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

if(bt[i]>0){

if(bt[i]>=ts){

tot+=ts;

bt[i]=bt[i]-ts;

if(bt[i]==0){

j++;

tat[i]=tot;

}}

else{

tot+=bt[i];

bt[i]=0;

j++;
```

```

tat[i]=tot;

}}}

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

wt[i]=tat[i]-bt1[i];

twt=twt+wt[i];

ttat=ttat+tat[i];

}

awt=(float)twt/n;

atat=(float)ttat/n;

printf("\n PID \t BT \t WT \t TAT\n");

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

printf("\n %d \t %d \t %d \t %d \t\n",i,bt1[i],wt[i],tat[i]);

}

printf("\n The average Waiting Time=%f",awt);

printf("\n The average Turn around Time=%f",atat);

}

```

**OUTPUT :**

```
E:\krishna\rr.exe
Enter the number of Processes
4
Enter the Timeslice
5
Enter the Burst Time for the process10
15
20
25
PID    BT    WT    TAT
1      10    15    25
2      15    30    45
3      20    40    60
4      25    45    70
The average Waiting Time=32.500000
The average Turn around Time=50.000000
Process returned 40 (0x28)    execution time : 13.199 s
Press any key to continue.
```

## VIVA –VOCE

### 1. List different CPU Scheduling algorithms.

The different CPU algorithms are:

- First Come First Serve.
- Shortest Job First.
- Shortest Remaining Time First.
- Round Robin Scheduling.
- Priority Scheduling.
- Multilevel Queue Scheduling.
- Multilevel Feedback Queue Scheduling.

### 2. What is FCFS Scheduling?

**First Come First Serve (FCFS) is an operating system scheduling algorithm that automatically executes queued requests and processes in order of their arrival. It is the easiest and simplest CPU scheduling algorithm. In this type of algorithm, processes which requests the CPU first get the CPU allocation first.**

### **3. What is SJF Scheduling?**

**Shortest Job First (SJF) is an algorithm in which the process having the smallest execution time is chosen for the next execution. This scheduling method can be preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution.**

### **4. What is RR Scheduling?**

**Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way. It is simple, easy to implement, and starvation-free as all processes get fair share of CPU. One of the most commonly used technique in CPU scheduling as a core.**

### **5. What is Priority Scheduling?**

**Priority Scheduling is a method of scheduling processes that is based on priority. In this algorithm, the scheduler selects the tasks to work as per the priority. The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis.**



## Week-12:

**NAME OF THE EXPERIMENT: File allocation techniques**

**AIM: Write C programs to simulate the following Fileallocation strategies**

**a) Sequential**

**b)Linked c)Indexed**

**a) Sequential Algorithm:**

**SOURCE CODE:**

```
#include<stdio.h>

main()
{
int n,i,j,b[20],sb[20],t[20],x,c[20][20];

printf("Enter no.of files:");

scanf("%d",&n);

for(i=0;i<n;i++)
{
printf("Enter no. of blocks occupied by file%d",i+1);

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

printf("Enter the starting block of file%d",i+1);

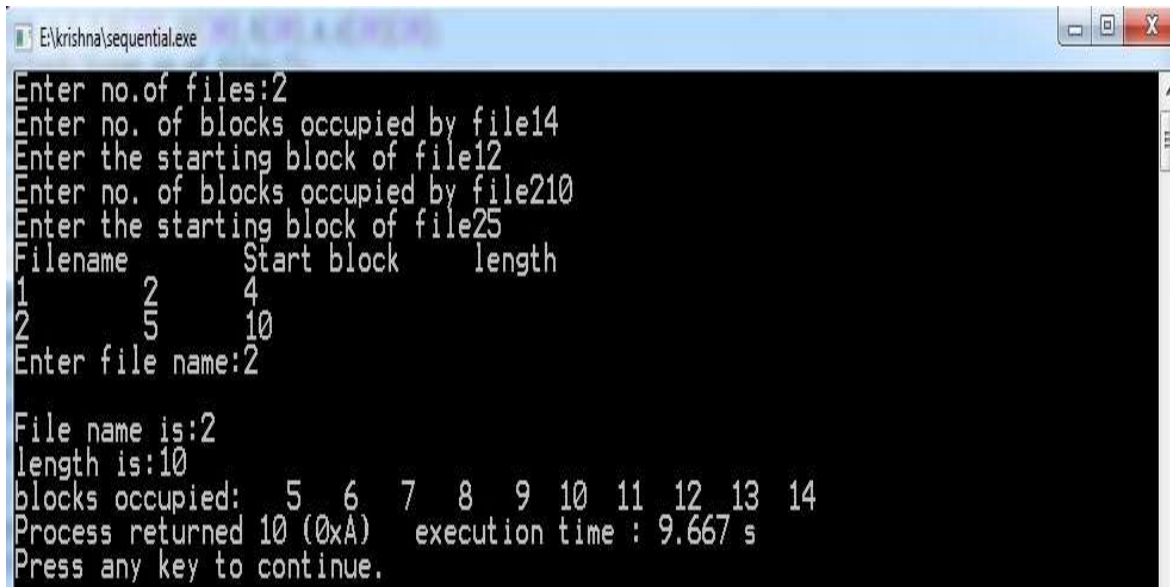
scanf("%d",&sb[i]);

t[i]=sb[i];

for(j=0;j<b[i];j++)
```

```
c[i][j]=sb[i]++;  
  
}  
  
printf("Filename\tStart block\tlength\n");  
  
for(i=0;i<n;i++)  
  
printf("%d\t %d \t%d\n",i+1,t[i],b[i]);  
  
printf("Enter file name:");  
  
scanf("%d",&x);  
  
printf("\nFile name is:%d",x);  
  
printf("\nlength is:%d",b[x-1]);  
  
printf("\nblocks occupied:");  
  
for(i=0;i<b[x-1];i++)  
  
printf("%4d",c[x-1][i]);  
  
}
```

### Output:



```
E:\krishna\sequential.exe
Enter no.of files:2
Enter no. of blocks occupied by file14
Enter the starting block of file12
Enter no. of blocks occupied by file210
Enter the starting block of file25
Filename      Start block    length
1            2            4
2            5            10
Enter file name:2
File name is:2
length is:10
blocks occupied: 5 6 7 8 9 10 11 12 13 14
Process returned 10 (0xA)   execution time : 9.667 s
Press any key to continue.
```

### b) LinkedAlgorithm:

#### SOURCE CODE:

```
#include<stdio.h>
```

```
struct file
```

```
{
```

```
char fname[10];
```

```
int start,size,block[10];
```

```
}f[10];
```

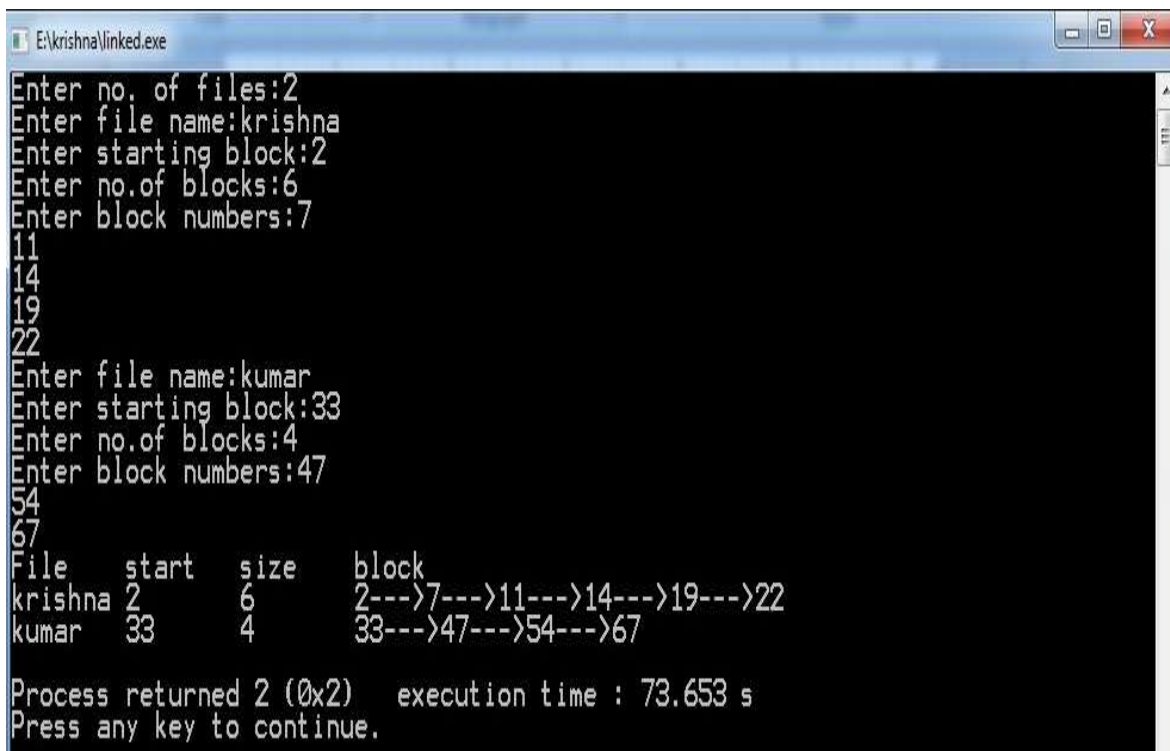
```
main()
```

```
{  
  
int i,j,n;  
  
printf("Enter no. of files:");  
  
scanf("%d",&n);  
  
for(i=0;i<n;i++){  
  
printf("Enter file name:");  
  
scanf("%s",&f[i].fname);  
  
printf("Enter starting block:");  
  
scanf("%d",&f[i].start);  
  
f[i].block[0]=f[i].start;  
  
printf("Enter no.of blocks:");  
  
scanf("%d",&f[i].size);  
  
printf("Enter block numbers:");  
  
for(j=1;j<f[i].size;j++){  
  
scanf("%d",&f[i].block[j]);  
  
}}  
  
printf("File\tstart\tsize\tblock\n");  
  
for(i=0;i<n;i++){  
  
printf("%s\t%d\t%d\t",f[i].fname,f[i].start,f[i].size);  
  
for(j=0;j<f[i].size-1;j++)
```

```
printf("%d--->",f[i].block[j]);
```

```
printf("%d\n",f[i].block[j]);
```

```
}}
```



```
E:\krishna\linked.exe
Enter no. of files:2
Enter file name:krishna
Enter starting block:2
Enter no.of blocks:6
Enter block numbers:7
11
14
19
22
Enter file name:kumar
Enter starting block:33
Enter no.of blocks:4
Enter block numbers:47
54
67
File      start   size   block
krishna  2         6      2--->7--->11--->14--->19--->22
kumar    33        4      33--->47--->54--->67

Process returned 2 (0x2)   execution time : 73.653 s
Press any key to continue.
```

### c) IndexedAlgorithm:

#### SOURCE CODE:

```
#include<stdio.h>
```

```
main()
```

```
{
```

```
int n,m[20],i,j,sb[20],b[20][20],x;
```

```
printf("\nEnter no. of files:");

scanf("%d",&n);

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

{

printf("\nEnter index block of file%d:",i+1);

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

printf("\nEnter length of file%d:",i+1);

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

printf("enter blocks of file%d:",i+1);

for(j=0;j<m[i];j++)

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

}

printf("\nFile\t Index\tLength\n");

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

{

printf("%d\t%d\t%d\n",i+1,sb[i],m[i]);

}

printf("\nEnter file name:");

scanf("%d",&x);

printf("\nfile name is:%d",x);
```

```
printf("\nIndex is:%d",sb[x-1]);
```

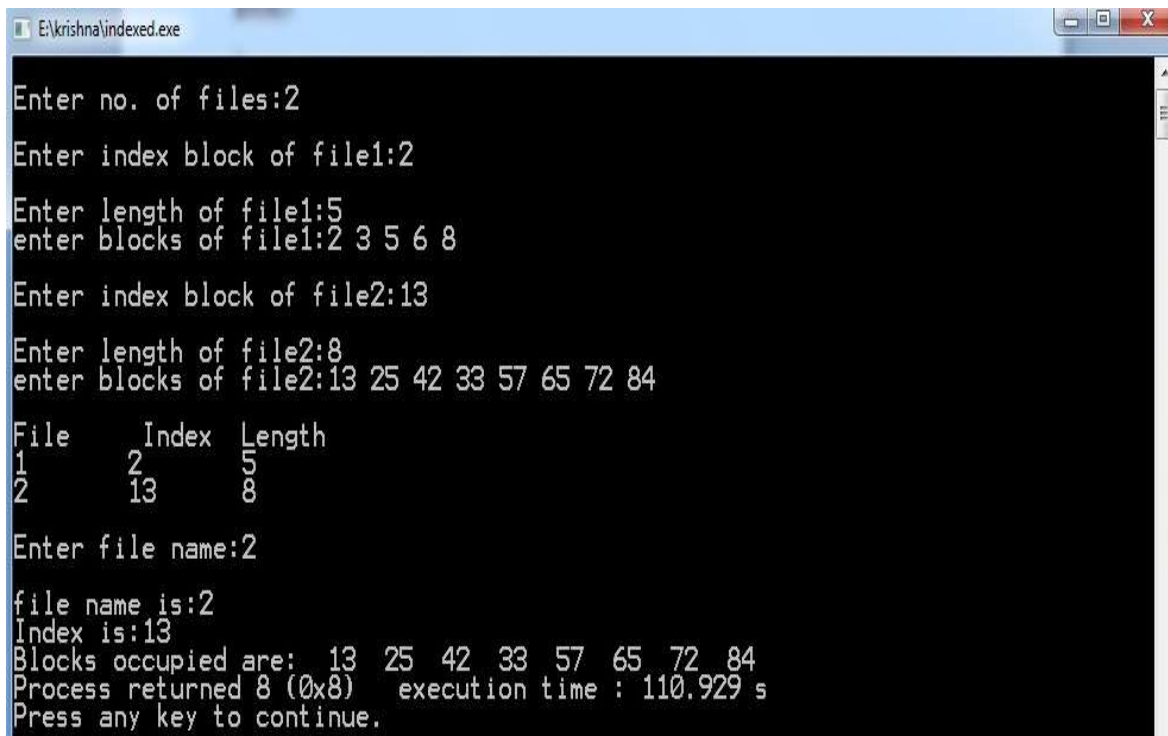
```
printf("\nBlocks occupied are:");
```

```
for(j=0;j<m[x-1];j++)
```

```
printf("%4d",b[x-1][j]);
```

```
}
```

**OUTPUT:**



```
E:\krishna\indexed.exe
Enter no. of files:2
Enter index block of file1:2
Enter length of file1:5
enter blocks of file1:2 3 5 6 8
Enter index block of file2:13
Enter length of file2:8
enter blocks of file2:13 25 42 33 57 65 72 84
File      Index  Length
1         2      5
2         13     8
Enter file name:2
file name is:2
Index is:13
Blocks occupied are: 13 25 42 33 57 65 72 84
Process returned 8 (0x8)   execution time : 110.929 s
Press any key to continue.
```

## VIVA-VOCE

### 1. List File access methods.

**There are three ways to access a file into a computer system: Sequential-Access, Direct Access, Index sequential Method.**

- **Sequential Access – It is the simplest access method. ...**
- **Direct Access – Another method is direct access method also known as relative access method. ...**
- **Index sequential method**

## **2. Explain Sequential file allocation method.**

**In this allocation strategy, each file occupies a set of contiguous blocks on the disk. This strategy is best suited. For sequential files, the file allocation table consists of a single entry for each file. It shows the filenames, starting block of the file and size of the file.**

## **3. Explain Linked file allocation method.**

**Each file is a linked list of disk blocks which need not be contiguous. The disk blocks can be scattered anywhere on the disk. The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.**

## **4. Explain Indexed file allocation method.**

**Indexed allocation scheme stores all the disk pointers in one of the blocks called as indexed block. Indexed block doesn't hold the file data, but it holds the pointers to all the disk blocks allocated to that particular file. Directory entry will only contain the index block address.**

## **5. What is file system mounting?**



**Mounting a file system attaches that file system to a directory (mount point) and makes it available to the system. The root ( / ) file system is always mounted. Any other file system can be connected or disconnected from the root ( / ) file system.**

### **Week-13:**

**NAME OF THE EXPERIMENT:Memory management techniques**

**AIM: Write a C program to simulate the following memory management techniques**

**a) Paging**

**b) Segmentation**

**a) PAGING Algorithm:**

**SOURCE CODE:**

```
#include<stdio.h>
```

```
void main(){
```

```
int i,j,temp,framearr[20],pages,pageno,frames,memsize,log,pagesize,prosize,base;
```

```
printf("Enter the Process size: ");
```

```
scanf("%d",&prosize);
```

```
printf("\nEnter the main memory size: ");
```

```
scanf("%d",&memsize);
```

```
printf("\nEnter the page size: ");
```

```
scanf("%d",&pagesize);
```

```

pages=prosize/pagesize;

printf("\nThe process is divided into %d pages",pages);

frames = memsize/pagesize;

printf("\n\nThe main memory is divided into %d frames\n",frames);

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

    framearr[i]=-1;    /* Initializing array elements with -1*/

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

pos:  printf("\nEnter the frame number of page %d: ",i);

    scanf("%d",&temp); /* storing frameno in temporary variable*/

    if(temp>=frames) /*checking wether frameno is valid or not*/

    {

        printf("\n\t***Invalid frame number***\n");

        goto pos;

    }

    /* storing pageno (i.e 'i' ) in framearr at framno (i.e temp ) index */

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

        if(temp==j)

            framearr[temp]=i;

}

printf("\n\nFrameno\tpageno\tValidationBit\n-----\t-----\t-----");

```

```

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

    printf("\n %d \t %2d \t",i,framearr[i]);

    if(framearr[i]==-1)

        printf(" 0");

    else

        printf(" 1");

}

printf("\nEnter the logical address: ");

scanf("%d",&log);

printf("\nEnter the base address: ");

scanf("%d",&base);

pageno = log/pagesize;

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

    if(framearr[i]==pageno)

    {

        temp = i;

        break;

    }

j = log%pagesize;    /* here 'j' is displacement */

```

**temp = base + (temp\*pagesize)+j; //lhs 'temp' is physical address rhs and 'temp' is frame num**

**printf("\nPhysical address is : %d",temp);**

**}**

**Output:**

## **b) SEGMENTATION ALGORITHM**

**SOURCE CODE:**

**#include<stdio.h>**

**void main(){**

```

int i,j,m,size,val[10][10],badd[20],limit[20],ladd;

printf("Enter the size of the segment table:");

scanf("%d",&size);

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

    printf("\nEnter infor about segment %d",i+1);

    printf("\nEnter base address:");

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

    printf("\nEnter the limit:");

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

    for(j=0;j<limit[i];j++){

        printf("\nEnter %d address values:",badd[i]+j);

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

    }

}

printf("\n\n****SEGMENT TABLE****");

printf("\nseg.no\tbase\tlimit\n");

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

{

    printf("%d\t%d\t%d\n",i+1,badd[i],limit[i]);

}

printf("\nEnter segment no.::");

```

```
scanf("%d",&i);

if(i>=size)

{

printf("invalid");

}

else

{

printf("\nEnter the logical address:");

scanf("%d",&ladd);

if(ladd>=limit[i])

printf("invalid");

else

{

m=badd[i]+ladd;

printf("\nmapped physical address is=%d",m);

printf("\nthe value is %d ",val[i][ladd]);

}  }}
```

**OUTPUT:**

```
E:\krishna\segment.exe
Enter the size of the segment table:3
Enter infor about segment 1
Enter base address:2
Enter the limit:3
Enter 2 address values:11
Enter 3 address values:12
Enter 4 address values:13
Enter infor about segment 2
Enter base address:22
Enter the limit:4
Enter 22 address values:33
Enter 23 address values:44
Enter 24 address values:55
Enter 25 address values:66
Enter infor about segment 3
Enter base address:75
Enter the limit:2
Enter 75 address values:88
Enter 76 address values:99

****SEGMENT TABLE****
seg.no  base  limit
1        2    3
2       22    4
3       75    2

Enter segment no.:1
Enter the logical address:2
mapped physical address is=24
the value is 55
Process returned 17 (0x11)   execution time : 86.104 s
Press any key to continue.
```



## **VIVA-VOCE**

### **1.What is the basic function of paging?**

Paging is a memory management scheme that permits the physical-address space of a process to be non contiguous. It avoids the considerable problem of having to fit varied sized memory chunks onto the backing store.

### **2.What is fragmentation?**

Fragmentation is an unwanted problem in the operating system in which the processes are loaded and unloaded from memory, and free memory space is fragmented. Processes can't be assigned to memory blocks due to their small size, and the memory blocks stay unused.

### **3.What is thrashing?**

Thrashing is caused by under allocation of the minimum number of pages required by a process, forcing it to continuously page fault.

### **4.Differentiate between logical and physical address.**

Logical Address is generated by CPU while a program is running. The logical address is virtual address as it does not exist physically, therefore, it is also known as Virtual Address.

Physical Address identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address.

### **5.Explain internal fragmentation and external fragmentation.**

#### **Internal Fragmentation**

When a process is allocated to a memory block, and if the process is smaller than the amount of memory requested, a free space is created in the given memory block. Due to this, the free space of the memory block is unused, which causes internal fragmentation.

## **External Fragmentation**

External fragmentation happens when a dynamic memory allocation method allocates some memory but leaves a small amount of memory unusable. The quantity of available memory is substantially reduced if there is too much external fragmentation. There is enough memory space to complete a request, but it is not contiguous. It's known as external fragmentation

### **Week-14:**

**NAME OF THE EXPERIMENT:Page Replacement Techniques**

**AIM: Write C programs to simulate the following Page Replacement Techniques:**

**a) FIFO   b) LRU   c)OPTIMAL**

**SOURCE CODE:**

```
#include<stdio.h>

void main()

{

    int i,j,n,a[50],frame[10],fno,k,avail,pagefault=0;

    printf("\nEnter the number of Frames : ");

    scanf("%d",&fno);

    printf("\nEnter number of reference string :");

    scanf("%d",&n);

    printf("\n Enter the Reference string :\n");

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

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

```

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

frame[i]= -1;

j=0;

printf("\n FIFO Page Replacement Algorithm\n\n The given reference string is:\n\n");

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

{

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

}

printf("\n");

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

{

printf("\nReference No %d-> ",a[i]);

avail=0;

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

if(frame[k]==a[i])

avail=1;

if (avail==0)

{

frame[j]=a[i];

j = (j+1) % fno;

```

```
pagefault++;

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

if(frame[k]!=-1)

printf(" %2d",frame[k]);

    }

printf("\n");

    }

printf("\nPage Fault Is %d",pagefault);

    }
```

**Output:**

```
E:\krishna\fifo.exe

Enter the number of Frames : 3
Enter number of reference string :10
Enter the Reference string :
4 3 2 3 1 4 3 1 3 2

FIFO Page Replacement Algorithm
The given reference string is:
4 3 2 3 1 4 3 1 3 2
Reference No 4-> 4
Reference No 3-> 4 3
Reference No 2-> 4 3 2
Reference No 3->
Reference No 1-> 1 3 2
Reference No 4-> 1 4 2
Reference No 3-> 1 4 3
Reference No 1->
Reference No 3->
Reference No 2-> 2 4 3

Page Fault Is 7
Process returned 16 (0x10) execution time : 25.888 s
Press any key to continue.
```

**b) LRU Algorithm:**

**SOURCE CODE:**

```
#include<stdio.h>
```

```
void main()
```

```
{
```

```
int i,j,l,max,n,a[50],frame[10],flag,fno,k,avail,pagefault=0,lru[10];
```

```

printf("\nEnter the number of Frames : ");

scanf("%d",&fno);

printf("\nEnter number of reference string :");

scanf("%d",&n);

printf("\n Enter the Reference string :\n");

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

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

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

{

frame[i]= -1;

lru[i] = 0;

}

printf("\nLRU Page Replacement Algorithm\n\nThe given reference string is:\n\n");

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

{

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

}

printf("\n");

j=0;

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

```

```

    {

max = 0;

flag=0;

printf("\nReference No %d-> ",a[i]);

avail=0;

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

if(frame[k]==a[i])

    {

    avail=1;

    lru[k]=0;

    break;

    }

if(avail==1)

    {

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

if(frame[k]!=-1)

    ++lru[k];

max = 0;

for(k=1;k<fno;k++)

if(lru[k]>lru[max])

```

```
max = k;

    j = max;

}

if(avail==0)

{

lru[j]=0;

frame[j]=a[i];

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

{

if(frame[k]!=-1)

    ++lru[k];

else

{

    j = k;

flag = 1;

break;

}

}

if(flag==0){

max = 0;
```



```
for(k=1;k<fno;k++)

if(lru[k]>lru[max])

max = k;

    j = max;

}

pagefault++;

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

if(frame[k]!=-1)

printf(" %2d",frame[k]);

}

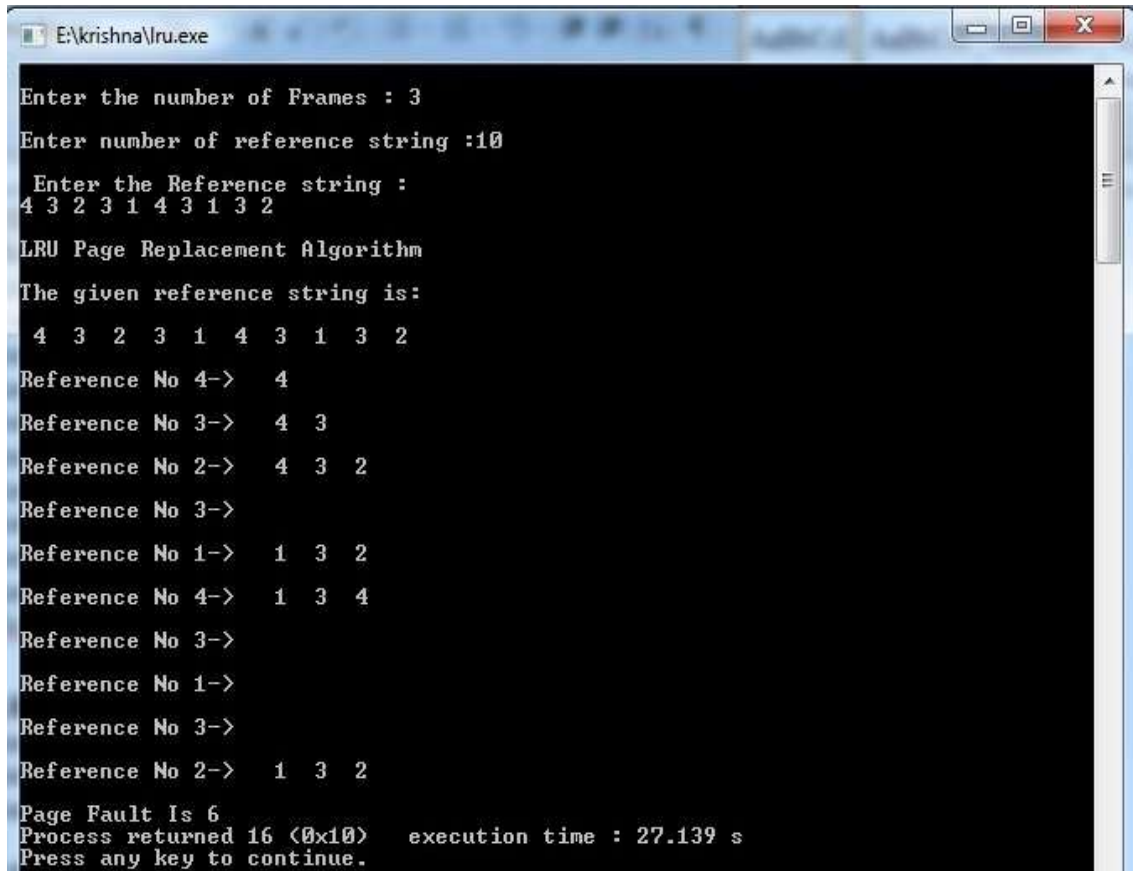
printf("\n");

}

printf("\nPage Fault Is %d",pagefault);

}
```

**OUTPUT :**



```
E:\krishna\lru.exe
Enter the number of Frames : 3
Enter number of reference string :10
Enter the Reference string :
4 3 2 3 1 4 3 1 3 2
LRU Page Replacement Algorithm
The given reference string is:
4 3 2 3 1 4 3 1 3 2
Reference No 4-> 4
Reference No 3-> 4 3
Reference No 2-> 4 3 2
Reference No 3->
Reference No 1-> 1 3 2
Reference No 4-> 1 3 4
Reference No 3->
Reference No 1->
Reference No 3->
Reference No 2-> 1 3 2
Page Fault Is 6
Process returned 16 (0x10) execution time : 27.139 s
Press any key to continue.
```

c) Optimal Algorithm:

SOURCE CODE :

```
#include<stdio.h>
```

```
int main()
```

```
{
```

```
int i,j,l,min,flag1,n,a[50],temp,frame[10],flag,fno,k,avail,pagefault=0,opt[10];
```

```
printf("\nEnter the number of Frames : ");
```

```
scanf("%d",&fno);
```

```
printf("\nEnter number of reference string :");
```

```

scanf("%d",&n);

printf("\n Enter the Reference string :\n");

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

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

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

{

frame[i]= -1;

opt[i]=0;

}

printf("\nLFU Page Replacement Algorithm\n\nThe given reference string is:\n\n");

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

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

printf("\n");

j=0;

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

{

flag=0;

flag1=0;

printf("\nReference No %d-> ",a[i]);

avail=0;

```

```
for(k=0;k<fno;k++)
```

```
if(frame[k]==a[i])
```

```
{
```

```
    avail=1;
```

```
    break;
```

```
}
```

```
if(avail==0)
```

```
{
```

```
    temp = frame[j];
```

```
    frame[j]=a[i];
```

```
    for(k=0;k<fno;k++)
```

```
    {
```

```
        if(frame[k]==-1)
```

```
        {
```

```
            j = k;
```

```
        flag = 1;
```

```
        break;
```

```
    }
```

```
}
```

```
if(flag==0)
```

```

        {

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

        {

opt[k]=0;

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

        {

if(frame[k]==a[l])

                {

                        flag1 = 1;

break;

                }

        }

if(flag1==1)

opt[k] = l-i;

else

        {

opt[k] = -1;

break;

        }

}

```

```

min = 0;

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

if(opt[k]<opt[min]&&opt[k]!=-1)

min = k;

else if(opt[k]==-1)

    {

min = k;

frame[j] = temp;

frame[k] = a[i];

break;

    }

    j = min;

    }

pagefault++;

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

if(frame[k]!=-1)

printf(" %2d",frame[k]);

    }

printf("\n");

    }

```

```
printf("\nPage Fault Is %d",pagefault);
```

```
return 0;
```

```
}
```

**Output:**

```
E:\krishna\optimal.exe
Enter the number of Frames : 3
Enter number of reference string :10
Enter the Reference string :
4 3 2 3 1 4 3 1 3 2
LFU Page Replacement Algorithm
The given reference string is:
4 3 2 3 1 4 3 1 3 2
Reference No 4-> 4
Reference No 3-> 4 3
Reference No 2-> 4 3 2
Reference No 3->
Reference No 1-> 4 3 1
Reference No 4->
Reference No 3->
Reference No 1->
Reference No 3->
Reference No 2-> 2 3 1
Page Fault Is 5
Process returned 0 (0x0) execution time : 17.034 s
Press any key to continue.
```

## **VIVA-VOCE**

### **1. Why do we use page replacement algorithms?**

Page replacement algorithms are an important part of virtual memory management and it helps the OS to decide which memory page can be moved out, making space for the currently needed page. However, the ultimate objective of all page replacement algorithms is to reduce the number of page faults

### **2. Which is best page replacement algorithm and why?**

LRU resulted to be the best algorithm for page replacement to implement. LRU maintains a linked list of all pages in the memory, in which, the most recently used page is placed at the front, and the least recently used page is placed at the rear.

### **3. Explain LRU algorithm.**

LRU stands for Least Recently Used. LRU replaces the line in the cache that has been in the cache the longest with no reference to it. It works on the idea that the more recently used blocks are more likely to be referenced again.

### **4. When does a page fault occur?**

Page fault occurs when a requested page is mapped in virtual address space but not present in memory.

### **5. What are page replacement algorithms in OS?**

**1. First In First Out (FIFO)**

**2. Optimal Page replacement**

**3. Least Recently Used**



