

Index

Experiment No.	Title	Page No.
1	Basics of UNIX commands	
2	Shell Programming	
3	Implement the following CPU scheduling algorithms: Round Robin SJF FCFS Priority	
4	Implement all file allocation strategies: Sequential Indexed Linked	
5	Implement Semaphores.	
6	Implement all File Organization Techniques: Single level directory Two level Hierarchical DAG	
7	Implement Bankers Algorithm for Dead Lock Avoidance.	
8	Implement an Algorithm for Dead Lock Detection.	
9	Implement e all page replacement algorithms: FIFO LRU LFU	
10	Implement Shared memory and IPC.	
11	Implement Paging Technique of memory management.	
12	Implement Threading & Synchronization Applications.	

COURSE STRUCTURE:

			Semester VI											
1	BTITC601	Operating Systems	3	-	-	20	20	60		100	3	3		
2	BTITC602	Compiler Construction	3	-	-	20	20	60		100	3	3		
3	BTITC603	Object Oriented Software and Web Engineering	3	-	-	20	20	60		100	3	3		
4	BTITC604	Digital Image Processing	3	-	-	20	20	60		100	3	3		
5	Elective V (Open)		3	-	-	20	20	60		100	3	3		
	BTITOE605A	Enterprise Resource Planning												
	BTITOE605B	Decision Support Systems												
	BTITOE605C	Software Project Management												
6	Elective VI		3	-	-	20	20	60		100	3	3		
	BTITPE606A	Software Testing												
	BTITPE606B	Data Storage Technologies & Networks												
	BTITPE606C	Service Oriented Architecture												
	BTITPE606D	Network Programming												
	BTITPE606E	Advanced Database Technology												
7	BTITL607	Operating Systems Lab	-	-	2	-	15	15	10	10	50	1	2	
8	BTITL608	Digital Image Processing Lab	-	-	2	-	15	15	10	10	50	1	2	

COURSE OBJECTIVES:

1. To learn shell programming and the use of filters in the UNIX environment.
2. To learn to programming in C using system calls.
3. To learn to use the file system related system calls.
4. To process creation and inter process communication.
5. To familiarize with implementation of CPU Scheduling Algorithms, page replacement algorithms and Deadlock avoidance.

COURSE OUTCOMES:

At the end of this course Students will be able to:

- CO1: Describe the important computer system resources and the role of operating system in their management policies and algorithms.
- CO2: Understand the process management policies and scheduling of processes by CPU
- CO3: Evaluate the requirement for process synchronization and coordination handled by operating system
- CO4: Describe and analyze the memory management and its allocation policies.
- CO5: Identify use and evaluate the storage management policies with respect to different storage management technologies.

GUIDELINES FOR STUDENT'S LAB JOURNAL

- 1) The laboratory assignments are to be submitted by student in the form of journal.
 - 2) The Journal consists of prologue, Certificate, table of contents, and handwritten write-up of each assignment (Title, Objectives, Problem Statement, Outcomes, software & Hardware requirements, Date of Completion, Assessment grade/marks and assessor's sign, Theory-Concept, circuit diagram, pin configuration, conclusion/analysis), printouts of the code written using coding standards, sample test cases etc.
 - 3) Practical Examination will be based on the term work submitted by the student in the form of Journal
 - 4) Candidate is expected to know the theory involved in the experiment
 - 5) The practical examination should be conducted if the journal of the candidate is completed in all respects and certified by concerned faculty and head of the department
 - 6) All the assignment mentioned in the syllabus must be conducted
-

Experiment No: 1

Date of Conduction:

Name:

Roll No:

Experiment No. 1

Problem Statement: Basics of UNIX commands

Objective(s): To study and Use Linux GUI and Commands

Software Required: Ubuntu 14.04 or above

Theory:

o **Operating system**

An operating system (OS) is a set of software that manages computer hardware resources and provides common services for computer programs. Operating system is the interface between computer hardware & user of computer.

o **GUI Commands:**

BOSS GNU/Linux provides reach set of commands to manipulate linux file system, user Access, managing hardwares, connecting to internet etc. Following are some of them

o **Navigating and Searching the File System**

This section introduces you to the basic navigation commands, and shows you how to move around your Linux file system, find files, and build file information databases

A. Moving to Different Directories with the cd Command

The cd (change directory) command is the basic navigation tool for moving your current location to different parts of the Linux file system. You can move directly to a directory by typing the command, followed by a pathname or directory. For example, the following command will move you to the /usr/bin directory:

```
# cd /usr/bin
```

When you're in that directory, you can move up to the /usr directory with the following command:

```
# cd ..
```

You could also move to the root directory, or / , while in the /usr/bin directory by using the following command:

```
# cd../ ..
```

Finally, you can always go back to your home directory (where your files are) by using either of the following commands:

```
# cd
```

or

```
# cd ~
```

Knowing Where You Are with the pwd Command

The pwd (print working directory) command tells you where you are, and prints the working (current) directory. For example, if you execute

```
# cd / usr/ bin and then type # pwd
you'll see
/ usr/ bin
```

B. Reading Directories and Files

Other basic Linux commands to list the contents of directories make a catalogue of your hard drive, and read the contents of files.

Listing Directories with the ls Command

The ls (list directory) command will quickly become one of your most often used programs. In its simplest form, ls lists nearly all of the files in the current directory. But this command, which has such a short name, probably has more command -line options

```
# ls
News  ax      homens      mail  Author.msg  documents
```

You can also list the files as a single line, with comma separations, with the -m option:

```
# ls -m
News, author.msg, auto, axhome, documents, mail, nsmail, reading, a research, search,
```

If you don't like this type of listing, you can have your files sorted horizontally, instead of vertically (the default), with the -x option:

```
# ls -x
      News      axhome      nsmail
      search    author.msg  documents
      reading   vultures.msg auto
      mail      research
```

Specifying Other Directories

You can also use the ls command to view the contents of other directories by specifying the directory, or pathname, on the command line. For example, if you want to see all the files in the /usr/bin directory, use

```
# ls /usr/bin
```

C. Manipulating Files or Directories

Using Linux isn't different from any other computer operating system. You create, delete, and move files on your hard drive in order to organize your information and manage how your system works or looks

Creating Files with the touch Command

The touch command is easy to use, and generally, there are two reasons to use it. The first reason is to create a file, and the second is to update a file's modification date. The touch command is part of the GNU file utilities package, and has several options. To create a file with touch, use

```
# touch newfile # ls -l newfile
-rw-rw-r-- 1 bball bball 0 Nov 13 08:50 newfile
```

Deleting Files with the rm Command, the rm command deletes files. This command has several simple options

```
# rm file
# rm file1 file2 file3 # rm filefolder
```

Creating Directories with the mkdir Command

The mkdir command can create one or several directories with a single command line. You may also be surprised to know that mkdir can also create a whole hierarchy of directories, which includes parent and children, with a single command line.

The following simple command line creates a single directory:

```
# mkdir temp
```

But you can also create multiple directories with

```
# mkdir temp2 temp3 temp4
```

You'd think that you could also type the following to make a directory named child under temp:

```
# mkdir temp/ child
```

Removing Directories with the rmdir command. The rmdir command is used to remove directories but it should be empty first. To remove a directory, all you have to do is type

```
# rmdir tempdirectory
```

The rmdir command, like mkdir, also has a -p, or parent, option. You can use this option to remove directory hierarchies, for example:

```
# rmdir -p temp5/parent/child
```

Finally, ! As you can see, you must specify the complete directory tree to delete it. If you use the same command line, but without the -p option, only the child directory would be deleted. But what if there are two or more subdirectories, for example:

In order to delete the entire directory system of temp5, you'd need to use

```
# rmdir temp5/ parent/*
```

Renaming Files with the mv Command. The mv command, called a rename command but known to many as a move command, will indeed rename files or directories, but it will also move them around your file system. In its simplest form, mv can rename files, for example:

```
# touch file1
```

```
# mv file1 file2
```

This command renames file1 to file2. However, besides renaming files, mv can rename directories, whether empty or not, for example:

```
# mv temp newtemp
```

Copying with the cp Command. The cp, or copy, command is used to copy files or directories. You'll most likely first use cp in its simplest form, for example:

```
# cp file1 file2
```

This creates file2, and unlike mv, leaves file1 in place. But you must be careful when using cp, because you can copy a file onto a second file, effectively replacing it!

○ Managing User Access

One of your main jobs as a sysadmin is to manage the users on your system. This involves creating accounts for new users, assigning home directories, specifying an initial shell for the user, and possibly restricting how much disk space, memory, or how many processes each person can use.

Creating Users with the adduser Command

One of the first things you should do after installing Linux is to create a user account for yourself. You'll want to do all your work in Linux through this account, and do your system management using the su command. There are several ways to create new users in Linux, but this section shows you the easy way, using a trio of commands: adduser, passwd, and chfn.

The first step in creating a new user is to use the adduser command, found under the /usr/sbin directory. You must be the root operator to run this program:

```
# adduser
```

Only root may add users to the system.

The adduser program also requires you to specify a user name on the command line, for example:

```
# adduser cloobie
```

```
Looking for first available UID... 502 Looking for first available GID... 502 Adding login:
cloobie...done.
```

```
Creating home directory: /home/cloobie...done. Creating mailbox:
/var/spool/mail/cloobie...done. Don't forget to set the password.
```

To add a password, type the command, along with the new user's name:

```
# passwd cloobie
```

```
New UNIX password:
```

```
Retype new UNIX password:
```

Finally, you'll also want to use the chfn command to enter formal information about users, or have your users enter this information. The chfn command will ask

```
# chfn cloobie
```

```
Changing finger information for cloobie. Name [RHS Linux User]: Mr. Cloobie Doo Office []: 400
Pennsylvania Ave.
```

```
Office Phone []: 202-555-1212 Home Phone []: 202-555-4000
```

○ Rebooting & shutting down

Linux also offers a way to shut down or reboot from the command line of a terminal. After typing su and pressing enter to switch to supervisor mode, use the shutdown command along with its -r(reboot option) followed by the word now like :

```
# shutdown -r now or # shutdown -r 0(zero)
```

○ Other Commands

Sudo Command

The sudo command is frequently used to execute a command that requires root privileges. For example, if you want to run a shell script setup.sh that installs a program into a directory that only root can modify you can use sudo as follow s:

```
#sudo -u root ./ setup.sh
```

You may be required to enter the root password before the command is executed. After you have logged in, you can continue to execute commands through sudo for a few minutes without having to specify the login (-u root) with every command. If possible, it is better to do your regular work using an account with restricted privileges to avoid causing serious damage to the system by accident. You can list the files of a protected directory with the following command:

```
#sudo ls /usr /local/ classified
```

Use the following line to reboot the system in 20 minutes:

```
#sudo shutdown -r +20 "rebooting to fix network issue"
```

rpm Command

Red Hat introduced RPM in 1995. RPM is now the package management system used for packaging in the Linux Standard Base (LSB). The rpm command options are grouped into three subgroups for:

- Querying and verifying packages
- Installing, upgrading, and removing packages
- Performing miscellaneous functions

We will focus on the first two sets of command options in this article. You will find information about the miscellaneous functions in the man pages for RPM.

We should also note that rpm is the command name for the main command used with RPM, while .rpm is the extension used for RPM files. So "an rpm" or "the xxx rpm" will generally refer to an RPM file, while rpm will usually refer to the command.

```
# rpm -i gcl-2.6.8-0.6.20090701cvs.fc12.x86_64.rpm
```

yum Command

yum adds automatic updates and package management, including dependency management, to RPM systems. In addition to understanding the installed packages on a system, YUM, like the Debian Advanced Packaging Tool (APT), works with repositories, which are collections of packages, typically accessible over a network connection.

```
# yum install gcl
```

```
#tty Cammand
```

print the file name of the terminal connected to standard input

```
tty [OPTION]...
```

options can be -s, --silent, --quiet print nothing, only return an exit status and if --version output version information and exit

who cammand

if you want to know which users are currently logged in to your Linux system, which console they're using, and the date and time they logged in, issue the who command. You'll see output something like this:

```
# who
```

```
root    tty1    Nov  2    17:57
hermietty3  Nov  2    18:43
```

sigmund tty2 Nov 2 18:08

In the output shown here, the term tty stands for teletype. In the olden days of computing, a terminal was just a keyboard with an attached printer, so you read everything off the teletype.

If you've logged in with multiple virtual consoles and changed your identity on any of them, you may have some trouble figuring out who you are --or at least what user is logged in to the console you're using. If you find yourself in such an identity crisis, try this related command:

The whoami command will tell you the name of the current user. Just as a side note, you can also use the who am i command (a variant of the who command) to return the name of the current user. But it doesn't always work s you might expect. If you're logged in as root, and use the su command to switch to another user, who am i will return "root" as the current user. For this reason, I recommend that you train yourself to always use the whoami command when you want to know the current user name.

○ Vi Commands

The default editor that comes with the UNIX operating system is called vi (visual editor). [Alternate editors for UNIX environments include pico and emacs, a product of GNU

To Creates an empty file with the help of vi editor use following command

#vi srcfile.ext

Once file created you can enter the contents into the file by swiching into insert mode by pressing i you can type the text in file, to return to command mode press esc button. You can save & exit the file using the following parameters & pressing enter

:wq

For compiling the program file use gcc compiler and give the filename as input as shown

#gcc srcfile.c

which creates the a.out file and execute the a.out file using

#./ a.out Or

#gcc -o exefile srcfile.ext

which creates executable output file with name exefile and execute this exefile using

#./ exefile

Output:

```

sitrc@sitrc-OptiPlex-380: ~
sitrc@sitrc-OptiPlex-380:~$ who
sitrc      :0                2014-10-14 10:58 (:0)
sitrc     pts/13            2014-10-14 10:59 (:0)
sitrc@sitrc-OptiPlex-380:~$ whoami
sitrc
sitrc@sitrc-OptiPlex-380:~$ date
Tue Oct 14 11:05:57 IST 2014
sitrc@sitrc-OptiPlex-380:~$ mkdir Bhushan-Chaudhari
sitrc@sitrc-OptiPlex-380:~$ man cp
sitrc@sitrc-OptiPlex-380:~$ ls
0                               gr.C~                               server.C~
a.out                           HelloWorld.java~                   ssh.java
App.java~                       mongo-java-driver-2.12.3          sh.java
Bhushan                         Mon.java~                         SimpleServlet.java~
Bhushan-Chaudhari              Music                             Templates
client.C~                      MyServlet.java~                  Untitled Document~
database.C~                    NetBeansProjects                 usb
Desktop                        output.pdf                       validation in php.odt
Documents                      pd bk                            Videos
Downloads                     Pictures                         VirtualBox VMs
fontconfig                     Public                           workspace
google                         RequestDemoServlet.java~
sitrc@sitrc-OptiPlex-380:~$ pwd
/home/sitrc
sitrc@sitrc-OptiPlex-380:~$

```

Conclusion:

Hence we learned use of Linux commands.

Experiment No: 2

Date of Conduction:

Roll No:

Experiment No. 2

Problem Statement: To learn Shell Programming

Theory:

A Linux shell is a command language interpreter, the primary purpose of which is to translate the command lines typed at the terminal into system actions. The shell itself is a program through which other programs are invoked

Shell script:

- A shell script is a file containing a list of commands to be executed by the Linux shell. Shell script provides the ability to create your own customized Linux commands
- Linux shell have sophisticated programming capabilities which makes shell script powerful Linux tools

EX :2a SHELL PROGRAM TO SIMULATE THE FORK() AND GETPID() SYSTEM CALLS

AIM

Write the shell program to simulate the fork and getpid() system calls.

ALGORITHM

Step 1: Initialize the max count value and buffer size.

Step 2: Get the fork() system calls to execute the process. It makes two identical copies of address the space one for the parent and other for the child.

Step 3: Both process start their execution right after the system call fork()

- a) If both the processes have identical but separate address space.
- b) The fork() call have the same values in both address space, since every process has its own address space, any modification will be independent of the others.

Step 4: Get the process id, by using getpid() of system call.

Step 5: The printf() is buffer meaning printf() will group the output of a process together it print the process id and value.

PROGRAM

```
#include<stdio.h>
#include<string.h>
#include<sys/types.h>
#define MAX_COUNT 5
#define BUF_SIZE 100
main(void)
{
    pid_t pid;

    int i;
    char buf[BUF_SIZE];
    fork();
```

```

pid=getpid();
for(i=1;i<MAX_COUNT;i++)
{
printf(buf,"this line is from pid %d value=%d \n",pid,i);
write(1,buf,strlen(buf));
}
}

```

OUTPUT

```

[User16@sambatelnet]$ cc prg2a.c
[User16@sambatelnet]$ ./a.out
this line is from pid 8081 value=1
this line is from pid 8081 value=2
this line is from pid 8081 value=3
this line is from pid 8081 value=4

```

CONCLUSION

Thus the shell program to simulate fork() and getpid() system call is executed and the output is verified.

EX:2b SHELL PROGRAM TO DEMONSTRATE EXECLP() FUNCTION

AIM

Write a shell program to implement the execlp() function.

ALGORITHM

Step 1: Obtain the pid value using fork() function.

Step 2: If pid<0, then print fork failed.

Step 3: Else if pid=0, execlp() function will be invoked. Step 4: Else print child process complete.

PROGRAM

```

#include<stdio.h>
main(int argC,char *argV[])
{
int pid; pid=fork(); if(pid<0)
{
fprintf(stderr,"fork failed \n"); exit(-1);
}
else if(pid==0)
{
execlp("/bin/ls","ls",NULL);
}
else
{
wait(NULL); printf("child complete"); exit(0);
}
}

```

OUTPUT

```
[User16@sambatelnet]$ cc execlp.c
[User16@sambatelnet]$ ./a.out
child complete
```

CONCLUSION

Thus the execlp() function is executed and the output is verified.

EX:2c SHELL PROGRAM TO DEMONSTRATE SLEEP() FUNCTION**AIM**

Write a shell program to demonstrate the sleep() function.

ALGORITHM

Step 1: Get the pid using fork() function.

Step 2: Check the pid value

- a) If pid=0, then display child process and put the process to sleep.
- b) Else display parent process function.
- c) Display the pid.

PROGRAM

```
#include<stdio.h>
main()
{
    int p; p=fork();
    if(p==0) {
        printf("\nI am a child process %d",getpid()); sleep(15);
        printf("\nI am a parent of child process %d",getpid());
    }
    else
    {
        printf("\nI am parent of child %d",getpid()); printf("\nI am parent's parent %d",getpid());
    }
}
```

OUTPUT

```
I am a parent of child 8563
I am parent's parent 8563
I am a child process 8564
I am a parent of child process 8564
```

CONCLUSION

Thus the sleep function was implemented and the output is verified.

EX:2d SHELL PROGRAM FOR OPENDIR(), READDIR() FUNCTION

AIM

Write a shell program to demonstrate the I/O system calls.

ALGORITHM

Step 1: Define a pointer to a structure dirent predefined structure DIR and another pointer to a structure called preaddir.

Step 2: The directory is opened using the opendir() function.

Step 3: The readdir() function reads the name of the first file from this opened directory.

Step 4: The third call to this function gets the first entry whose name is stored in the member dirent of the structure properties.

Step 5: The closedir() function closes this open directory.

PROGRAM

```
#include<stdio.h>
#include<sys/types.h>
#include<dirent.h>
main(argC,argV)
int argC; char *argV[];
{
    DIR *dirname;
    struct dirent *preaddir;
    dirname=opendir(argV[1]);
    preaddir=readdir(dirname);
    preaddir=readdir(dirname);
    preaddir=readdir(dirname);
    printf("opened %s \n",argV[1]);
    printf("the first entry in the directory is %s \n",preaddir->d_name);
    closedir(dirname);
}
```

OUTPUT

```
[28jeit60@localhost 28jeit60]$ cd apple
[28jeit60@localhost 28jeit60]$ cat> operating system Hi
How are you
[28jeit60@localhost 28jeit60]$ cd
[28jeit60@localhost 28jeit60]$ ./a.out apple
Opened apple
The first entry in this directory is operating system
```

CONCLUSION

Thus the shell program to demonstrate the I/O system calls is completed.

Experiment No: 3

Date of Conduction:

Roll No:

Experiment No. 3

Problem Statement:

Implement the following CPU scheduling algorithms:

- a) Round Robin
- b) SJF
- c) FCFS
- d) Priority

Theory:

With this scheme, the process that requests the CPU first is allocated the CPU first. The implementation of the FCFS policy is easily managed with a FIFO queue. The FCFS scheduling algorithm is non-preemptive. Once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU either by terminating or by requesting I/O. While considering the performance of the FCFS scheduling algorithm, the higher burst time process makes the lower burst time process to wait for a long time. This effect is known as CONVOY effect.

AIM

Write a program to perform first come first serve scheduling algorithm and compute the average waiting time and average turnaround time.

ALGORITHM

Step 1: Get the number of processes and burst time.

Step 2: The process is executed in the order given by the user. Step 3: Calculate the waiting time and turnaround time.

Step 4: Display the gantt chart, avg waiting time and turnaround time.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int bt[20],p,wt=0,tat,i,twt=0,ttat=0;
    printf("Enter the no.of process:");
    scanf("%d",&p);
    printf("Enter the no.of burst time for each process:");
    for(i=0;i<p;i++)
    {
        scanf("\n%d",&bt[i]);
    }
    printf("Burst time \t\t Waiting time \t\t Turn-round time");
    for(i=0;i<p;i++)
```

```

{
twt=twt+wt; tat=bt[i]+wt;
ttat=ttat+tat;
printf("\n %d \t\t %d \t\t %d",bt[i],wt,tat);
wt=wt+bt[i];
}
printf("\n\n Average waiting time :%d",twt/p);
printf("\n\n Average turn around time :%d",ttat/p);
return(0);
getch();
}

```

OUTPUT

[csea2001@samba csea2001]\$ cc 5a.c

[csea2001@samba csea2001]\$./a.out

Enter the no. of process: 3

Enter the no. of burst time for each process:

5

10

15

Burst time	Waiting time	Turn-around time
5	0	5
10	5	15
15	15	30

Average waiting time: 6

Average turnaround time: 16

CONCLUSION

Thus the Program for first come first serve scheduling algorithm is completed.

EX:3b SHORTEST JOB FIRST SCHEDULING ALGORITHM THEORY

This algorithm associates with each process the length of the latter's next CPU burst. When the CPU is available, it is assigned to the process that has the smallest next CPU burst. If two processes have the same length next CPU burst, FCFS scheduling is used to break the tie. The SJF algorithm may be either preemptive or non-preemptive. The choice arises when a new process arrives at the ready queue while a previous process is executing. The SJF scheduling algorithm is provably optimal, in that it gives the minimum average waiting time for a given set of processes.

AIM

Write a program to perform shortest job first scheduling algorithm and compute the average waiting time and average turnaround time.

ALGORITHM

Step 1: Initialize and declare the variables.

Step 2: Enter the number of process and give for loop to display burst time.

Step 3: Using temporary variables shortest job is selected and executed.

Step 4: Average waiting time and turnaround time is displayed.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int main()
{
int bt[20],p,wt=0,tat,i,j,twt=0,ttat=0,temp;
printf("Enter the no.of process:");
scanf("%d",&p);
printf("Enter the no.of burst time for each process:");
for(i=0;i<p;i++)
{
scanf("\n%d",&bt[i]);
}
for(j=i+1;j<p;j++)
{
if(bt[i]>bt[j])
{
temp=bt[i];
bt[i]=bt[j];
bt[j]=temp;
}
}
printf("Burst time \t\t Waiting time \t\t Turn-round time");

for(i=0;i<p;i++)
{
twt=twt+wt; tat=bt[i]+wt;
ttat=ttat+tat;
printf("\n %d \t\t\t %d \t\t\t %d",bt[i],wt,tat);
wt=wt+bt[i];
}
printf("\n\n Average waiting time :%d",twt/p);
printf("\n\n Average turn around time :%d",ttat/p);
```

```
getch();
}
```

OUTPUT

```
[csea2001@samba csea2001]$ cc 5b.c
```

```
[csea2001@samba csea2001]$ ./a.out
```

Enter the no. of process: 3

Enter the no. of burst time for each process:

8

2

4

Burst time	Waiting time	Turn-around time
2	0	2
4	2	6
8	6	14

Average waiting time: 2

Average turnaround time: 7

CONCLUSION

Thus the Program for shortest job first scheduling algorithm is completed.

EX: 3c PRIORITY SCHEDULING ALGORITHM

THEORY

In Priority Scheduling, each process is given a priority, and higher priority methods are executed first, while equal priorities are executed First Come First Served or Round Robin.

AIM

Write a program to create and execute priority scheduling algorithm using c program.

ALGORITHM

Step 1: Declare the array size

Step 2: Read the number of processes to be inserted Step 3: Read the Priorities of processes

Step 4: Sort the priorities and Burst times in ascending order Step 5: Calculate the waiting time of each process

$wt[i+1]=bt[i]+wt[i]$

Step 6: Calculate the turnaround time of each process $tt[i+1]=tt[i]+bt[i+1]$

Step 7: Calculate the average waiting time and average turnaround time. Step 8. Display the values

PROGRAM

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

```

#include<conio.h>
void main()
{
int i,j,pno[10],prior[10],bt[10],n,wt[10],tt[10],w1=0,t1=0,s;
float aw,at;
clrscr();
printf("enter the number of processes:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("The process %d:\n",i+1);
printf("Enter the burst time of processes:");
scanf("%d",&bt[i]);
printf("Enter the priority of processes %d:",i+1);
scanf("%d",&prior[i]);
pno[i]=i+1;
}

for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
if(prior[i]<prior[j])
{
s=prior[i];
prior[i]=prior[j];
prior[j]=s;
s=bt[i]; bt[i]=bt[j];
bt[j]=s;
s=pno[i];
pno[i]=pno[j];
pno[j]=s;
}
}
}

for(i=0;i<n;i++)
{ wt[0]=0;
tt[0]=bt[0];
wt[i+1]=bt[i]+wt[i];
tt[i+1]=tt[i]+bt[i+1];
w1=w1+wt[i];
t1=t1+tt[i];
}

```

```

aw=w1/n;
at=t1/n;
}
printf(" \n job \t bt \t wt \t tat \t prior\n");
for(i=0;i<n;i++)
printf("%d \t %d \t %d \t %d \t %d\n",pno[i],bt[i],wt[i],tt[i],prior[i]);
printf("aw=%f \t at=%f \n",aw,at);
getch();
}

```

OUTPUT

```

TurboC Simulator [build 1.6]
enter the number of processes:3
The process 1:
Enter the burst time of processes:12
Enter the priority of processes 1:2
The process 2:
Enter the burst time of processes:14
Enter the priority of processes 2:1
The process 3:
Enter the burst time of processes:2
Enter the priority of processes 3:3

  job    bt    wt    tat    prior
2       14     0     14      1
1       12    14     26      2
3        2    26     28      3
aw=13.000000    at=22.000000

```

CONCLUSION

Thus the Program for priority scheduling algorithm is completed.

EX: 3d ROUND ROBIN SCHEDULING

THEORY

The round-robin scheduling algorithm is designed especially for time-sharing systems. It is similar to FCFS scheduling, but preemption is added to switch between processes. A small unit of time, called a time quantum is defined. A time quantum is generally from 10 to 100 milliseconds. The ready queue is treated as circular queue.

AIM

Write a program to implement Round Robin scheduling algorithms using C.

ALGORITHM


```

}
else if(b[i]<q)
{
t1=t1+b[i];
p[j]=t1;
j++;
}
b[i]=b[i]-q;
if(b[i]<=0)
tat[i]=t1;
}
else a++;
if(a==n+1) break;
if(i==n)
i=0;
}
printf("\n    \n");
for(i=0;i<j;i++)
printf("%d\t",p[i]);
for(i=1;i<=n;i++)
{
t=t+tat[i];
w=w+tat[i]-burst[i];
}
w=w/n; t=t/n;
printf("\nThe average waiting time is %0.2f",w);
printf("\nThe average turn around time is %0.2f",t);
getch();
}

```

OUTPUT

Enter the number of processes and Quantum 3 5 Enter the Burst Time 10
Enter the Burst Time 5 Enter the Burst Time 3

GanttChart

p1	p2	p3	p1
0	5	10	13
			18

The average Waiting Time 9.3333

The average Turnaround Time 13.67

CONCLUSION

Thus the Program for round robin scheduling algorithm is completed.

Experiment No: 4

Date of Conduction:

Roll No:

Experiment No. 4

Problem Statement:

Implement all file allocation strategies:

- Sequential
- Indexed
- Linked

Theory:

EX:4A SEQUENTIAL FILE ALLOCATION THEORY

In this type of strategy, the files are allocated in a sequential manner such that there is a continuity among the various parts or fragments of the file.

AIM

Write a c program for sequential file allocation.

ALGORITHM

Step 1: Read the number of files

Step 2: For each file, read the number of blocks required and the starting block of the file. Step 3: Allocate the blocks sequentially to the file from the starting block.

Step 4: Display the file name, starting block, and the blocks occupied by the file.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
main()
{
    int n,i,j,b[20],sb[20],t[20],x,c[20][20];
    clrscr();
    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("blocks occupiedare:");
    for(i=0;i<n;i++)
```

```

{
printf("fileno%d",i+1);
for(j=0;j<b[i];j++)
printf("\t%d",c[i][j]);
printf("\n");
}
getch();
}

```

OUTPUT

```

TurboC Simulator [build 1.6]
Enter no.of files:2
Enter no. of blocks occupied by file13
Enter the starting block of file14
Enter no. of blocks occupied by file22
Enter the starting block of file28
Filename      Start block      length
1             4             3
2             8             2
blocks occupied are:fileno1      4      5      6
fileno2 8      9

```

CONCLUSION

Thus the Program for sequential file allocation is completed.

EX:4B LINKED FILE ALLOCATION

THEORY

In this type of strategy, the files are allocated in a linked list format where each and every fragment is linked to the other file through either addresses or pointers. Thus, the starting location of the file serves for the purpose of extraction of the entire file because every fragment is linked to each other

AIM

Write a c program for linked file allocation strategy

ALGORITHM

Step 1: Read the number of files

Step 2: For each file, read the file name, starting block, number of blocks and block numbers of the file.

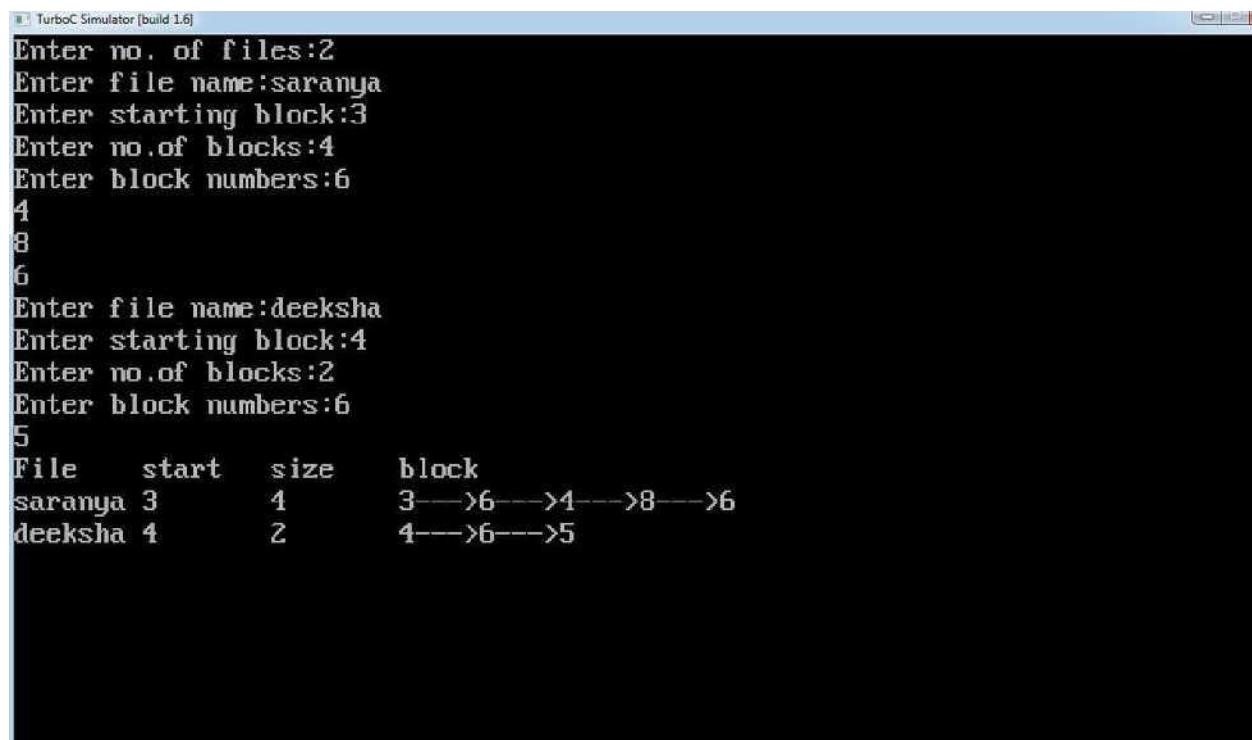
Step 3: Start from the starting block and link each block of the file to the next block in a Linked list fashion.

Step 4: Display the file name, starting block, size of the file, and the blocks occupied by the file.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
struct file
{
char fname[10];
int start,size,block[10];
}f[10];
main()
{
int i,j,n;
clrscr();
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;j++)
printf("%d--->",f[i].block[j]);
printf("%d",f[i].block[j]);
printf("\n");
}
getch();
}
```

OUTPUT



```

TurboC Simulator [build 1.6]
Enter no. of files:2
Enter file name:saranya
Enter starting block:3
Enter no.of blocks:4
Enter block numbers:6
4
8
6
Enter file name:deeksha
Enter starting block:4
Enter no.of blocks:2
Enter block numbers:6
5
File    start   size    block
saranya 3        4      3-->6-->4-->8-->6
deeksha 4        2      4-->6-->5
  
```

CONCLUSION

Thus the Program for file allocation using linked method is completed

EX: 4C INDEXED FILE ALLOCATION

THEORY

In this case, the files are allocated based on the indexes that are created for each fragment of the file such that each and every similar indexed file is maintained by the primary index thereby providing flow to the file fragments.

AIM

Write a C program for indexed file allocation.

ALGORITHM

Step 1: Read the number of files

Step 2: Read the index block for each file.

Step 3: For each file, read the number of blocks occupied and number of blocks of the file. Step

4: Link all the blocks of the file to the index block.

Step 5: Display the file name, index block, and the blocks occupied by the file.

PROGRAM

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

```
#include<conio.h>
```



```

main()
{
int n,m[20],i,j,ib[20],b[20][20];
clrscr();
printf("Enter no. of files:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter index block :",i+1);
scanf("%d",&ib[i]);
printf("Enter blocks occupied by 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,ib[i],m[i]);
printf("blocks occupiedare:");
for(i=0;i<n;i++)
{
printf("fileno%d",i+1);
for(j=0;j<m[i];j++)
printf("\t%d--->%d\n",ib[i],b[i][j]);
printf("\n");
}
getch();
}

```

OUTPUT

```

Enter no. of files:2
Enter index block 3
Enter blocks occupied by file1: 4
enter blocks of file1:9
4 6 7
Enter index block 5
Enter blocks occupied by file2:2
enter blocks of file2: 10 8
File index length
1 3 4
2 5 2

```

Blocks occupied are:

file1

3--->9

3--->4

3--->6

3--->7

file2

5--->10

5--->8

CONCLUSION

Thus the Program for allocating a file using indexed technique is completed.

Experiment No: 5

Date of Conduction:

Roll No:

Experiment No. 5

Problem Statement: Implement Semaphores for Producer Consumer Problem.

Objective: To understand use of semaphores.

Theory:

AIM

Write a c program to implement producer-consumer problem.

ALGORITHM

Step 1: Declare the variables.

Step 2: Define producer and consumer process.

Step 3: When the producer is called, perform a wait operation on semaphore associated with buffer and producer on time.

Step 4: If the consumer tries to access the buffer at the same time if its inherited, from doing so using semaphores control operations.

Step 5: The producer process produce items and the consumer process consume the items in the order in which the producer produces.

Step 6: Producer finish an item, it calls the signal operation on the semaphore to unlock the buffer and now the consumer can access the buffer.

Step 7: In the producer process after each item is produced a global variable counter is incremented by one.

Step 8: In the consumer process after the consumption of each item, the counter is decremented by one.

Step 9: All operations are performed by semaphore wait and signal process. Step 10: Display the items produced.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int mutex=1,full=0,empty=3,x=0;
main()
{
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("\n1.PRODUCER\n2.CONSUMER\n3.EXIT\n");
    while(1)
    {
        printf("\nENTER YOUR CHOICE\n");
        scanf("%d",&n);
        switch(n)
        {
            case 1:
```

```

if((mutex==1) && (empty!=0))
producer();
else
printf("BUFFER IS FULL");
break;
case 2:
if((mutex==1)&&(full!=0)) consumer();
else
printf("BUFFER IS EMPTY");
break;
case 3:
exit(0);
break;
}
}
getch();
}
int wait(int s)
{
return(--s);
}
int signal(int s)
{
return(++s);
}
void producer()
{
mutex=wait(mutex);
full=signal(full);
empty=wait(empty);
x++;
printf("\nproducer produces the item%d",x);
mutex=signal(mutex);
}
void consumer()
{
mutex=wait(mutex);
full=wait(full);
empty=signal(empty);
printf("\n consumer consumes item%d",x);
x--;
mutex=signal(mutex);
}

```

OUTPUT

- 1.PRODUCER
- 2.CONSUMER
- 3.EXIT

ENTER YOUR CHOICE 1
producer produces the item1 ENTER YOUR CHOICE
1
producer produces the item2 ENTER YOUR CHOICE
1
producer produces the item3 ENTER YOUR CHOICE
1
BUFFER IS FULL ENTER YOUR CHOICE 2
consumer consumes item3 ENTER YOUR CHOICE
2
consumer consumes item2 ENTER YOUR CHOICE
2
consumer consumes item1 ENTER YOUR CHOICE
2
BUFFER IS EMPTY ENTER YOUR CHOICE
3

CONCLUSION

Thus the program to implement producer-consumer problem is executed and the output is verified.

Experiment No: 6

Date of Conduction:

Roll No:

Experiment No. 6

Problem Statement: Implement all File Organization Techniques:

- a) Single level directory
- b) Two level
- c) Hierarchical
- d) DAG

Objectives: To learn implementation of file organization techniques.

Theory:

Ex:6a SINGLE LEVEL DIRECTORY

THEORY

The directory contains information about the files, including attributes, location and ownership. Sometimes the directories consist of subdirectories also. The directory is itself a file, owned by the o. s and accessible by various file management routines.

Single Level Directories: It is the simplest of all directory structures, in this the directory system having only one directory, it consisting of the all files. Sometimes it is said to be root directory. The following dig. Shows single level directory that contains four files (A, B, C, D) .

It has the simplicity and ability to locate files quickly. It is not used in the multi- user system , it is used on small embedded system .

AIM

Write a c program to implement single level directory

ALGORITHM

1. Start
2. Declare the number, names and size of the directories and file names
3. Get the values for the declared variables.
4. Display the files that are available in the directories
5. Stop

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<graphics.h>
void main( )
{
int gd=DETECT,gm,count,i,j,mid,cir_x;
char fname[10][20];
clrscr( );
initgraph(&gd,&gm,"c:\\tc\\bgi");
```

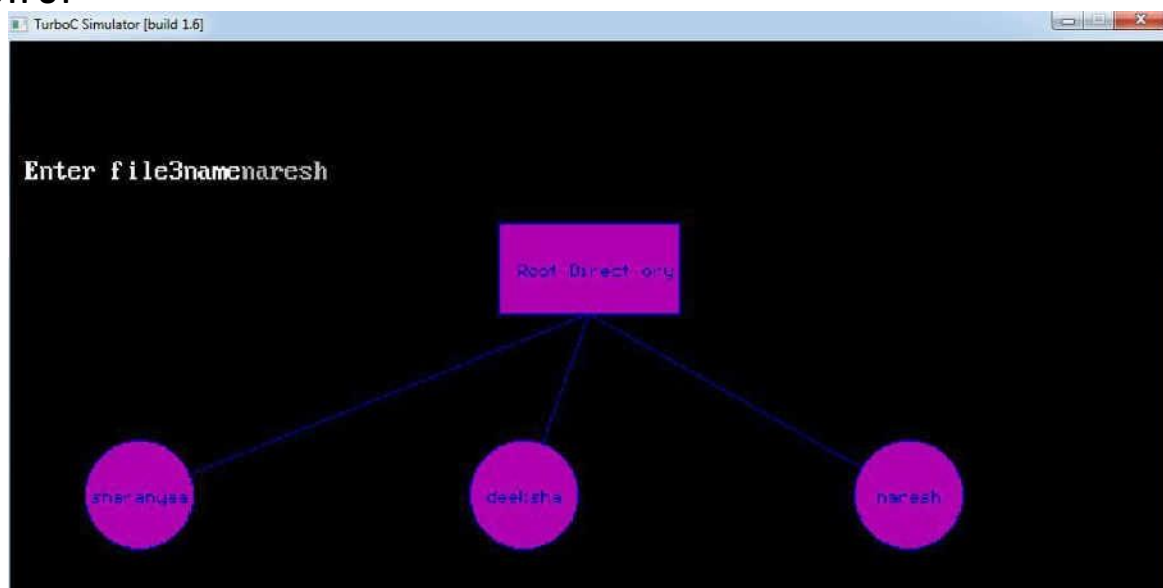


```

cleardevice( );
setbkcolor(BLACK) ;
puts( "Enter no of files u have?");
scanf("%d",&count);
for(i=0;i<count;i++)
{
cleardevice( );
setbkcolor(BLACK);
printf(" Enter file%dname",i+ 1);
scanf("%s",fname[ i] );
setfillstyle(1,MAGENTA);
mid=640/count ;
cir_x=mid/3 ;
bar3d(270,100,370,150,0,0);
settextstyle(2,0,4);
settextjustify(1,1);
outtextxy(320,125," Root Direct ory");
setcolor( BLUE) ;
for(j=0;j<=i;j++,cir_x+=mid)
{
line(320,150,cir_x,250);
fillellipse(cir_x,250,30,30);
outtextxy(cir_x,250,fname[j]);
}
getch( ) ;
}
}

```

OUTPUT



CONCLUSION

Thus the program for implementing single level directory has been completed.

EX: 6b TWO LEVEL DIRECTORY

THEORY

Two Level Directory: The problem in single level directory is different users may be accidentally using the same names for their files. To avoid this problem, each user need a private directory.

In this way names chosen by one user don't interface with names chosen by a different user. The following dig 2 - level directory

Here root directory is the first level directory it consisting of entries of user directory. User 1, User 2, User 3 are the user levels of directories. A, B, C are the files.

AIM

Write a C program to implement two level directory structures.

ALGORITHM

1. Start
2. Declare the number, names and size of the directories and subdirectories and file names.
3. Get the values for the declared variables.
4. Display the files that are available in the directories and subdirectories.
5. Stop.

PROGRAM

```
#include<stdio.h>
#include<graphics.h>
struct tree_element
{
char name[20];
int x,y,ftype,lx,rx,nc,level;
struct tree_element*link[5];
};
typedef struct tree_element node;
void main( )
{
int gd= DETECT,gm; node*root;
root=NULL;
clrscr( );
create(&root,0,"null",0,639,320);
clrscr( );
initgraph(&gd,&gm,"c:\\tc\\bgi");
display(root);
getch( );
closegraph( );
}
create(node**root,int lev,char *dname,int lx,int rx,int x)
{
int i,gap; if(*root==NULL)
```

```

{
(* root)=(node*)malloc(sizeof(node));
printf("enter name of dir/ file(under%s):",dname);
fflush(stdin);
gets((*root)->name);
if(lev==0 || lev==1)(*root)->ftype=1;
else(*root)->ftype=2;(* root)->level=lev;
(* root) ->y=50+lev*50;(*root)->x=x;
(* root)->lx=lx;
(*root)-> rx=rx;
for(i=0;i<5;i++){(*root)->link[i]=NULL;
if((*root)->ftype==1)
{
if(lev==0 || lev==1)
{
if((*root)->level==0)
printf(" How many users");
else
printf(" how many files");
printf("(for%s) :",(* root)->name);
scanf("%d",&(*root)->nc);
}
else(*root)->nc=0;
if((*root)->nc==0)
gap=rx-lx; else gap=(rx-lx)/(*root)->nc;
for(i=0;i<(*root)->nc;i++)
create(&((*root)->link[i]),lev+1,(* root )->name,lx+gap*i,lx+gap*i+gap,lx+gap* i+gap/2);
}
else(*root)->nc=0;
}
}
display(node*root)
{
int i;
settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
setcolor(14);
if(root!=NULL)
{
for(i=0;i<root->nc;i++)
{
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
}
if(root->ftype==1)
bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0);
else
fillellipse(root->x,root->y,20,20);
}
}

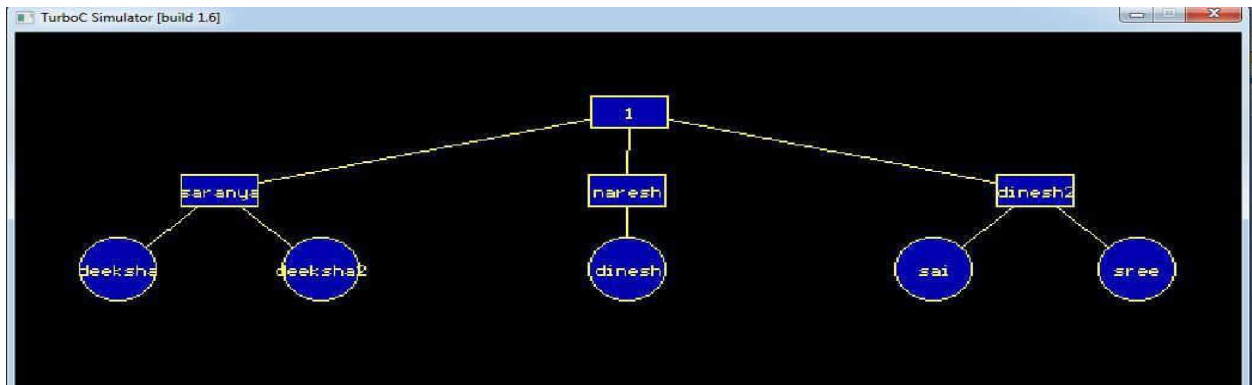
```

```

outtextxy(root->x,root->y,root->name);
for(i=0;i<root->nc;i++)
{
display(root->link[i]);
}}

```

OUTPUT



CONCLUSION

Thus the program to implement two level directory structure has been implemented.

EX:6C HIERARCHICAL STRUCTURE THEORY

Hierarchical Directory: The two level directories eliminate name conflicts among users but it is not satisfactory for users with a large no of files. To avoid this, create the subdirectory and load the same type of the files into the subdirectory. So, in this method each can have as many directories are needed.

AIM

Write a C program to implement Hierarchical Directory structure.

ALGORITHM

1. Start the program
2. Declare the number, names and size of the directories and subdirectories and file names.
3. Get the values for the declared variables.
4. Display the files that are available in the directories and subdirectories.
5. Stop

PROGRAM

```

#include<stdio.h>
#include<graphics.h> struct tree_element
{
char name[20];
int x,y,ftype,lx,rx,nc,level; struct tree_element*link[5];
};

```

```

typedef struct tree_element node;
void main( )
{
int gd=DETECT,gm;node*root;
root=NULL;
clrscr( );
create(&root,0,"root",0,639,320);
clrscr();
initgraph(&gd,&gm,"c:\\tc\\BGI");
display(root);getch();
closegraph();
}
create(node**root,int lev, char * dname, int lx , int rx, int x)
{
int i,gap; if(*root==NULL)

{
(*root)=(node*)malloc(sizeof(node));
printf("Enter name of dir/file(under%s):",dname);
fflush(stdin);
gets((*root)->name);
printf("enter 1 for Dir / 2 for file :");
scanf("%d",&(*root)->ftype);
(*root)->level=lev;
(*root)->y=50+lev*50;
(*root)->x=x;(*root)->lx=lx;
(*root)->rx=rx; for(i=0;i<5;i++)
(*root)->link[i]=NULL;
if((*root)->ftype==1)
{
printf("No of sub directories/files(for%s):",(*root)->name);
scanf("%d",&(*root)->nc);
if((*root)->nc==0)gap=rx-lx;
else
gap=(rx-lx)/(*root)->nc;
for(i=0;i<(*root)->nc;i++)
create(&((*root)->link[i]),lev+1,(*root)->name,lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2);
}
else(*root)->nc=0;
}
}
display(node*root)
{
int i; settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
setcolor(14);
if( root!=NULL)

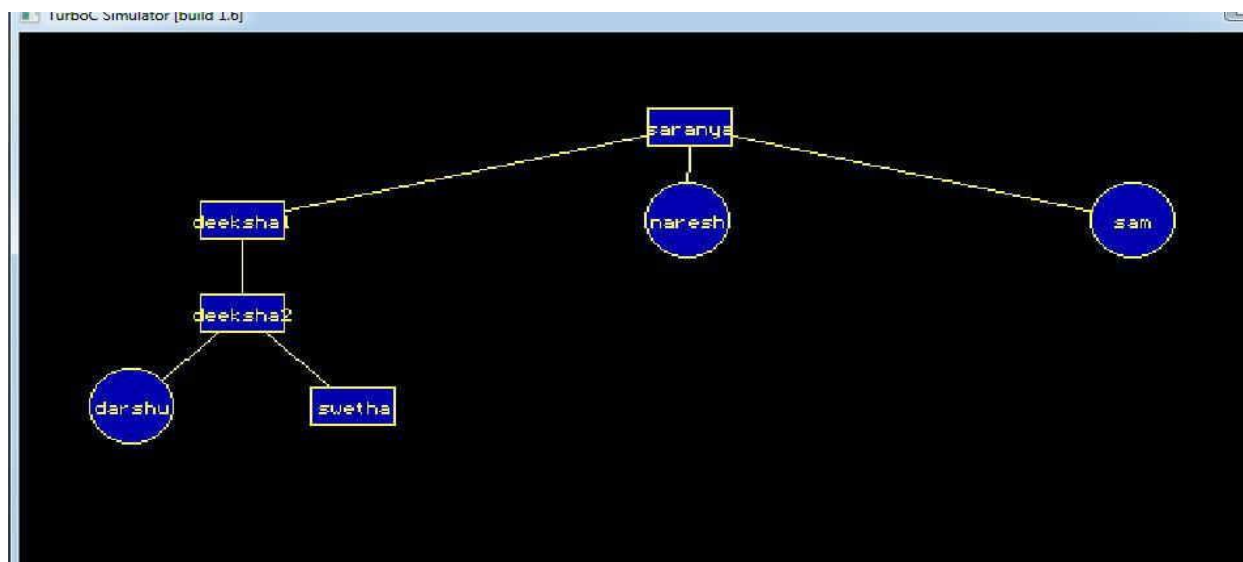
```

```

{
for(i=0;i<root->nc;i++)
{
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
}
if(root->ftype==1)
bar3d(root->x-20,root->y-10,root->x+20,root->y+10,0,0);
elsefillellipse(root->x,root->y,20,20);
outtextxy(root->x,root->y,root->name); for(i=0;i<root->nc;i++)
{
display(root->link[i]);
}
}
}
}

```

OUTPUT



CONCLUSION

Thus the program for implementing hierarchical directory structure has been completed.

Experiment No: 7

Date of Conduction:

Roll No:

Experiment No. 7

Problem Statement: Implement Bankers Algorithm for Dead Lock Avoidance.

Objectives: To learn deadlock avoidance using Bankers algorithm.

Theory:

Bankers algorithm is used to detect the occurrence of dead-lock and prevents it from occurrence through the safety algorithm embedded in it which deals with the information about safe sequence which when followed creates no problem to the flow of execution of the processes without any dead-lock occurrence. Thus, the efficiency of the CPU increases and for the same reason every CPU is inbuilt with a banker's algorithm.

AIM

Write a program for simulating a Bankers algorithm for Dead Lock Avoidance.

ALGORITHM

Step 1: Get the values of resources and processes. Step 2: Get the number of instance per resource.

Step 3: Get the avail value.

Step 4: After allocation find the need value.

Step 5: Check whether it's possible to allocate.

Step 6: If it is possible then the system is in safe state. Step 7: Else system is not in safety state.

Step 8: If the new request comes then check that the system is in safety or not if we allow the request.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main(){
int r[1][10],av[1][10];
int all[10][10],max[10][10],ne[10][10],w[10],safe[10];
int i=0,j=0,k=0,l=0,np=0,nr=0,count=0,cnt=0;
clrscr();
printf("enter the number of processes in a system");
scanf("%d",&np);
printf("enter the number of resources in a system");
scanf("%d",&nr);
for(i=1;i<=nr;i++){
printf("\n enter the number of instances of resource R%d ",i);
scanf("%d",&r[0][i]);
av[0][i]=r[0][i];
}
for(i=1;i<=np;i++)
for(j=1;j<=nr;j++)
all[i][j]=ne[i][j]=max[i][j]=w[i]=0;
printf(" \nEnter the allocation matrix");
for(i=1;i<=np;i++)
```



```

{ printf("\n");
for(j=1;j<=nr;j++)
{
scanf("%d",&all[i][j]);
av[0][j]=av[0][j]-all[i][j];
}
}
printf("\nEnter the maximum matrix");
for(i=1;i<=np;i++)
{ printf("\n");
for(j=1;j<=nr;j++)
{
scanf("%d",&max[i][j]);
}
}
for(i=1;i<=np;i++)
{
for(j=1;j<=nr;j++)
{
ne[i][j]=max[i][j]-all[i][j];
}
}
for(i=1;i<=np;i++)
{
printf("process P%d",i);
for(j=1;j<=nr;j++)
{
printf("\n allocated %d\t",all[i][j]);
printf("maximum %d\t",max[i][j]);
printf("need %d\t",ne[i][j]);
}
printf("\n      \n");
}
printf("\nAvailability");
for(i=1;i<=nr;i++)
printf("R%d %d\t",i,av[0][i]);
printf("\n      ");
printf("\n safe sequence");
for(count=1;count<=np;count++)
{

for(i=1;i<=np;i++)
{ cnt=0;

for(j=1;j<=nr;j++)
{
if(ne[i][j]<=av[0][j] && w[i]==0)
cnt++;

```

```

}
if(cnt==nr)
{
k++;
safe[k]=i;
for(l=1;l<=nr;l++)
av[0][l]=av[0][l]+all[i][l];
printf("\n P%d ",safe[k]);
printf("\t Availability");
for(l=1;l<=nr;l++)
printf("R%d %d\t",l,av[0][l]);
w[i]=1;
}
}
}
getch();
}

```

OUTPUT

```

Enter the maximum matrix
753
322

902
222

902
433
process P1
allocated 10    maximum 753    need 743
allocated 200  maximum 322    need 122
-----
process P2
allocated 302  maximum 902    need 600
allocated 211  maximum 222    need 11
-----
process P3
allocated 2     maximum 902    need 900
allocated 222   maximum 433    need 211
-----
Availability R1 -313    R2 -632

```

CONCLUSION

Hence we learned implementation of Bankers algorithm.

Experiment No: 8

Date of Conduction:

Roll No:

Experiment No. 8

Problem Statement: Implement an Algorithm for Dead Lock Detection.

Objectives: To write a C program to implement deadlock detection algorithm.

Theory:

ALGORITHM:

Step 1: Start the program

Step 2: Declare the necessary variable

Step 3: Get no. Of. process, resources, max & need matrix Step 4: Get the total resource and available resources

Step 5: Claim the deadlock occurred process Step 6: Display the result

Step 7: Stop the program

PROGRAM:

```
#include <stdio.h>
main()
{
int found,flag,l,p[4][5],tp,tr,c[4][5],i,j,k=1,m[5],r[5],a[5],temp[5],sum=0;
printf("Enter total no of processes");
scanf("%d",&tp);
printf("Enter total no of resources");
scanf("%d",&tr);
printf("Enter claim (Max. Need) matrix\n");
for(i=1;i<=tp;i++)
{
printf("process %d:\n",i);
for(j=1;j<=tr;j++)
scanf("%d",&c[i][j]);
}
printf("Enter allocation matrix\n");
for(i=1;i<=tp;i++)
{
printf("process %d:\n",i);
for(j=1;j<=tr;j++)
scanf("%d",&p[i][j]);
}
printf("Enter resource vector (Total resources):\n");
for(i=1;i<=tr;i++)
{
scanf("%d",&r[i]);
}
printf("Enter availability vector (available resources):\n");
for(i=1;i<=tr;i++)
{
scanf("%d",&a[i]);
}
```

```

temp[i]=a[i];
}
for(i=1;i<=tp;i++)
{
sum=0;
for(j=1;j<=tr;j++)
{
sum+=p[i][j];
}
if(sum==0)
{
m[k]=i; k++;
}
}
for(i=1;i<=tp;i++)
{
for(l=1;l<k;l++)
if(i!=m[l])
{flag=1;
for(j=1;j<=tr;j++)
if(c[i][j]<temp[j])
{
flag=0;
break;
}}
if(flag==1)
{
m[k]=i;
k++;
for(j=1;j<=tr;j++)
temp[j]+=p[i][j];
}
}
printf("deadlock causing processes are:");
for(j=1;j<=tp;j++)
{
found=0; for(i=1;i<k;i++)
{
if(j==m[i]) found=1;
}
if(found==0) printf("%d\t",j);
}
}

```

OUTPUT:

Enter total no. of processes : 4
Enter total no. of resources : 5

Enter claim (Max. Need) matrix : 0 1 0 0 1

0 0 1 0 1

0 0 0 0 1

1 0 1 0 1

Enter allocation matrix :

1 0 1 1 0

1 1 0 0 0

0 0 0 1 0

0 0 0 0 0

Enter resource vector (Total resources): 2 1 1 2 1

Enter availability vector (available resources): 0 0 0 0 1

deadlock causing processes are: 2 3

CONCLUSION

Thus the Program for deadlock detection algorithm was executed.

Experiment No: 9

Date of Conduction:

Roll No:

Experiment No. 9

Problem Statement: Implement all page replacement algorithms:

- FIFO
- LRU
- LFU

Theory:

The simpler page replacement algorithm is a FIFO algorithm. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. We can create a FIFO queue to hold all pages in memory. We replace the page at the head of the queue when a page is brought into memory; we insert it at the tail of the queue.

AIM

Write a c program to implement FIFO page replacement algorithm

ALGORITHM

Step 1: Get number of frames and number of pages Step 2: Declare the size with respect to page length

Step 3: Check the need of replacement from the page to memory

Step 4: Check the need of replacement from old page to new page in memory Step 5: Allocate the pages in to frames in First in first out order.

Step 6: Display the number of page faults.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int main()
{
    int i,j,n,a[50],frame[10],no,k,avail,count=0;
    printf("\n ENTER THE NUMBER OF PAGES:\n");
    scanf("%d",&n);
    printf("\n ENTER THE REF STRING :\n");
    for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
    printf("\n ENTER THE NUMBER OF FRAMES :");
    scanf("%d",&no); for(i=0;i<no;i++) frame[i]= -1;
    j=0;

    printf("\t ref string\t page frames\n");
    for(i=1;i<=n;i++)
    {
        if(frame[j]==a[i])
            printf("%d\t\t",a[i]); avail=0;
        for(k=0;k<no;k++)
            avail=1;
        if (avail==0)
```



```

{
frame[j]=a[i];
j=(j+1)%no;
count++;
for(k=0;k<no;k++)
printf("%d\t",frame[k]);
}
printf("\n");
}
printf("Page Fault Is %d",count);
return 0;
getch();
}

```

OUTPUT

ENTER THE NUMBER OF PAGES: 20

ENTER THE REF STRING :7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

ENTER THE NUMBER OF FRAMES :3

ref string	page frames
7	7 -1 -1
0	7 0 -1
1	7 0 1
2	2 0 1
0	
3	2 3 1
0	2 3 0
4	4 3 0
2	4 2 0
3	4 2 3
0	0 2 3
3	
2	
1	0 1 3
2	0 1 2
0	
1	
7	7 1 2
0	7 0 2
1	7 0 1

Page Fault Is 15

Conclusion

Thus the Program for FIFO page replacement algorithm is completed.

EX:9b IMPLEMENTATION OF LRU PAGE REPLACEMENT ALGORITHM**AIM**

Write a c program to implement LRU page replacement algorithm

ALGORITHM

Step 1: Get the number of pages and frame number Step 2: Get the values of reference string
 Step 3: Declare counter and stack
 Step 4: Select the least recently used page by counter value
 Step 5: Stack them according the selection and allocate the pages in to frames by selecting the page that has not been used for the longest period of time.
 Step 6: Display the number of page faults.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
main()
{
int q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20];
printf("Enter no of pages:"); scanf("%d",&n);
printf("Enter the reference string:");
for(i=0;i<n;i++)
scanf("%d",&p[i]);
printf("Enter no of frames:");
scanf("%d",&f);
q[k]=p[k];
printf("\n\t%d\n",q[k]);
c++;
k++;
for(i=1;i<n;i++)
{
c1=0;
for(j=0;j<f;j++)
{
if(p[i]!=q[j]) c1++;
}
if(c1==f)
{
c++;
if(k<f)
{
q[k]=p[i];

}
else
{
k++;
for(j=0;j<k;j++)
```

```

printf("\t%d",q[j]);
printf("\n");
for(r=0;r<f;r++)
{
c2[r]=0;
for(j=i-1;j<n;j--)
{
if(q[r]!=p[j])
c2[r]++;
else break;
}
}
for(r=0;r<f;r++)
b[r]=c2[r];
for(r=0;r<f;r++)
{
for(j=r;j<f;j++)
{
if(b[r]<b[j])
{
t=b[r]; b[r]=b[j];
b[j]=t;
}
}
}
for(r=0;r<f;r++)
{
if(c2[r]==b[0])
q[r]=p[i];
printf("\t%d",q[r]);
}
printf("\n");
}
}
printf("\nThe no of page faults is %d",c);
getch();
}

```

OUTPUT

Enter no of pages:10

Enter the reference string:7 5 9 4 3 7 9 6 2 1 Enter no of frames:3

7

7 5

7 5 9

4 5 9

4 3 9

4 3 7

9	3	7
9	6	7
9	6	2
1	6	2

The no of page faults is 10

CONCLUSION

Thus the Program for least recently used page replacement algorithm is completed.

EX: 9c OPTIMAL (LFU) PAGE REPLACEMENT ALGORITHM

AIM

Write a program to implement Optimal page replacement algorithm.

ALGORITHM

Step 1: Read the number of frames Step 2: Read the number of pages Step 3: Read the page numbers

Step 4: Initialize the values in frames to -1

Step 5: Allocate the pages in to frames by selecting the page that will not be used for the Longest period of time.

Step 6: Display the number of page faults.

PROGRAM

```
/*OPTIMAL(LFU) page replacement algorithm*/

#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],optcal[50],count=0;
int optvictim();
void main()
{
clrscr();
printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHM");
printf("\n.    ");
printf("\nEnter the no.of
frames"); scanf("%d",&nof);
printf("Enter the no.of reference string");
scanf("%d",&nor);
printf("Enter the reference string");
for(i=0;i<nor;i++)
scanf("%d",&ref[i]);
clrscr();
printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHM");
printf("\n.    ");
printf("\nThe given string");
printf("\n.    \n");
```

```

for(i=0;i<nor;i++)
printf("%4d",ref[i]);
for(i=0;i<nof;i++)
{
frm[i]=-1;
optcal[i]=0;
}
for(i=0;i<10;i++)
recent[i]=0;
printf("\n");
for(i=0;i<nor;i++)
{
flag=0;
printf("\n\tref no %d ->\t",ref[i]);
for(j=0;j<nof;j++)
{
if(frm[j]==ref[i])
{
flag=1; break;
}
}
if(flag==0)
{
count++;
if(count<=nof)
victim++;
else
victim=optvictim(i);
pf++;
frm[victim]=ref[i];
for(j=0;j<nof;j++)
printf("%4d",frm[j]);
}
}
printf("\n Number of page faults: %d",pf);
getch();
}
int optvictim(int index)
{
int i,j,temp,notfound;
for(i=0;i<nof;i++)
{
notfound=1;
for(j=index;j<nor;j++)
if(frm[i]==ref[j])
{
notfound=0;
optcal[i]=j;

```

```

break;
}
if(notfound==1)
return i;
}
temp=optcal[0];
for(i=1;i<nof;i++)
if(temp<optcal[i])
temp=optcal[i];
for(i=0;i<nof;i++)
if(frm[temp]==frm[i])
return i;
return 0;
}

```

OUTPUT

OPTIMAL PAGE REPLACEMENT ALGORITHM

Enter no.of Frames.3

Enter no.of reference string 6

Enter reference string..

6 5 4 2 3 1

OPTIMAL PAGE REPLACEMENT ALGORITHM

The given reference string:

..... 6 5 4 2 3 1

Reference NO 6-> 6 -1 -1

Reference NO 5-> 6 5 -1

Reference NO 4-> 6 5 4

Reference NO 2-> 2 5 4

Reference NO 3-> 2 3 4

Reference NO 1-> 2 3 1

No.of page faults.....6

Conclusion:

Thus the Program for LFU page replacement algorithm is completed

Experiment No: 10

Date of Conduction:

Roll No:

Experiment No. 10

Problem Statement: Implement Shared memory and IPC.

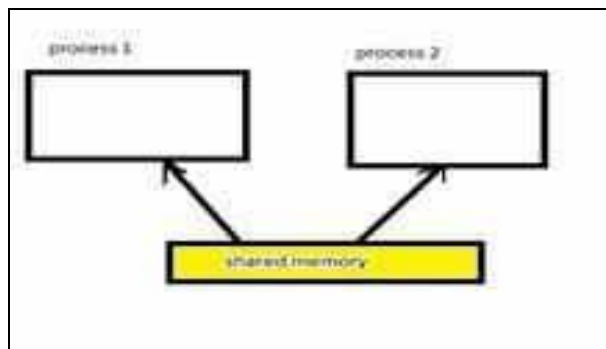
Objectives: To learn implementation of Inter Process Communication.

Theory:

A shared memory is an extra piece of memory that is attached to some address spaces for their owners to use. As a result, all of these processes share the same memory segment and have access to it. Consequently, race conditions may occur if memory accesses are not handled properly. The following figure shows two processes and their address spaces.

The yellow rectangle is a shared memory attached to both address spaces and both process 1 and process 2 can have access to this shared memory as if the shared memory is part of its own address space.

In some sense, the original address space is "extended" by attaching this shared memory.



AIM

Write a program for interprocess communication using shared memory.

ALGORITHM

Step 1: Create the child process using `fork()`

Step 2: Create the shared memory for parent process using `shmget()` system call

Step 3: Now allow the parent process to write in shared memory using `shmptr` pointer which is return type of `shmat()`

Step 4: Now attach the same shared memory to the child process

Step 5: The data in the shared memory is read by the child process using the `shptr` pointer. Now, detach and rebase the shared memory

PROGRAM

```
#include<sys/types.h>
#include<sys/shm.h>
#include<sys/ipc.h>
main()
{
```



```

int shmid;
key_t key=0*10;
shmid=shmget(key,100,IPC_CREAT|0666);
if(shmid<0)
printf("\nfirst SHMID failed\n");
else
printf("\n first SHMID succeeded id=%d\n",shmid);
shmid=shmget(key,101,IPC_CREAT|0666);
if(shmid<0)
printf("\nsecond SHMID failed\n"); else
printf("\n secondt SHMID succeeded id=%d\n",shmid);
shmid=shmget(key,90,IPC_CREAT|0666);
if(shmid<0)
printf("\nthird SHMID failed\n");
else
printf("\n third SHMID succeeded id=%d\n",shmid);
}

```

OUTPUT

```

[cse2@localhost ~]$ cc share.c
[cse2@localhost ~]$ ./a.out
first SHMID succeeded id=589833
secondt SHMID succeeded id=622604
third SHMID succeeded id=655373
[cse2@localhost ~]$

```

CONCLUSION

Thus the interprocess communication using shared memory was successfully executed.

Experiment No: 11

Date of Conduction:

Roll No:

Experiment No. 11

Problem Statement: Implement Paging Technique of memory management.

Objectives: Write a program to implement the Memory management policy- Paging.

Theory:

ALGORITHM

Step 1: Read all the necessary input from the keyboard.

Step 2: Pages - Logical memory is broken into fixed - sized blocks.

Step 3: Frames – Physical memory is broken into fixed – sized blocks.

Step 4: Calculate the physical address using the following $\text{Physical address} = (\text{Frame number} * \text{Frame size}) + \text{offset}$

Step 5: Display the physical address.

PROGRAM

```
/* Memory Allocation with Paging Technique */
#include <stdio.h>
#include <conio.h>
struct pstruct
{
    int fno;
    int pbit;
}ptable[10];
int pmsize, lmsize, psize, frame, page, ftable[20], frameno;
void info()
{
    printf("\n\nMEMORY MANAGEMENT USING PAGING\n\n");
    printf("\n\nEnter the Size of Physical memory: ");
    scanf("%d",&pmsize);
    printf("\n\nEnter the size of Logical memory: ");
    scanf("%d",&lmsize);
    printf("\n\nEnter the partition size: ");
    scanf("%d",&psize);
    frame = (int) pmsize/psize; page = (int) lmsize/psize;
    printf("\nThe physical memory is divided into %d no.of frames\n",frame);
    printf("\nThe Logical memory is divided into %d no.of pages",page);
}

void assign()
{
    int i;
    for (i=0;i<page;i++)
    {
        ptable[i].fno = -1;
        ptable[i].pbit= -1;
    }
}
```

```

    }
    for(i=0; i<frame;i++)
        ftable[i] =
32555; for
(i=0;i<page;i++)
{
    printf("\n\nEnter the Frame number where page %d must be placed: ",i);
    scanf("%d",&framenno);
    ftable[framenno] = i;
    if(ptable[i].pbit == -1)
    {
        ptable[i].fno = framenno;
        ptable[i].pbit = 1;
    }
}
getch();
printf("\n\nPAGE TABLE\n\n");
printf("PageAddress FrameNo.
PresenceBit\n\n"); for (i=0;i<page;i++)
printf("%d\t%d\t%d\n",i,ptable[i].fno,ptable[i].pbit);
printf("\n\n\nFRAME TABLE\n\n");
printf("FrameAddress PageNo\n\n");
for(i=0;i<frame;i++)
printf("%d\t%d\n",i,ftable[i]);
}
void cphyaddr()
{
    int laddr,paddr,disp,phyaddr,baddr;
    getch();
    // clrscr();
    printf("\n\n\nProcess to create the Physical Address\n\n");
    printf("\nEnter the Base Address: ");
    scanf("%d",&baddr); printf("\nEnter the Logical Address: ");
    scanf("%d",&laddr);
    paddr = laddr / psize; disp = laddr % psize;
    if(ptable[paddr].pbit == 1 )
        phyaddr = baddr + (ptable[paddr].fno*psize) + disp;
    printf("\nThe Physical Address where the instruction present: %d",phyaddr);
}
void main()
{
    clrscr();

    info();
    assign();
    cphyaddr();
    getch();
}

```

OUTPUT

MEMORY MANAGEMENT USING PAGING

Enter the Size of Physical memory: 16

Enter the size of Logical memory: 8

Enter the partition size: 2

The physical memory is divided into 8 no. of frames

The Logical memory is divided into 4 no.of pages

Enter the Frame number where page 0 must be placed: 5

Enter the Frame number where page 1 must be placed: 6

Enter the Frame number where page 2 must be placed: 7

Enter the Frame number where page 3 must be placed: 2

PAGE TABLE

PageAddress	FrameNo.	PresenceBit
0	5	1
1	6	1
2	7	1
3	2	1

FRAME TABLE

FrameAddress	PageNo
0	32555
1	32555
2	3
3	32555
4	32555
5	0
6	1
7	2

Process to create the Physical

Address Enter the Base Address:

1000

Enter the Logical Address: 3

The Physical Address where the instruction present: 1013

CONCLUSION

Thus the paging technique for memory management is implemented.

Experiment No: 12

Date of Conduction:

Roll No:

Experiment No. 12

Problem Statement: Implement Threading & Synchronization Applications.

Theory:

AIM

Write a c program to implement dining philosopher problem using threads

ALGORITHM

Step 1: Philosopher i is thinking.
 Step 2: Lock the left fork spoon.
 Step 3: Lock the right fork spoon.
 Step 4: Philosopher i is eating.
 Step 5: sleep
 Step 6: Release the left fork spoon.
 Step 7: Release the right fork spoon.
 Step 8: Philosopher i Finished eating.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
#include<semaphore.h>
void *func(int n);
pthread_t philosopher[5];
pthread_mutex_t chopstick[5];
int main()
{
  int i,k; void *msg;
  for(i=1;i<=5;i++)
  {
    k=pthread_mutex_init(&chopstick[i],NULL); if(k==-1)
    {
      printf("\n Mutex initialization failed");
      exit(1);
    }
  }
  for(i=1;i<=5;i++)
  {
    k=pthread_create(&philosopher[i],NULL,(void *)func,(int *)i);
    if(k!=0)
    {
      printf("\n Thread creation error \n");
      exit(1);
    }
  }
  for(i=1;i<=5;i++)
  {
    k=pthread_join(philosopher[i],&msg); if(k!=0)
    {
```

```

printf("\n Thread join failed \n"); exit(1);
}
}
for(i=1;i<=5;i++)
{
k=pthread_mutex_destroy(&chopstick[i]); if(k!=0)
{
printf("\n Mutex Destroyed \n");
exit(1);
}
}
return 0;
}
void *func(int n)
{
printf("\nPhilosopher %d is thinking ",n);
pthread_mutex_lock(&chopstick[n]); //when philosopher 5 is eating he takes fork 1 and fork 5
pthread_mutex_lock(&chopstick[(n+1)%5]);
printf("\nPhilosopher %d is eating ",n); sleep(3);
pthread_mutex_unlock(&chopstick[n]);
pthread_mutex_unlock(&chopstick[(n+1)%5]);
printf("\nPhilosopher %d Finished eating ",n);
}

```

OUTPUT:

```

[root@localhost ~]$ gcc -o c dining.c -pthread
[root@localhost ~]$ ./c
Philosopher 1 is thinking
Philosopher 1 is eating Philosopher
2 is thinking Philosopher 3 is
thinking Philosopher 3 is eating
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 1 Finished eating
Philosopher 3 Finished eating
Philosopher 4 is eating Philosopher
5 is eating Philosopher 2 is eating
Philosopher 4 Finished eating
Philosopher 5 Finished eating
Philosopher 2 Finished eating

```

CONCLUSION

Thus the implementation of threading and synchronization was executed.
