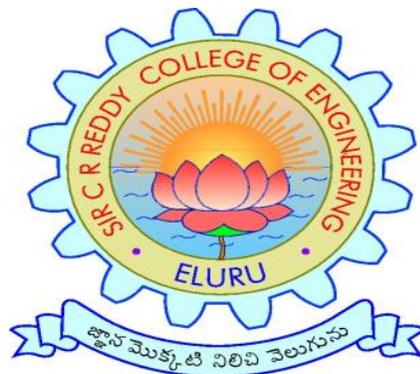


**SIR C R REDDY COLLEGE OF ENGINEERING, ELURU**  
**DEPARTMENT OF INFORMATION TECHNOLOGY**

**UNIX OPERATING SYSTEMS LAB MANUAL**

**A.Y. 2020-2021**



**CLASS: 2/4 B. Tech. I.T., II SEMESTER**

**(Approved by AICTE, Affiliated to JNTU Kakinada)**

## Experiment 1

### LINUX PROGRAMMING

1. a) Study of Unix/Linux general purpose utility command list: *man, who, cat, cd, cp, ps, ls, mv, rm, mkdir, rmdir, echo, more, date, time, kill, history, chmod, chown, finger, pwd, cal, logout, shutdown.*

1. **man** - view manual pages for UNIX commands

*Syntax: man [-s section] item*

*\$ man cat*

2. **who** - displays the list of users logged presently

*Syntax: who [option] ... [file][arg1]*

*\$ who*

3. **cat** - concatenate files and show contents to the standard output

*Syntax: cat [option] ...[file]*

*\$ cat > file1*

*hello*

4. **cd** - change directory

*Syntax: cd [option] directory*

*\$ cd dir1*

5. **cp** - copy files

*Syntax: cp [option] source destination*

*\$ cp file1 file2*

6. **ps** - Reports process status

*Syntax: Ps [options]*

*\$ ps*

7. **ls** - List files & directories

*Syntax: ls [option] [file]*

*\$ ls*

8. **mv** - Rename or move files & directories to another location

*Syntax: mv [option] source destination*

*\$ mv file1 file2*

9. **rm** - removes files & directories

*Syntax: rm [option] ...[file]*

*\$ rm file1*

10. **mkdir** - makes new directory

*Syntax: mkdir [option] directory*

*\$ mkdir dir1*

11. **rmdir** - removes directory

*Syntax: rmdir [option] directory*

*\$ rmdir dir1*

**12. echo** - displays a line of text to the standard output

*Syntax: \$ echo "Hello, World!"*

*\$ echo \$x*

**13. more** - basic pagination when viewing text files or parsing UNIX commands output

*Syntax: more [-options] [-num] [+pattern] [+linenum] [file\_name]*

*\$ more +/reset sample.txt*

*\$ more +30 sample.txt*

**14. date** - Show current date & time

*Syntax: date [+format]*

*\$ date +%d/%m/%y*

**15. time** - shows current time

*\$ time*

**16. kill** - terminates a specified process

*Syntax: Kill PID*

*\$ kill 4901*

**17. history** - displays history commands in the current session

*Syntax: history [options]*

*\$ history*

**18. chmod** - change file / directory access permissions

### Symbolic mode

*Syntax: chmod [u go a] [[+|=][mode]] file*

- The first optional parameter indicates who – this can be (u)ser, (g)roup, (o)thers or (a)ll
- The second optional parameter indicates opcode – this can be for adding (+), removing (-) or assigning (=) a permission.
- The third optional parameter indicates the mode – this can be (r)ead, (w)rite, or e(x)ecute.

*\$ chmod o-w file1*

### Numeric mode

*Syntax: chmod [mode] file*

- The mode is a combination of three digits – the first digit indicates the permission for the user, the second digit for the group, and the third digit for others.
- Each digit is computed by adding the associated permissions. Read permission is '4', write permission is '2' and execute permission is '1'.

*\$ chmod 777 file1*

**19. chown** - change file / directory ownership

*Syntax: chown [owner] [file]*

*\$ chown user2 file1*

**20. finger** - displays user info if login is known

*Syntax: finger username*

**21. pwd** - confirm current directory

*Syntax: pwd [option]*

*\$ pwd*

**22. cal** - displays calendar

*Syntax: cal [[month] year]*

*\$ cal 6 2019*

**23. logout** - logs off UNIX

*Syntax: Logout [options]*

*\$ logout*

**24. shutdown** - brings the system down in a secure way

*Syntax: shutdown [options]*

*\$ shutdown*

**b) Study of vi editor.**

- Visual editor for editing programs.
- The **vi** editor is the most popular and classic text editor in the Linux family.
- This editor enables you to edit lines in context with other lines in the file.
- Below, are some reasons which make it a widely used editor?
  - It is available in almost all Linux Distributions
  - It works the same across different platforms and Distributions
  - It requires very few resources.
  - It is more user-friendly than other editors such as the **ed** or the **ex**.
- We can use the **vi** editor to edit an existing file or to create a new file from scratch.
- We can also use this editor to just read a text file.
- An improved version of the vi editor which is called the **VIM** has also been made available now. Here, VIM stands for **Vi IM**proved.

**Starting the vi Editor**

To launch the **vi** Editor -Open the Terminal (CLI) and type

*vi <filename\_NEW> or <filename\_EXISTING>*

& if you specify an existing file, then the editor would open it for you to edit. Else, you can create a new file.

The following table lists out the basic commands to use the vi editor –

S. No	Command	Description
1	<b>vi filename</b>	Creates a new file if it already does not exist, otherwise opens an

		existing file.
2	<b>vi -R filename</b>	Opens an existing file in the read-only mode.

### Operation Modes

To work on **vi** editor, we need to understand its operation modes.

- **Command mode** – This mode enables you to perform administrative tasks such as saving the files, executing the commands, moving the cursor, cutting (yanking) and pasting the lines or words, as well as finding and replacing. In this mode, whatever you type is interpreted as a command.
- **Insert mode** – This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and placed in the file.

**vi** always starts in the **command mode**. To enter text, you must be in the insert mode for which simply type **i**. To come out of the insert mode, press the **Esc** key, which will take you back to the command mode.

### vi Editing commands

**Note:** You should be in the "**command mode**" to execute these commands.

**vi** editor is **case-sensitive** so make sure to type the commands in the right letter-case, otherwise you will end up making undesirable changes to the file.

S. No	Command	Description
1	<b>i</b>	Insert at cursor ( <b>goes into insert mode</b> )
2	<b>a</b>	Write after cursor ( <b>goes into insert mode</b> )
3	<b>A</b>	Write at the end of line ( <b>goes into insert mode</b> )
4	<b>ESC</b>	Terminate insert mode
5	<b>u</b>	Undo last change
6	<b>U</b>	Undo all changes to the entire line
7	<b>o</b>	Open a new line ( <b>goes into insert mode</b> )
8	<b>dd</b>	Delete line
9	<b>3dd</b>	Delete 3 lines.
10	<b>D</b>	Delete contents of line after the cursor
11	<b>C</b>	Delete contents of a line after the cursor and insert new text. Press ESC key to end insertion.
12	<b>dw</b>	Delete word
13	<b>4dw</b>	Delete 4 words
14	<b>cw</b>	Change word
15	<b>x</b>	Delete character at the cursor
16	<b>r</b>	Replace character
17	<b>R</b>	Overwrite characters from cursor onward
18	<b>s</b>	Substitute one character under cursor continue to insert

<b>19</b>	<b>S</b>	Substitute entire line and begin to insert at the beginning of the line
<b>20</b>	<b>~</b>	Change case of individual character

**Moving within a file**

**Note:** You need to be in the “**command mode**” to move within a file.

The default keys for navigation are mentioned below else; you can also use the arrow keys on the keyboard.

<b>S. No</b>	<b>Command</b>	<b>Description</b>
<b>1</b>	<b>k</b>	Move cursor up
<b>2</b>	<b>j</b>	Move cursor down
<b>3</b>	<b>h</b>	Move cursor left
<b>4</b>	<b>l</b>	Move cursor right

**Saving and Closing the file**

**Note:** You should be in the “**command mode**” to exit the editor and save changes to the file.

<b>S. No</b>	<b>Command</b>	<b>Description</b>
<b>1</b>	<b>Shift+zz</b>	Save the file and quit
<b>2</b>	<b>:w</b>	Save the file but keep it open
<b>3</b>	<b>:q</b>	Quit without saving
<b>4</b>	<b>:wq</b>	Save the file and quit

**c) Study of Bash shell, Bourne shell and C shell in Unix/Linux operating system.**

**What Is a Shell?**

A shell is a program that provides an interface between a user and an operating system (OS) kernel.

An OS starts a shell for each user when the user logs in or opens a terminal or console window.

A kernel is a program that:

- Controls all computer operations.
- Coordinates all executing utilities
- Ensures that executing utilities do not interfere with each other or consume all system resources.
- Schedules and manages all system processes.

By interfacing with a kernel, a shell provides a way for a user to execute utilities and programs.

**Types of Shell**

1. Bourne Shell (sh)

- The Bourne shell is one of the original shells, developed for UNIX computers by Stephen Bourne at AT&T Bell Labs in 1977.
- It offers features such as input and output redirection, shell scripting with string and integer variables, and condition testing and looping.
- Command full-path name is **/bin/sh** and **/sbin/sh**.
- Default prompt for a non-root user is **\$**.
- Default prompt for a root user is **#**.

2. C Shell (csh)

- The C shell was designed to allow users to write shell script programs using syntax very similar to that of the C programming language.
- The C shell is a UNIX shell created by Bill Joy.
- The C shell is a command processor typically run in a text window, allowing the user to type commands.
- Command full-path name is **/bin/csh**.
- Default prompt for a non-root user is hostname **%**.
- Default prompt for a root user is hostname **#**.

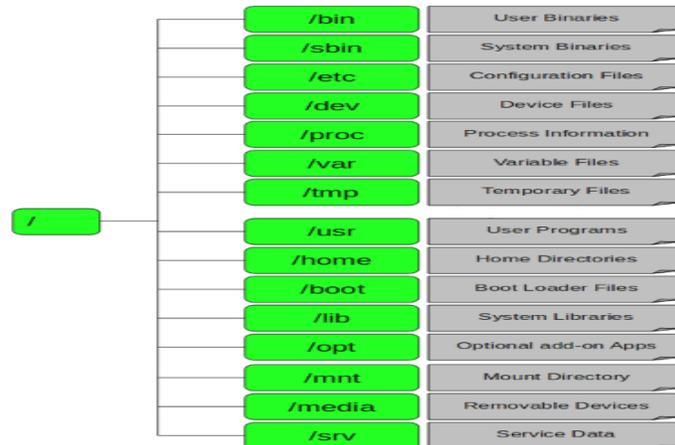
3. Bourne Again Shell (bash)

- The default Linux shell.
- Backward-compatible with the original sh UNIX shell.
- Bash is largely compatible with sh and incorporates useful features from the Korn shell ksh and the C shell csh.
- Command full-path name is **/bin/bash**.
- Default prompt for a non-root user is **bash-x.xx\$**. (Where *x.xx* indicates the shell version number. For example, *bash-3.50\$*)
- Root user default prompt is **bash-x.xx#**. (Where *x.xx* indicates the shell version number. For example, *bash-3.50\$#*)

d) **Study of Unix/Linux files system (tree structure).**

**Unix/Linux files system (tree structure)**

- UNIX file system is a logical method of **organizing and storing** large amounts of information in a way that makes it easy to manage.
- All data in UNIX is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the file system.
- A file is a smallest unit in which the information is stored.
- Files in Unix System are organized into multi-level hierarchy structure known as a directory tree. At the very top of the file system is a directory called “root” which is represented by a “/”. All other files are “descendants” of root.



### 1. / – Root

- Every single file and directory starts from the root directory.
- Only root user has write privilege under this directory.
- Please note that /root is root user’s home directory, which is not same as /.

### 2. /bin – User Binaries

- Contains binary executables.
- Common linux commands you need to use in single-user modes are located under this directory.
- Commands used by all the users of the system are located here.
- For example: ps, ls, ping, grep, cp.

### 3. /sbin – System Binaries

- Just like /bin, /sbin also contains binary executables.
- But, the linux commands located under this directory are used typically by system administrator, for system maintenance purpose.
- For example: iptables, reboot, fdisk, ifconfig, swapon

### 4. /etc – Configuration Files

- Contains configuration files required by all programs.
- This also contains startup and shutdown shell scripts used to start/stop individual programs.
- For example: /etc/resolv.conf, /etc/logrotate.conf

### 5. /dev – Device Files

- Contains device files.
- These include terminal devices, usb, or any device attached to the system.
- For example: /dev/tty1, /dev/usbmon0

### 6. /proc – Process Information

- Contains information about system process.
- This is a pseudo filesystem contains information about running process. For example: /proc/{pid} directory contains information about the process with that particular pid.

- This is a virtual filesystem with text information about system resources. For example:  
/proc/uptime

### **7. /var – Variable Files**

- var stands for variable files.
- Content of the files that are expected to grow can be found under this directory.
- This includes — system log files (/var/log); packages and database files (/var/lib); emails (/var/mail); print queues (/var/spool); lock files (/var/lock); temp files needed across reboots (/var/tmp);

### **8. /tmp – Temporary Files**

- Directory that contains temporary files created by system and users.
- Files under this directory are deleted when system is rebooted.

### **9. /usr – User Programs**

- Contains binaries, libraries, documentation, and source-code for second level programs.
- /usr/bin contains binary files for user programs. If you can't find a user binary under /bin, look under /usr/bin. For example: at, awk, cc, less, scp
- /usr/sbin contains binary files for system administrators. If you can't find a system binary under /sbin, look under /usr/sbin. For example: atd, cron, sshd, useradd, userdel
- /usr/lib contains libraries for /usr/bin and /usr/sbin
- /usr/local contains users programs that you install from source. For example, when you install apache from source, it goes under /usr/local/apache2

### **10. /home – Home Directories**

- Home directories for all users to store their personal files.
- For example: /home/john, /home/nikita

### **11. /boot – Boot Loader Files**

- Contains boot loader related files.
- Kernel initrd, vmlinuz, grub files are located under /boot
- For example: initrd.img-2.6.32-24-generic, vmlinuz-2.6.32-24-generic

### **12. /lib – System Libraries**

- Contains library files that supports the binaries located under /bin and /sbin
- Library filenames are either ld\* or lib\*.so.\*
- For example: ld-2.11.1.so, libncurses.so.5.7

### **13. /opt – Optional add-on Applications**

- opt stands for optional.
- Contains add-on applications from individual vendors.
- Add-on applications should be installed under either /opt/ or /opt/ sub-directory.

### **14. /mnt – Mount Directory**

- Temporary mount directory where sysadmins can mount filesystems.

### **15. /media – Removable Media Devices**

- Temporary mount directory for removable devices.
- For examples, /media/cdrom for CD-ROM; /media/floppy for floppy drives; /media/cdrecorder for CD writer

## 16. /srv – Service Data

- srv stands for service.
- Contains server specific services related data.
- For example, /srv/cvs contains CVS related data.

### e) Study of **.bashrc**, **/etc/bashrc** and Environment variables.

#### **.bashrc:**

A **.bashrc** file is automatically executed when new terminal (shell) is opened.

*Purpose of .bashrc file:*

- You can export environment variables(So there is no need to export environment variable every time)
- You can define aliases
- You can provide the path for cross compiler
- You can add your own script which can start automatically whenever new shell is opened.
- You can change the history length

#### **/etc/bashrc:**

A **.bashrc** file exists in your home directory. This file is meant for setting command aliases and functions used by bash shell users.

The **/etc/bashrc** for Red Hat and **/etc/bash.bashrc** in Ubuntu is the system wide version of **.bashrc**.

Interestingly enough in the Red Hat implementation the **/etc/bashrc** also executes the shell scripts within **/etc/profile.d** but only if the users shell is an Interactive Shell (aka Login Shell)

In Linux you can have two types of login shells, Interactive Shells and Non-Interactive Shells. An Interactive shell is used where a user can interact with the shell, i.e. your typical bash prompt. Whereas a non-Interactive shell is used when a user cannot interact with the shell, i.e. a bash scripts execution.

The **/etc/bashrc** is executed for both interactive and non-interactive shells

```
# grep TEST /etc/bash.bashrc
export TESTBASHRC=1
```

#### **Environment Variables:**

Linux environment variables act as placeholders for information stored within the system that passes data to programs launched in shells or sub-shells.

Admin's have the ability to modify environment variables to fit personal or larger group needs of users within their environments. As you'll notice below, admins can alter the hostname, command-line prompt, coloring in shells for text, and various other environment variables to better suit user preference.

**Commands for Environment Variables**

1. *env* – The command lists all of the environment variables in the shell.
2. *printenv* – The command prints all (if no environment variable is specified) of environment variables and definitions of the current environment.
3. *set* – The command assigns or defines an environment variable.
4. *unset* – The command deletes the environment variable.
5. *export* – The command exports the value of the newly assigned environment variable.

**Persistent and Non-persistent Environment Variables**

When modifying environment variables in your current shell, those variables remain non-persistent. The changes stay temporary and vanish once you log out of the shell.

You can modify the variables to stay persistent by editing the bash configuration files. After you logout of the current shell, those changes remain intact and permanent for the user(s) or groups.

**COMMON ENVIRONMENT VARIABLES**

Environment Variable	Description
DISPLAY	Names the display if running a graphical environment.
EDITOR	Names the preferred text editor.
HOME	Names the shell program.
SHELL	Displays the pathname of the home directory.
LANG	Defines the language character set.
OLD_PWD	Displays the previous working directory.
PAGER	Names the program used for paging output.
PATH	Displays a colon separated list of directories the user searched for when entering names of executable programs.
PS1	"Prompt string 1" (PS1) defines prompt content of shell.
PWD	Prints the current working directory.
TERM	Names the terminal type.
TZ	Displays the time zone. Most Linux/Unix-like systems maintain internal clock according to Coordinated Universal Time (UTC).
USER	Displays your username.
BROWSER	Displays the path to the default web browser.
MAIL	Displays the path to the current user's mailbox.
LS_COLORS	Defines color codes for coloring ls command output.
HOSTNAME	Displays computer's hostname.
SHELLOPTS	Displays shell options defined with the set command.
HISTSIZE	Displays number of command history lines allotted for use.
HISTFILESIZE	Displays number of lines in command-line history.
BASH_VERSION	Displays the version of bash shell used.

## Experiment 2

**Write a C program that makes a copy of a file using standard I/O, and system calls**

### Program:

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
void typefile (char *filename)
{
int fd, nread;
char buf[1024];
fd = open (filename, O_RDONLY);
if (fd == -1) {
perror (filename);
return;
}
while ((nread = read (fd, buf, sizeof (buf))) > 0)
write (1, buf, nread);
close (fd);
}
int main (int argc, char **argv)
{
int argno;
for (argno = 1; argno<argc; argno++)
typefile (argv[argno]);
}
```

### **OUTPUT:**

```
$cc linux2.c
$cat>file1
Hi hello
$./a.out file1
Hi hello
```

**Experiment 3**

**Write a C program to emulate the UNIX ls -l command.**

**Program:**

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>

int main()
{
int pid;           //process id
pid = fork();     //create another process
if ( pid< 0 )
{                 //fail
printf("\n Fork failed\n");
exit (-1);
}
else if ( pid == 0 )
{                 //child
execlp ( "/bin/ls", "ls", "-l", NULL ); //execute ls
}
else
{                 //parent
wait (NULL);     //wait for child
printf("\n child complete\n");
}
}
```

**OUTPUT:**

```
$cc linux3.c
```

```
$/a.out
```

```
-rw-r- - r- -file1
```

```
-rw-r- - r- -file2
```

```
-rw-r- - r- -file3
```

```
-rw-r- - r- -file4
```

```
Child completed
```

**Experiment 4**

**Write a C program that illustrates how to execute two commands concurrently with a command pipe.** Ex: - ls -l | sort

**Program:**

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
int main()
{
int pfd[2];
char buf[30];
if(pipe(pfd)==-1)
{
perror("&quot;pipe failed&quot;");
exit(1);
}
if(!fork())
{
close(1);
dup(pfd[1]);
system("&quot;ls -l&quot;");
}
else
{
printf("&quot;parent reading from pipe \n&quot;");
while(read(pfd[0],buf,80))
printf("&quot;%s \n&quot;",buf);
}
}
```

**Output:**

```
parent reading from pipe
total 16
-rwxr-xr-x 1 14100 14100 9082 Jun 23 04:40 a.out
-rwxrwxrwx 1 root root
t 693 Jun 23 04:40 main.
```

**Experiment 5****Simulate the following CPU scheduling algorithms**a) **Round Robin****Program:**

```
#include<stdio.h>
#include<conio.h>
main()
{
int st[10],bt[10],wt[10],tat[10],n,tq;
int i,count=0,swt=0,stat=0,temp,sq=0;
float awt=0.0,atat=0.0;
clrscr();
printf("Enter number of processes:");
scanf("%d",&n);
printf("Enter burst time for sequences:");
for(i=0;i<n;i++)
{
scanf("%d",&bt[i]);
st[i]=bt[i];
}
printf("Enter time quantum:");
scanf("%d",&tq);
while(1)
{
for(i=0,count=0;i<n;i++)
{
temp=tq;
if(st[i]==0)
{
count++;
continue;
}
if(st[i]>tq)
st[i]=st[i]-tq;
else
if(st[i]>=0)
```

```

{
temp=st[i];
st[i]=0;
}
sq=sq+temp;
tat[i]=sq;
}
if(n==count)
break;
}
for(i=0;i<n;i++)
{
wt[i]=tat[i]-bt[i];
swt=swt+wt[i];
stat=stat+tat[i];
}
awt=(float)swt/n;
atat=(float)stat/n;
printf("Process_no \t Burst time \t Wait time \t Turn around time\n");
for(i=0;i<n;i++)
printf("%d \t%d \t%d \t%d\n",i+1,bt[i],wt[i],tat[i]);
printf("Avg wait time is %f\n Avg turn around time is %f",awt,atat);
getch();
}

```

**OUTPUT:**

Enter number of processes: 3

Enter burst time for sequences: 24 3 3

Enter time quantum: 4

Process_no	Burst time	Wait time	Turnaround time
1	24	6	30
2	3	4	7
3	3	7	10

Avg wait time is 5.666667

Avg turn around time is 15.666667

b) **SJF (Shortest Job First)****Program:**

```
#include<conio.h>
#include<stdio.h>
void main()
{
int i, j, n, process[10], total=0, wtime[10], ptime[10], temp, ptemp;
float avg=0;
clrscr();
printf("\nEnter number of Processes:");
scanf("%d", &n);
for(i=0;i<n;i++)
{
printf("\nEnter Process %d ID:",i+1);
scanf("%d", &process[i]);
printf("\nEnter Process %d Time:",i+1);
scanf("%d",&ptime[i]);
}
for(i=0;i<n-1;i++)
{
for(j=i+1;j<n;j++)
{
if(ptime[i]>ptime[j])
{
temp = ptime[i];
ptime[i] = ptime[j];
ptime[j] = temp;
ptemp = process[i];
process[i] = process[j];
process[j] = ptemp;
}
}
}
wtime[0]=0;
for(i=1;i<n;i++)
{
```

```
wtime[i]=wtime[i-1]+ptime[i-1];
total=total+wtime[i];
}
avg=(float)total/n;
printf("\nP_ID\t P_TIME\t W_TIME\n");
for(i=0;i<n;i++)
printf("%d\t %d\t %d\n",process[i],ptime[i],wtime[i]);
printf("\nTotal Waiting Time: %d \nAverage Waiting Time: %f", total, avg);
getch();
}
```

**OUTPUT:**

Enter number of Processes: 4

Enter Process 1 ID: 1

Enter Process 1 Time: 6

Enter Process 2 ID: 2

Enter Process 2 Time: 8

Enter Process 3 ID: 3

Enter Process 3 Time: 7

Enter Process 4 ID: 4

Enter Process 4 Time: 3

<b>P_I D</b>	<b>P_TIM E</b>	<b>W_TIM E</b>
4	3	0
1	6	3
3	7	9
2	8	16

Total Waiting Time: 28

Average Waiting Time: 7.000000

c) **FCFS (First Come First Serve)**

Program:

```

#include<stdio.h>
void main()
{
int i,n,sum,wt,tat,twt,ttat;
int t[10];
float awt,atat;
clrscr();
printf("Enter number of processors:\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\n Enter the Burst Time of the process %d",i+1);
scanf("\n %d",&t[i]);
}
printf("\n\n FIRST COME FIRST SERVE SCHEDULING ALGORITHM \n");
printf("\n Process ID \t Waiting Time \t Turn Around Time \n");
printf("1 \t\t 0 \t\t %d \n",t[0]);
sum=0;
twt=0;
ttat=t[0];
for(i=1;i<n;i++)
{
sum+=t[i-1];
wt=sum;
tat=sum+t[i];
twt=twt+wt;
ttat=ttat+tat;
printf("\n %d \t\t %d \t\t %d",i+1,wt,tat);
printf("\n\n");
}
awt=(float)twt/n;
atat=(float)ttat/n;
printf("\n Average Waiting Time %4.2f",awt);
getch();

```

}

**OUTPUT:**

Enter number of processors:

3

Enter the Burst Time of the process: 124

Enter the Burst Time of the process: 23

Enter the Burst Time of the process: 33

**FIRST COME FIRST SERVE SCHEDULING ALGORITHM**

Process ID	Waiting Time	Turn Around Time
1	0	24
2	24	27
3	27	30

Average Waiting Time: 17.00

Average Turnaround Time: 27.00

**d.priority**

**Program:**

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i,j,n,tat[10],wt[10],bt[10],pid[10],pr[10],t,twt=0,ttat=0;
float awt,atat;
clrscr();
printf("\n-----PRIORITY SCHEDULING-----\n");
printf("Enter the No of Process: ");
scanf("%d", &n);
for (i=0;i<n;i++)
{
pid[i] = i;
printf("Enter the Burst time of Pid_%d : ",i);
scanf("%d",&bt[i]);
printf("Enter the Priority of Pid_%d : ",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;
}
}
tat[0] = bt[0];
wt[0] = 0;
for (i=1;i<n;i++)
{
wt[i] = wt[i-1] + bt[i-1];
tat[i] = wt[i] + bt[i];
}
printf("\n-----\n");
printf("Pid\t Priority\tBurst time\t WaitingTime\tTurnArroundTime\n");
printf("\n-----\n");
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]);
}
for(i=0;i<n;i++)
{
ttat = ttat+tat[i];
twt = twt + wt[i];
}

```

```

awt = (float)tw / n;
atat = (float)ttat / n;
printf("\n\nAvg.Waiting Time: %f\nAvg.Turn Around Time: %f\n",awt,atat);
getch();
}

```

**OUTPUT:**

-----**PRIORITY SCHEDULING**-----

Enter the No of Process: 5  
Enter the Burst time of Pid\_0: 10  
Enter the Priority of Pid\_0: 3  
Enter the Burst time of Pid\_1: 1  
Enter the Priority of Pid\_1: 1  
Enter the Burst time of Pid\_2: 2  
Enter the Priority of Pid\_2: 4  
Enter the Burst time of Pid\_3: 1  
Enter the Priority of Pid\_3: 5  
Enter the Burst time of Pid\_4: 5  
Enter the Priority of Pid\_4: 2

<b>Pid</b>	<b>Priority</b>	<b>Burst time</b>	<b>WaitingTime</b>	<b>TurnArroundTime</b>
1	1	1	0	1
4	2	5	1	6
0	3	10	6	16
2	4	2	16	18
3	5	1	18	19

Avg.Waiting Time: 8.200000  
Avg.Turn around Time: 12.000000

**Experiment 6:**

Multiprogramming-Memory management-Implementation of fork (), wait (), exec() and exit (), System calls

**Program:**

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h> /* for fork */
#include <sys/types.h> /* for pid_t */
#include <sys/wait.h> /* for wait */
int main(int argc,char** argv)
{ /*Spawn a child to run the program.*/
pid_t pid=fork();
if (pid==0)
{ /* child process */
execv("/bin/ls",argv);
exit(127); /* only if execv fails */

}
else
{ /* pid!=0; parent process */
printf("\nWaiting Child process to finish");
//waitpid(pid,0,0); /* wait for child to exit */wait(NULL);
}
printf("\nExiting main process\n");
return 0;
}
```

Expected Input/Output:

\$os509 \$cc multiprogramming.c

\$os509 \$./a.out

Total 28

rw-rw-rx 1 staff2 staff2 7317 aug14 09:22 a.out

-

Rw-rw-rx x 1 staff2 staff2 7317 aug14 09:16 multiprogramming

-

rw-rw-rx 1 staff2 staff2 361 aug16 09:20 multiprogramming.c

-

rw-rw-rx 1 staff2 staff2 359 aug14 09:16 multiprogramming.c

-

rw-rw-rx 1 staff2 staff2 1320 aug14 09:18 multiprogramming.c

Waiting until child process is finished

End of main() process

**Experiment 7:**

Multiprogramming with a fixed number of tasks (MFT)

**Program:**

```

#include<stdio.h>

main()
{
int ms, bs, nob, ef,n, mp[10],tif=0;
int i,p=0;

printf("Enter the total memory available (in Bytes) -- ");
scanf("%d",&ms);
printf("Enter the block size (in Bytes) -- ");
scanf("%d", &bs);
nob=ms/bs;
ef= ms-nob*bs;

printf("\nEnter the number of processes -- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter memory required for process %d (in Bytes)-- ",i+1);
scanf("%d",&mp[i]);
}
printf("\nNo. of Blocks available in memory -- %d",nob);
printf("\n\nPROCESS\tMEMORY REQUIRED\t ALLOCATED\tINTERNAL
FRAGMENTATION");
for(i=0;i<n && p<nob;i++)
{
printf("\n %d\t\t%d",i+1,mp[i]);
if(mp[i] > bs)
printf("\t\tNO\t\t---");
else
{
printf("\t\tYES\t\t%d",bs-mp[i]);
tif = tif + bs-mp[i];
p++;
}
}
if(i<n)
printf("\nMemory is Full, Remaining Processes cannot be accomodated");
printf("\n\nTotal Internal Fragmentation is %d",tif+ef);

}

```

**Output:**

Enter the total memory available (in Bytes) -- 100

Enter the block size (in Bytes) -- 25

Enter the number of processes -- 4

Enter memory required for process 1 (in Bytes)-- 20

Enter memory required for process 2 (in Bytes)-- 15

Enter memory required for process 3 (in Bytes)-- 20

Enter memory required for process 4 (in Bytes)-- 10

No. of Blocks available in memory -- 4

PROCESS	MEMORY REQUIRED	ALLOCATED	INTERNAL FRAGMENTATION
1	20	YES	5
2	15	YES	10
3	20	YES	5
4	10	YES	15

Total Internal Fragmentation is 35

**b) Multiprogramming with a variable number of tasks (MVT )**

**Program:**

```
#include<stdio.h>
main()
{
int ms,mp[10],i, temp,n=0;
char ch = 'y';
printf("\nEnter the total memory available (in Bytes)-- ");
scanf("%d",&ms);    50
10 temp=ms;
for(i=0;ch=='y';i++,n++) 2,2
{
printf("\nEnter memory required for process %d (in Bytes) -- ",i+1);
scanf("%d",&mp[i]);  25,15
if(mp[i]<=temp)  15<=25
{
printf("\nMemory is allocated for Process %d ",i+1);
temp = temp - mp[i];
}
else
{
printf("\nMemory is Full");
break;
}
}
printf("\nDo you want to continue(y/n) -- ");
```

```
scanf(" %c", &ch);
}
printf("\n\nTotal Memory Available -- %d", ms);
printf("\n\n\tPROCESS\t\tMEMORY ALLOCATED ");
for(i=0;i<n;i++)
printf("\n \t%d\t\t%d",i+1,mp[i]);
printf("\n\nTotal Memory Allocated is %d",ms-temp);
printf("\n\nTotal External Fragmentation is %d",temp);

}
```

**Output:**

Enter the total memory available (in Bytes)-- 105

Enter memory required for process 1 (in Bytes) -- 25

Memory is allocated for Process 1  
Do you want to continue(y/n) -- y

Enter memory required for process 2 (in Bytes) -- 19

Memory is allocated for Process 2  
Do you want to continue(y/n) -- y

Enter memory required for process 3 (in Bytes) -- 50

Memory is allocated for Process 3  
Do you want to continue(y/n) -- y

Enter memory required for process 4 (in Bytes) -- 100

Memory is Full

Total Memory Available -- 105

PROCESS	MEMORY ALLOCATED
1	25
2	19
3	50

Total Memory Allocated is 94  
Total External Fragmentation is 11

**Experiment 8:  
Simulate Bankers Algorithm for Dead Lock Avoidance****Program:**

```
#include<stdio.h>
int max[100][100];
int alloc[100][100];
int need[100][100];
int avail[100];
int n,r;
void input();
void show();
void cal();
int main()
{
int i,j;
printf("***** Banker's Algo *****\n");
input();
show();
cal();
return 0;
}
void input()
{
int i,j;
printf("Enter the no of Processes\t");
scanf("%d",&n);
printf("Enter the no of resources instances\t");
scanf("%d",&r);
printf("Enter the Max Matrix\n");
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
scanf("%d",&max[i][j]);
}
}
printf("Enter the Allocation Matrix\n");
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
scanf("%d",&alloc[i][j]);
}
}
```

```
}
printf("Enter the available Resources\n");
for(j=0;j<r;j++)
{
scanf("%d",&avail[j]);
}
}
void show()
{
int i,j;
printf("Process\t Allocation\t Max\t Available\t");
for(i=0;i<n;i++)
{
printf("\nP%d\t ",i+1);
for(j=0;j<r;j++)
{
printf("%d ",alloc[i][j]);
}
printf("\t");
for(j=0;j<r;j++)
{
printf("%d ",max[i][j]);
}
printf("\t");
if(i==0)
{
for(j=0;j<r;j++)
printf("%d ",avail[j]);
}
}
}

void cal()
{
int finish[100],temp,need[100][100],flag=1,k,c1=0;
int safe[100];
int i,j;
for(i=0;i<n;i++)
{
finish[i]=0;
}
//find need matrix
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
```

```

{
need[i][j]=max[i][j]-alloc[i][j];
}
}
printf("\n");
while(flag)
{
flag=0;
for(i=0;i<n;i++)
{
    int c=0;
    for(j=0;j<r;j++)
    {
        if((finish[i]==0)&&(need[i][j]<=avail[j]))
        {
            c++;
            if(c==r)
            {
                for(k=0;k<r;k++)
                {
                    avail[k]+=alloc[i][j];
                    finish[i]=1;
                    flag=1;
                }
            }
        }
        printf("P%d->",i);
        if(finish[i]==1)
        {
            i=n;
        }
    }
}
}
}
for(i=0;i<n;i++)
{
if(finish[i]==1)
{
c1++;
}
else
{

```

```

printf("P%d->",i);
}

}
if(c1==n)
{
printf("\n The system is in safe state");
}
else
{
printf("\n Process are in dead lock");
printf("\n System is in unsafe state");
}
}
}

```

**OUTPUT:**

```

Enter the no of processes 5
Enter the no of resources instances 3
Enter the max matrix
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3
Enter the allocation matrix
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Enter available resources 3 2 2
P1->p3->p4->p2->p0->

```

The system is in safe state

**Experiment 9:  
Simulate Bankers Algorithm for Dead Lock Prevention  
Program:**

```

#include< stdio.h>

void main()
{
int allocated[15][15],max[15][15],need[15][15],avail[15],tres[15],work[15],flag[15];
int pno,rno,i,j,prc,count,t,total;
count=0;

```

```
printf("\n Enter number of process:");
scanf("%d",&pno);
printf("\n Enter number of resources:");
scanf("%d",&rno);
for(i=1;i<=pno;i++)
{
flag[i]=0;
}
printf("\n Enter total numbers of each resources:");
for(i=1;i<= rno;i++)
scanf("%d",&tres[i]);
printf("\n Enter Max resources for each process:");
for(i=1;i<= pno;i++)
{
printf("\n for process %d:",i);
for(j=1;j<= rno;j++)
scanf("%d",&max[i][j]);
}
printf("\n Enter allocated resources for each process:");
for(i=1;i<= pno;i++)
{
printf("\n for process %d:",i);
for(j=1;j<= rno;j++)
scanf("%d",&allocated[i][j]);
}

printf("\n available resources:\n");
for(j=1;j<= rno;j++)
{
avail[j]=0;
total=0;
for(i=1;i<= pno;i++)
{
total+=allocated[i][j];
}
avail[j]=tres[j]-total;
work[j]=avail[j];
printf(" %d \t",work[j]);
}
do
{
for(i=1;i<= pno;i++)
{
for(j=1;j<= rno;j++)
{
need[i][j]=max[i][j]-allocated[i][j];
```

```
}
}
printf("\n Allocated matrix Max need");
for(i=1;i<= pno;i++)
{
printf("\n");
for(j=1;j<= rno;j++)
{
printf("%4d",allocated[i][j]);
}
printf("|");
for(j=1;j<= rno;j++)
{
printf("%4d",max[i][j]);
}
printf("|");
for(j=1;j<= rno;j++)
{
printf("%4d",need[i][j]);
}

}
prc=0;
for(i=1;i<= pno;i++)
{
if(flag[i]==0)
{
prc=i;
for(j=1;j<= rno;j++)
{
if(work[j]< need[i][j])
{
prc=0;
break;
}
}
}
if(prc!=0)
break;
}
if(prc!=0)
{
printf("\n Process %d completed",i);
count++;
printf("\n Available matrix:");
for(j=1;j<= rno;j++)
```

```

{
work[j]+=allocated[prc][j];
allocated[prc][j]=0;
max[prc][j]=0;
flag[prc]=1;
printf(" %d",work[j]);
}
}
}while(count!=pno&&prc!=0);
if(count==pno)
printf("\nThe system is in a safe state!!");
else

printf("\nThe system is in an unsafe state!!");

}

```

**Output:**

```

Enter number of process:5
Enter number of resources:3
Enter total numbers of each resources:10 5 7
Enter Max resources for each process:
for process 1:7 5 3
for process 2:3 2 2
for process 3:9 0 2
for process 4:2 2 2
for process 5:4 3 3
Enter allocated resources for each process:
for process 1:0 1 0
for process 2:3 0 2
for process 3:3 0 2
for process 4:2 1 1
for process 5:0 0 2
available resources:
2 3 0
Allocated matrix Max need
0 1 0| 7 5 3| 7 4 3
3 0 2| 3 2 2| 0 2 0
3 0 2| 9 0 2| 6 0 0
2 1 1| 2 2 2| 0 1 1
0 0 2| 4 3 3| 4 3 1

Process 2 completed
Available matrix: 5 3 2
Allocated matrix Max need

```

```

0 1 0|7 5 3|7 4 3
0 0 0|0 0 0|0 0 0
3 0 2|9 0 2|6 0 0
2 1 1|2 2 2|0 1 1
0 0 2|4 3 3|4 3 1
Process 4 completed
Available matrix: 7 4 3
Allocated matrix Max need
0 1 0|7 5 3|7 4 3
0 0 0|0 0 0|0 0 0
3 0 2|9 0 2|6 0 0
0 0 0|0 0 0|0 0 0
0 0 2|4 3 3|4 3 1
Process 1 completed
Available matrix: 7 5 3
Allocated matrix Max need
0 0 0|0 0 0|0 0 0
0 0 0|0 0 0|0 0 0
3 0 2|9 0 2|6 0 0
0 0 0|0 0 0|0 0 0
0 0 2|4 3 3|4 3 1
Process 3 completed
Available matrix: 10 5 5
Allocated matrix Max need
0 0 0|0 0 0|0 0 0
0 0 0|0 0 0|0 0 0
0 0 0|0 0 0|0 0 0
0 0 0|0 0 0|0 0 0
0 0 2|4 3 3|4 3 1
Process 5 completed
Available matrix: 10 5 7
The system is in a safe state!!

```

### Experiment 10:

**AIM: Write a program to simulate page replacement algorithm for FIFO**

#### Program:

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

```
int p[30],a[30],m,a[30];
```

```
int fifo(int);
```

## UNIX OPERATING SYSTEMS LAB

```
main()
{
int i,n1,n2,pf1,pf2;
printf("***FIFO***");
printf("enter number of pages\n");
scanf("%d",&m);
printf("enter first number of frames\n");
scanf("%d",&n1);
printf("enter second number of frames\n");
scanf("%d",&n2);
printf("enter pages inorder to be loaded\n");
for(i=0;i<m;i++)
scanf("%d",&p[i]);
printf("\nthe pagefaults for %d pageframe is \n",n1);
pf1=fifo(n1);
printf("\nthe pagefaults for %d pageframe is \n",n2);
pf2=fifo(n2);
if(pf1<pf2)
printf("\nbeladys anamoly exists\n");
else
printf("\nbeladys anamoly doesnot exists\n");
}
int fifo(int n)
{
int i,j,flag,pfault=0;
```

## UNIX OPERATING SYSTEMS LAB

```
for(i=0;i<n;i++)
a[i]=-1;
for(i=0;i<m;i++)
{
flag=0;
for(j=0;j<n;j++)
if(a[j]==p[i])
{
flag=1;
break;
}
if(flag==1)
continue;
else
{
for(j=0;j<n-1;j++)
{
a[j]=a[j+1];
}
a[j]=p[i];
pfault++;
}
printf("\na= ");
for(j=0;j<n;j++)
printf("%3d",a[j]);
```

```
}  
printf("\n number of pagefaults are %d",pfault);  
return(pfault);  
}
```

**Output:**

\*\*\*FIFO\*\*\*enter number of pages

5

enter first number of frames

3

enter second number of frames

4

enter pages inorder to be loaded

7

4

1

2

7

the pagefaults for 3 pageframe is

a= -1 -1 7

a= -1 7 4

a= 7 4 1

a= 4 1 2

a= 1 2 7

number of pagefaults are 5

the pagefaults for 4 pageframe is

a= -1 -1 -1 7

a= -1 -1 7 4

a= -1 7 4 1

a= 7 4 1 2

number of pagefaults are 4

beladys anamoly doesnot exists

**b) Write a program to simulate page replacement algorithm for LRU**

**Program**

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

```
int findLRU(int time[], int n){  
    int i, minimum = time[0], pos = 0;  
  
    for(i = 1; i < n; ++i)  
    {  
        if(time[i] < minimum){  
            minimum = time[i];  
            pos = i;  
        }  
    }  
  
    return pos;  
}
```

```
int main()  
{  
    int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j,  
    pos, faults = 0;  
    printf("Enter number of frames: ");  
    scanf("%d", &no_of_frames);
```

```
printf("Enter number of pages: ");
scanf("%d", &no_of_pages);

printf("Enter reference string: ");

for(i = 0; i < no_of_pages; ++i){
    scanf("%d", &pages[i]);
}

for(i = 0; i < no_of_frames; ++i){
    frames[i] = -1;
}

for(i = 0; i < no_of_pages; ++i)
{
    flag1 = flag2 = 0;

    for(j = 0; j < no_of_frames; ++j)
    {
        if(frames[j] == pages[i])
        {
            counter++;
            time[j] = counter;
            flag1 = flag2 = 1;
            break;
        }
    }
}

if(flag1 == 0)
{
    for(j = 0; j < no_of_frames; ++j)
    {
        if(frames[j] == -1)
        {
            counter++;
            faults++;
        }
    }
}
```

```

        frames[j] = pages[i];
        time[j] = counter;
        printf("\ntime = %d\n",time[j]);
        flag2 = 1;
        break;
    }
}

if(flag2 == 0){
    pos = findLRU(time, no_of_frames);
    counter++;
    faults++;
    frames[pos] = pages[i];
    time[pos] = counter;
}

printf("\n");

for(j = 0; j < no_of_frames; ++j)
{
    printf("%d\t", frames[j]);

}
}

printf("\n\nTotal Page Faults = %d", faults);

return 0;
}
/*

```

**Output**

Enter number of frames: 3

Enter number of pages: 6

Enter reference string: 5 7 5 6 7 3

5 -1 -1

5 7 -1

5 7 -1

5 7 6

5 7 6

3 7 6

Total Page Faults = 4

\*/

**c) AIM: Write a program to simulate page replacement algorithm for optimal by least recently**

**Program:**

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

```
int main()
```

```
{
```

```
    int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k,  
    pos, max, faults = 0;
```

```
    printf("Enter number of frames: ");
```

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

```
    printf("Enter number of pages: ");
```

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

```
    printf("Enter page reference string: ");
```

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

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

```
    }
```

## UNIX OPERATING SYSTEMS LAB

```
for(i = 0; i < no_of_frames; ++i)
{
    frames[i] = -1;
}

for(i = 0; i < no_of_pages; ++i)
{
    flag1 = flag2 = 0;

    for(j = 0; j < no_of_frames; ++j)
    {
        if(frames[j] == pages[i])
        {
            flag1 = flag2 = 1;
            break;
        }
    }
}

if(flag1 == 0)
{
    for(j = 0; j < no_of_frames; ++j)
    {
        if(frames[j] == -1)
        {
            faults++;
        }
    }
}
```

```
        frames[j] = pages[i];
        flag2 = 1;
        break;
    }
}
}

if(flag2 == 0)
{
    flag3 = 0;

    for(j = 0; j < no_of_frames; ++j)
    {
        temp[j] = -1;

        for(k = i + 1; k < no_of_pages; ++k)
        {
            if(frames[j] == pages[k])
            {
                temp[j] = k;
                break;
            } }
    }

    for(j = 0; j < no_of_frames; ++j)
    {
        if(temp[j] == -1)
```

```
    {
        pos = j;
        flag3 = 1;
        break;
    }
if(flag3 ==0)
{
    max = temp[0];
    pos = 0;

    for(j = 1; j < no_of_frames; ++j)
    {
        if(temp[j] > max)
        {
            max = temp[j];
            pos = j;
        }
    }
}
frames[pos] = pages[i];
faults++;
}
printf("\n");
for(j = 0; j < no_of_frames; ++j)
{
```

```
        printf("%d\t", frames[j]);
    }
}
printf("\n\nTotal Page Faults = %d", faults);
return 0;
}
```

**Output:**

Enter number of frames: 3

Enter number of pages: 5

Enter page reference string: 4 1 2 3 4

4     -1     -1

4     1     -1

4     1     2

4     3     2

4     3     2

Total Page Faults = 4

**Experiment 11**

**Simulate the following File allocation strategies**

**(a) Sequenced (b) Indexed (c) Linked**

**SEQUENTIAL**

**Program:**

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

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

## UNIX OPERATING SYSTEMS LAB

```
#include<process.h>
struct sequence
{
    char n[20];
    int i;

}s[20];
int create(int);
int del(int);
void display(int);
void main()
{
    int x=0,j=0;
    clrscr();
    while(1)
    {
        printf("1.creation\n2.delete\n3.display\n4.exit");
        printf("\nenter one option");
        scanf("%d",&x);
        switch(x)
        {
            case 1: j=create(j);
                    break;
            case 2: j=del(j);
                    break;
            case 3: display(j);
                    break;
            case 4: exit(1);
                    default : printf("wrong option");
        }
    }
}
int create(int j)
{
    int m,v;
    j++;
    w:printf("\nenter the file name:");
    scanf("%s",&s[j].n);
    m=1;
    while(m<j)
    {
        v=strcmp(s[j].n,s[m].n);
        if(v==0)
        {
            printf("file is already exist\nplease enter another name");
            goto w;
        }
    }
}
```

```
        m++;
    }
    printf("\nenter field:");
    scanf("%d",&s[j].i);
    return(j);
}
int del(int j)
{
    j--;
    return(j);
}
void display(int j)
{
    int l;
    printf("filename\tfield");
    for(l=1;l<=j;l++)
        printf("\n%s\t\t%d\n",s[l].n,s[l].i);
}
}
```

**Output:**

1.creation  
2.delete  
3.display  
4.exit  
enter one option1

enter the file name:1.c

enter field:1  
1.creation  
2.delete  
3.display  
4.exit  
enter one option1

enter the file name:2.c

enter field:2  
1.creation  
2.delete  
3.display  
4.exit  
enter one option3

```
filename    field
1.c        1

2.c        2
1.creation
2.delete
3.display
4.exit
enter one option4
```

## **INDEXED**

### **Program:**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct file
{
    char n[20];
    int fld,ind;
}s[20];
int no,i=-1,a,b,f,j=-1,fe,t;
char tem[20];
void create();
void display();
void del();
void main()
{
    clrscr();
    while(1)
    {
        printf("\n\nmenu");
        printf("\n1.create\n2.display\n3.delete\n4.exit");
        printf("enter ur choice:");
        scanf("%d",&no);
        switch(no)
        {
            case 1: create();
                    break;
            case 2: display();
                    break;
            case 3: del();
                    break;
            case 4: exit(0);
                    default: printf("wrong choice");
        }
    }
}
```

```

void create()
{
    i++;
    printf("\nenter the name of the recoed:");
    scanf("%s",&s[i].n);
    printf("\nenter the index no:");
    scanf("%d",&s[i].ind);
    printf("\nenter the field no:");
    scanf("%d",&s[i].fld);
    j++;
}
void display()
{
    for(a=0;a<i;a++)
    {
        for(b=0;b<i;b++)
        {
            if(s[b].ind>s[b+1].ind)
            {
                t=s[b].ind;
                s[b].ind=s[b+1].ind;
                s[b+1].ind=t;
                strcpy(tem,s[b].n);
                strcpy(s[b].n,s[b+1].n);
                strcpy(s[b+1].n,tem);
                t=s[b].fld;
                s[b].fld=s[b+1].fld;
                s[b+1].fld=t;
            }
            else
                continue;
        }
    }
    printf("\n -----");
    printf("\n\t Index   Recordname   FieldNo");
    for(i=0;i<=j;i++)
    {
        printf("\n\t%d\t",s[i].ind);
        printf("\t%s",s[i].n);
        printf("\t%d",s[i].fld);
    }
    i--;
    printf("\n -----\n");
}
void del()
{
    int de,index=-1,k=0,l;

```



```
-----
```

Index	Recordname	FieldNo
0	a.java	1
1	b.c	2

```
-----
```

menu

1.create

2.display

3.delete

4.exitenter ur choice:4

### LINKED.C

#### Program:

```
#include<stdio.h>
#include<stdlib.h>
typedef struct
{
    int bno,flag,next;
}block;
block b[200],b1;
void main()
{
    int rnum();
    int i,n,s,s1,p[30],r,k[20];
    clrscr();
    printf("\nEnter number of programs:");
    scanf("%d",&n);
    printf("\nEnter the memory block request");
    for(i=1;i<=n;i++)
    {
        printf("\nEnter program requirement");
        scanf("%d",&p[i]);
    }
    for(i=1;i<=n;i++)
    {
        s=rnum();
        b[s].bno=0;
        b[s].flag=1;
        k[i]=0;
        r=p[i]-1;
        while(r!=0)
        {
            s1=rnum();
            b[s].next=s1;
```

```

                b[s1].flag=1;
                b[s1].bno=0;
                s=s1;
                r=r-1;
            }
            b[s1].next=NULL;
        }
        printf("\n Starting blocks for program");
        for(i=1;i<=n;i++)
            printf("\n%5d%5d",i,k[i]);
        printf("\n allocated blocks");
        for(i=1;i<=200;i++)
        {
            if(b[i].flag==1)
                printf("\n%5d%5d",b[i].bno,b[i].next);
        }
    }
int rnum()
{
    int k,i;
    for(i=1;i<=200;i++)
    {
        k=rand()%200;
        if(b[i].flag!=1)
            break;
    }

    return k;
}

```

**Output:**

Enter number of programs:2

Enter the memory block request

Enter program requirement3

Enter program requirement4

Starting blocks for program

1 0

2 0

allocated blocks

0 117

0 56

0 195

0 182

0 130

```
0 0
0 0
```

## Experiment 12

**Write a C program that illustrates two processes communicating using shared memory.**

### Program:

//Program 1: This program creates a shared memory segment, attaches itself to it and then writes some content into the shared memory segment.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
int i;
void *shared_memory;
char buff[100];
int shmid;
shmid=shmget((key_t)2345, 1024, 0666|IPC_CREAT); //creates shared memory segment with
key 2345, having size 1024 bytes. IPC_CREAT is used to create the shared segment if it does
not exist. 0666 are the permissions on the shared segment
printf("Key of shared memory is %d\n",shmid);
shared_memory=shmat(shmid,NULL,0); //process attached to shared memory segment
printf("Process attached at %p\n",shared_memory); //this prints the address where the segment is
attached with this process
printf("Enter some data to write to shared memory\n");
read(0,buff,100); //get some input from user
strcpy(shared_memory,buff); //data written to shared memory
printf("You wrote : %s\n",(char *)shared_memory);
}
.
```

### Output

```
Key of shared memory is 0
Process attached at 0x7ff3f2048000
Enter some data to write to shared memory
hi! this is shared memory program
```

You wrote : hi! this is shared memory program

**//Program 2: This program attaches itself to the shared memory segment created in Program 1. Finally, it reads the content of the shared memory**

**Program:**

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
int i;
void *shared_memory;
char buff[100];
int shmid;
shmid=shmget((key_t)2345, 1024, 0666);
printf("Key of shared memory is %d\n",shmid);
shared_memory=shmat(shmid,NULL,0); //process attached to shared memory segment
printf("Process attached at %p\n",shared_memory);
printf("Data read from shared memory is : %s\n",(char *)shared_memory);
}
```

How it works?

shmget() here generates the identifier of the same segment as created in Program 1. Remember to give the same key value. The only change is, do not write IPC\_CREAT as the shared memory segment is already created. Next, shmat() attaches the shared segment to the current process.

After that, the data is printed from the shared segment. In the output, you will see that it is the same data that you have written while executing the Program 1.

## **Output**

hi! this is shared memory program

## **Experiment 13.**

**Write a C program to simulate producer and consumer problem using semaphores.**

**Program:**

## UNIX OPERATING SYSTEMS LAB

```
#include<stdio.h>
int mutex=1,full=0,empty=2,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;
}
}
}
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:**

\$cc linux6.c

\$/a.out

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

BUFFER IS FULL

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

### **Experiment 14:**

**Write C program to create a thread using pthreads library and let it run its function.**

#### **Program:**

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
void *mythread(void *vargp)
{
sleep(1);
printf("WELCOME TO PTHREADS LIBRARY\n");
return NULL;
}
int main()
{
pthread_t id;
printf("Before thread\n");
pthread_create(&id,NULL,mythread,NULL);
pthread_join(id,NULL);
printf("After thread\n");
exit(0);
}
```

#### **OUTPUT:**

```
$cc linux7.c
```

```
$/a.out
```

```
WELCOME TO PTHREADS LIBRARY
```

### **Experiment 15:**

**Write a C program to illustrate concurrent execution of threads using pthreads library.**

**Program:**

```
#include<stdio.h>
#include<stdlib.h>
#include<pthread.h>
void *mythread1(void *vargp)
{
int i;
printf("THREAD1\n");
for(i=1;i<=10;i++)
printf("i=%d\n",i);
printf("Exit from Thread1\n");
return NULL;
}
void *mythread2(void *vargp)
{
int j;
printf("THREAD2 \n");
for(j=1;j<=10;j++)
printf("j=%d\n",j);
printf("Exit from Thread2\n");
return NULL;
}
int main()
{
pthread_t id;
printf("Before thread\n");
pthread_create(&id,NULL,mythread1,NULL);
pthread_create(&id,NULL,mythread2,NULL);
pthread_join(id,NULL);
pthread_join(id,NULL);
printf("After thread\n");
}
```

**OUTPUT:**

```
$cc linux8.c
```

```
$/a.out
```

```
thread 1
```

```
i=1
```

## UNIX OPERATING SYSTEMS LAB

i=2

i=3

thread 2

j=1

j=2

j=3

j=4

j=5

j=6

j=7

j=8

i=4

i=5

i=6

i=7

i=8

i=9

i=10

Exit from thread1

J=9

J=10

Exit from thread2