GODAVARI INSTITUTE OF ENGINEERING & TECHNOLOGY
(Autonomous)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

II B.Tech I Sem GRBT-20

LINUX & SHELL PROGRAMMING

LAB RECORD

GODAVARI INSTITUTE OF ENGINEERING & TECHNOLOGY
(Autonomous)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

II B.Tech I Sem GRBT-20

LINUX & SHELL PROGRAMMING LAB

List of Exercises:

1. Study of general purpose utility commands of Linux.
2. Installation, Configuration & Customizations of Linux.
3. Study of vi editor execute shell commands through *vi* editor
4. Write a shell script that takes a command –line argument and reports on whether it is directory, a file, or something else.
5. Write a shell script that accepts one or more file name as arguments and converts all of them to uppercase, provided they exist in the current directory.
6. Write a shell script that determines the period for which a specified user is working on the system.
7. Write a shell script that deletes all lines containing a specified word in one or more files supplied as arguments to it.
8. Write a shell script that accepts two integers as its arguments and computes the value of first number raised to the power of the second number.
9. Given two files each of which contains names of students. Create a program to display only those names that are found on both the files.
10. Create a program to find out the inode number of any desired file.
11. Write a pipeline of commands, which displays on the monitor as well as saves the information about the number of users using the system at present on a file called user.txt.
12. Write a shell script that computes the gross salary of a employee according to the following rules: The basic salary is entered interactively through the key board.
    i. If basic salary is < 25000 then HRA =10% of the basic and DA =90% of the basic.
    ii. If basic salary is >=25000 then HRA =Rs500 and DA=98% of the basic.
13. Write a shell script that accepts any number of arguments and prints them in the reverse order.
14. Write a shell script to find the smallest of three numbers that are read from the keyboard.
15. Write a shell script that displays a list of all the files in the current directory to which the user has read, write and execute permissions.
16. Write a shell script that reports the logging in of a specified user within one minute after he/she logs in. The script automatically terminates if the specified user does not login during a specified period of time.

Additional Experiments:

1. Write a C program that takes one or more file or directory names as command line input and reports the following information on the file:
   i. File type ii. Number of links
   iii. Read, write and execute permissions iv. Time of last access
2. Write a C programs that simulate the following Linux commands:
   i) mv ii) cp iii) ls
3. Write a C program to illustrate concurrent execution of threads using threads library.
4. Write a C program that illustrates two processes communicating using shared memory.

GODAVARI INSTITUTE OF ENGINEERING AND TECHNOLOGY

Department of **COMPUTER SCIENCE AND ENGINEERING**

Name:_____ Pin No: | | | | | | | | | | |

CERTIFICATE

Certified that this is the bonafide record of practical work done by

Mr./Ms._____

a Student of_____with Pin No:_____

in the_____Laboratory during the Academic year_____

No. of Experiments Conducted: | | | No. of Experiments attended: | | |

Faculty In-charge Head of the Department

Submitted for Practical Examination Conducted on_____

Examiner – 1 Examiner – 2

INDEX					
S.No	Date	Name of the Experiment	Page No.	Signature	Remarks

What is UNIX?

- A computer operating system.
- It is designed to be used by many people at the same time (multi-user).
- Runs on a variety of processors
- It provides a number of facilities:
  - management of hardware resources
  - directory and file system
  - loading / execution / suspension of programs

Why Use UNIX?

- multi-tasking / multi-user
- networking capability
- graphical (with command line)
- easy to program
- portable (PCs, mainframes, super-computers)

File: a container for storing information.

A file is of 3 types.

Ordinary file: It contains data as a stream of characters. It is of 2 types.

- Text file: contains printable characters.
- Binary file: contains both printable & non printable characters.

Directory file: contains no data but it maintains some details of the files & subdirectories that it contains.

Device file: It represents the device or peripheral.

Study of general purpose utility commands of Linux.

Text processing utilities

cat: cat is used to create the files.

```
$cat> filename
Type some text here
Press ctrl+d
$
```

cat can also be used to display the contents of a file.

```
$cat filename
```

cat can also concatenate the contents of 2 files and store them in third file.

```
$cat>file1 file2>new file
```

To append the contents of two files into another file use

```
$cat>file1 file2>>new file
```

tail:

tail command displays the end of the file. It displays the last ten lines by default.

$tail file

To display last 3 lines use

$tail –n 3 file or

$tail -3 file

We can also address the lines from the beginning of the file instead of the end.

The + count allows to do that.

head:

- head command as the name implies, displays the top of the file. When used without an option, it displays the first 10 lines of the file.

  $head file

- We can use –n option to specify a line count and display, say first 3 lines of the file.

  $head –n 3 file or

  $head -3 file

sort: Sort can be used for sorting the contents of a file.

$sort shortlist

- Sorting starts with the first character of each line and proceeds to the next character only when the characters in two lines are identical.

Sort options:

- With –t option sorts a file based on the fields.

  $sort –t "|" +2 shortlist

- The sort order can be reversed with –r option.
- If the primary key is the third field and the secondary key the second field, we can use

  $sort –t \| +2 -3 +1 shortlist

- Numeric sort (-n):
- To sort on number field use sort with –n option.

  $sort –t +2 -3 –n group1

- Removing duplicate lines (-u):
- The –u option u purge duplicate lines from a file.

nl: nl is used for numbering lines of a file.

- nl numbers only logical lines –those containing something other apart from the new line character.

  $nl file

- nl uses a tab as a default delimiter, but we can change it with –s option.

  $nl –s file

- nl won't number a line if it contains nothing.

grep: globally search for a regular expression and print.

- grep scans a file for the occurrence of a pattern and depending on the options used, displays
- Lines containing the selected pattern.
- Lines not containing the selected pattern (-v).
- Line numbers where pattern occurs (-n)
- No. of lines containing the pattern (-c)
- File names where pattern occurs (-l)

Syntax: **grep option pattern filename(s)**

egrep: extended grep

egrep extended set includes 2 special characters + and ?.

+ --matches one or more occurrences of the pervious character.

?-- matches zero or more occurrences of the pervious character.

fgrep: fast grep

- If search criteria requires only sequence expressions, fgrep is the best utility.
- Fgrep supports only string patterns, no regular expressions.
- To extract all the lines that contain an apostrophe use fgrep as follows:

  $fgrep "'" file

Cut: slitting the file vertically

U can slice a file vertically with cut command.

- Cutting columns(-c):

Cut with –c option cuts the columns.

To extract first 4 columns of the group file :

$cut –c 1-4 group1

The specification –c 1-4 cuts columns 1 to 4.

Cutting fields: To cut 1st and 3rd fields use

$cut –d: -f1,3 group1

Paste: pasting files

- What u cut with the cut can be pasted back with paste command-but vertically rather than horizontally. u can view two files side by side by pasting them.
- To join two files calc.lst and result.lst use

  $paste –d= calc.lst result.lst

Join:

- is a command in Unix-like operating systems that merges the lines of two sorted text files based on the presence of a common field.
- The join command takes as input two text files and a number of options. If no command-line argument is given, this command looks for a pair of lines from the two files having the same first field (a sequence of characters that are different from space), and outputs a line composed of the first field followed by the rest of the two lines.

  $join file1 file2

tee:

Unix tee command breaks up its input into two components; one component is saved in a file, and other is connected to the standard output.

tee doesn't perform any filtering action on its input; it gives exactly what it takes.

tee can be placed anywhere in a pipeline.

we can use tee to save the output of the who command in a file and display it as well:

$who |tee user.lst

- The tee command reads standard input, writes its content to standard output and simultaneously copies it into the specified file or files.

Comm:

- Suppose if u have 2 list of people, u are asked to find out the names available in one and not the other or even those common to both. Comm is the command that u need to for this work.
- It requires two sorted file and lists the differing entries in different columns.

  $comm file1 file2
- Comm display a three-column output**.**

Cmp: comparing two files

- The two files are compared byte by byte and the location of the first mismatch is echoed to the screen using cmp.
- Cmp when invoked without options it does not bother about possible subsequent mismatches.

  $cmp group1 group2

  If two files are identical cmp display s no message, but simply returns the $ prompt.

diff: converting one file to another

- diff takes different approach to displaying the differences.
- It tells u which lines in one file have to be changed to make two files identical.

- When used with the same files it produces a detailed output.

  $diff group[12]

- diff uses certain special symbols with its instructions to indicate the changes needed to make two files identical.

File handling utilities

cp (Copying Files)

– To create an exact copy of a file you can use the "cp" command. The format of this command is:

Syntax: cp [-option] source destination

Eg:

cp file1 file2

Here file1 is copied to file2.

Eg:

cp file1 file2 dir

File1 file2 are copied to dir.

- Copying Files

cp turns to interactive when –i option is used & destination file also exists.

$cp -i file1 file2

overwrite file2 (yes/no)?

Y at this prompt overwrites the file.

mv (Moving and Renaming Files)

Used to move or rename the files/directories.

Syntax: mv source destination

$mv test sample

Here test is renamed as sample.

ln (link):

- Used to create links (both soft & hard links).
- It creates the alias & increase the link count by one.

Syntax: $ln file1 file2

- ln won't work if the destination file also exists.

rm (Deleting Files and Directories)

To delete or remove a file, you use the "rm" command. For example,

Eg: $rm my. listing

will delete "my.listing".

With –i option removes the files interactively.

$rm –i file1

With –r option recursively removes directories.

$rm –r dir1

mkdir(Make Directory): used to create one or more directories.

Eg: $mkdir book

Creates the directory named book.

Eg: $mkdir dbs doc dmc

Creates three directories.

rmdir (**remove directories**):

Removes empty directories.

Eg: $rmdir book

removes directory named book if it is empty.

Eg: $rmdir dbs doc dmc

Removes 3 directories.

find: It recursively examines a directory tree to look for matching some criteria and then takes some action on the selected files.

Syntax:

$find path_list selection_criteria action

To locate all files named a. out use

Eg: $find / -name a. out –print

- '/' indicates search should start from root directory.
- To locate all c files in current directory

  $find . -name "*.c" –print
- To find all files begin with an uppercase letter use

  $find . –name '[A-Z]*' –print

chmod: changing file permission

chmod sets a file's permissions (read, write and execute) for all three categories of users (owner, group and others)

Syntax: chmod category operation permission file(s)

Category	**operation**	**permissions**
u-user	+ assign permission	r-read permission
g-group	- remove permission	w-write permission
o-others	= assigns absolute permission	x-execute permission
a-all		

Process utilities

ps (process status): Display some process attributes.

- $ps

```
PID   TTY  TIME    CMD
1078  pts/2 0:00    bash
```

- Ps presents a snapshot of the process table.
- Ps with –f option displays a fuller listing that includes the PPID.
- Ps with –u option followed by user-id displays the processes owned by the user-id.
- Ps with –e option displays the system processes.

Who: know the users. Displays the users currently logged in the system.

$who

Whoami: Show you the owner of this account

$whoami

W: Tell you who is logging in and doing what!

$w

Finger: Displays the information about the users.

$finger *user* Find out the personal information of a user

$finger *name* Try to find the person's info. by name

- finger ***email-address***

Try to find the person's info across the network

Disk utilities

- **du:** disk usage
- **du** estimate the file space usage on the disk.
- It produces a list containing the usage of each subdirectory of its argument and finally produces a summary.

$du /home/usr1

Mount:

- Used to mount the file systems.
- Takes 2 arguments-**device name ,mount point**.
- Mount uses an option to specify the type of file system.
- To mount a file system on the /oracle directory on Linux system use

```
$mount –t ext2 /dev/hda3 /oracle
$mount –t iso9660 /dev/cdrom /mnt /cdrom
$mount –t vfat /dev/hda1 /msdos
$mount –t msdos /dev/fd0 /floppy
```

Umount: unmounting file systems

- Unmounting is achieved with the umount command. which requires either file system name or the mount point as argument.
- $umount /oracle
- $umount /dev/hda3
- Unmounting a file system is not possible if the file is opened.

Networking commands

ftp: file transfer protocol

ftp is used to transfer files. It can be used with host name.

$ftp Saturn

Connected to Saturn
220 Saturn ftp server
Name (Saturn: summit): Henry
Password: ******

To quit ftp use close and then bye or quit.

ftp>close

221 good bye

ftp>bye

Transferring files: Files can be of 2 types.

- Uploading(put & mput):
- To upload ur web pages & graphic files to website.

The put command sends a single file to the remote machine.

ftp>binary 200 type set to I

ftp>put penguin.gif

To copy multiple files use mput.

ftp>mput t*.sql

Downloading files: get & mget

- To download the files from remote machine use get & mget.

ftp>get ls-lR.gz

ftp>_

telnet: Remote login

If u have an account on the host in a local network (or on internet),u can use this with the host name or the ip address as argument.

$telnet Saturn

Trying to 192.168.0.1…

Connected to Saturn

- Login:----
- Password:-----
- To quit telnet by using exit command.
- telnet prompt:

rlogin: remote login without password

- rlogin is the Berkley's implementation of the remote login facility.
- To log on to your own identical remote account without using either the user name or password.
- **$rlogin Jupiter**
- Last login :….
- rlogin is terminated with ctrl+d or exit or logout.

Backup utilities

tar: the tape archive program

- Tar doesn't normally write to the standard output but creates an archive in the media.
- Tar accepts file and directory names as arguments.
- It operates recursively.
- It can create several versions of same file in a single archive.
- It can append to an archive without overwriting the entire archive.
- -c option is used to copy files to backup device.

  $tar –cvf /dev/rdsk/foq18dt /home/sales/sql/*.sql
- The verbose option (-v) shows the no. of blocks used by each file.
- Files are restored with the –x (extract) key option. when no file or directory name is specified it restores all files from the backup device.

Awk: Aho, Weinberger and Kernighan

- Awk is not just a command, but a programming language too.

Syntax: awk options 'selection criteria {action}' file(s)

- Simple filtering

  $ awk '/Simpsons/ { print }' homer |Simpsons
- Splitting a line into fields

  $ **awk –F "|" '/Simpsons/ {print $1}' homer**

tr: translating characters

- **tr** command manipulates individual characters in a character stream.

  tr options expr1 expr2< standard input

It takes input only from the standard input, it does not take input a file name as its argument.

Examples:

$ tr "[a-z]" "z[a-y]" < computer.txt

$tr –d '|/' <shortlist | head -3

pg:

- pg is used to display the output of a file in page by page.

  $pg file1

When a single command is not sufficient to solve a problem, try to join the commands together.

Two approaches for this are:

- Redirection
- Piping

Redirection:

- Unix commands are built-up in such a way that they can take the input from the keyboard, often called standard input and usually send their output to the screen, often called standard output. Commands also send error information to the screen.

We can also change these defaults by using a process called **redirection**.

Study of vi editor execute shell commands through *vi* editor

vi Editor

- vi is a full screen text editor. It was created by Bill Joy.
- Bram Moolenaor improved it and called it vim (vi improved).
- **Invoking vi:**

  $Vi file_name

Modes of operation

- Vi has 3 mode of operation.

  1. **Command mode:** In this mode all the keys pressed by the user are interpreted as commands. It may perform some actions like move cursor, save, delete text, quit vi, etc.
  2. **Input/Insert mode:** used for inserting text.
     - start by typing i ;

Inserting or Adding Text:

i	insert text before cursor
I	insert text at beginning of current line
a	append text after cursor
A	append text to end of current line
o	open and put text in a new line below current line
O	open and put text in a new line above current line

Changing Text

r	replace single character under cursor (no <Esc> needed)
R	replace characters, starting with current cursor position, until <Esc> hit
cw	change the current word with new text, starting with the character under cursor, until <Esc> hit
cNw	change N words beginning with character under cursor, until <Esc> hit; e.g., c5w changes 5 words
C	change (replace) the characters in the current line, until <Esc> hit
cc	change (replace) the entire current line, stopping when <Esc> is hit
Ncc or **cNc**	change (replace) the next N lines, starting with the current line stopping when <Esc> is hit

Deleting Text

X	delete single character under cursor
Nx	delete N characters, starting with character under cursor
Dw	delete the single word beginning with character under cursor
dNw	delete N words beginning with character under cursor; e.g., d5w deletes 5 words
D	delete the remainder of the line, starting with current cursor position
dd	delete entire current line
Ndd or dNd	delete N lines, beginning with the current line; e.g., 5dd deletes 5 lines

– finish with ESC

To Exit vi

Usually the new or modified file is saved when you leave vi. However, it is also possible to quit vi without saving the file

Note: The cursor moves to bottom of screen whenever a colon (:) is typed. This type of command is completed by hitting the <Return> (or <Enter>) key.

:x quit vi, writing out modified file to file named in original invocation

:w quit vi, writing out modified file to file named in original invocation

:q quit (or exit) vi

:q! quit vi even though latest changes have not been saved for this vi call

3. Ex mode or last line mode:

– Used for giving commands at command line.

– The bottom line of vi is called the command line.

Write a shell script that takes a command -line argument and reports on whether it is directory, a file, or something else.

Script:

```
echo " Enter File Name"
read str
if [ -f $str ]
then
echo "File exists and it is an ordinary file"
elif [ -d $str ]
then
echo "directory file"
else
echo "not exists"
fi
if [ -c $str ]
then
echo "character device files"
fi
```

Output:

```
Enter File Name
Samplefile
File exists and it is an ordinary file
```

Write a shell script that accepts one or more file name as arguments and converts all of them to uppercase, provided they exist in the current directory.

Script:

```
echo "Enter File Name: "
read fileName
if [ ! -f $fileName ]
then
echo "Filename $fileName does not exists"
exit
fi
echo “The content in the File $fileName is:”
`tr '[A-Z]' '[a-z]' < $fileName`
```

Output:

```
Enter File Name:
Sample
The content in the File Sample is:
HI
THIS IS LINUX & SHELL PROGRAMMING LAB
```

Write a shell script that determines the period for which a specified user is working on the system.

Script:

```
echo "Enter the login of the user "
read name
log=`who -l | grep -w $name`
`who -l | grep -w $name | cat > f11`
echo "Login Details=$log"
if [ $? -ne 0 ]
then
echo "$name has not logged in yet"
exit
fi
t=`cut -c 30,31 f11`
m=`cut -c 33,34 f11`
`date | cat> sdate`
st=`cut -c 12,13 sdate`
mt=`cut -c 15,16 sdate`
echo "Login Time in Hours="$t
echo "Login Time Minutes="$m
echo "system time="$st
echo "System Time Minutes="$mt
th=`expr $st - $t`
tm=`expr $mt - $m`
echo "$name is working since $th Hour - $tm Minutes"
```

Output:

```
Enter the login of the user
user17
Login Details= user17   pts/1   Jan  1 08:02 (192.168.0.117)
Login Time in Hours=08
Login Time Minutes=02
system time=10
System Time Minutes=22
user17 is working since 2 Hour - 20 Minutes
```

Write a shell script that deletes all lines containing a specified word in one or more files supplied as arguments to it.

Script:

```
if [ $# -eq 0 ]
then
echo "Please enter one or more filenames as argument"
exit
fi
echo "Enter the word to be searched in files"
read word
echo “After deleting the specified word:”
for file in $*
do
sed "/$word/d" $file | tee tmp
mv  tmp  $file
done
```

Output:

```
sh delete.sh  f1
Enter the word to be searched in files
Hi
After deleting the specified word:
This is LINUX & SHELL PROGRAMMING Lab
```

Write a shell script that accepts two integers as its arguments and computes the value of first number raised to the power of the second number.

Script:

```
if [ $# -ne 2] then
echo “check the number of arguments”
fi
count=1
result=1
if [ $2 –ge 0 ] then
while [ $count –le $2 ] do
result=`expr $result \* $1`
count=`expr $count + 1`
done
echo “The Result is : $result”
fi
```

Output:

```
sh power.sh 5 2
The Result is: 25
```

Given two files each of which contains names of students. Create a program to display only those names that are found on both the files.

Script:

```
if [ $# -ne 2 ]
then
echo "Error : Invalid number of arguments."
exit
fi
str=`cat $1 | tr '\n' ' '`
for a in $str
do
echo "Word = $a, Count = `grep -c "$a" $2`"
done
```

Output:

```
$ cat test
hello giet csm
$ cat test1
hello giet csm
hello giet csc
hello
$ sh 1.sh test test1
Word = hello, Count = 3
```

Create a program to find out the inode number of any desired file.

Script:

```
echo “Enter the file name for which do you want inode number:”
read  filename
echo “The inode number of given file $filename is:”
ls -i
```

Output:

```
Enter the file name for which do you want inode number:
file7.txt
The inode number of given file file7.txt is:
844424930177767   file7.txt
```

Write a pipeline of commands, which displays the information about the number of users using the system at present on the monitor as well as saves on a file called user.txt

Script:

```
who –H | cat > user.txt
echo “The number of users logged in:”
cat < user.txt
```

Output:

The Number of users currently logged in:

NAME	LINE	TIME	COMMENT
charles	pts/0	2022-02-06 12:43	(:0.0)
babbage	pts/1	2022-02-06 12:43	(:0.0)

Write a shell script that computes the gross salary of a employee according to the following rules: The basic salary is entered interactively through the key board.

i. **If basic salary is < 25000 then HRA =10% of the basic and DA =90% of the basic.**
ii. **If basic salary is >=25000 then HRA =Rs500 and DA=98% of the basic.**

Script:

```
echo enter the basic
read basic
if [ $basic -lt 15000 ]
then
hra=`echo "scale=2; $basic * 0.1" | bc`
da=`echo "scale=2; $basic * 0.9" | bc`
else
hra=500
da=`echo "scale=2; $basic * 0.98" | bc`
fi
gs=`echo "scale=2;$basic +$hra +$da" | bc`
echo " gross =" $gs
echo "hra =" $hra
echo "da =" $da
```

Output:

```
$ sh 8a.sh
enter the basic pay
1000
gross = 2000.0
hra = 100.0
da = 900.0
$ sh 8a.sh
enter the basic
20000
gross = 40100.00
hra = 500
da = 19600.00
```

Write a shell script that accepts any number of arguments and prints them in the reverse order.

Script:

```
a=$#
echo "Number of arguments are" $a
x=$*
c=$a
res="
while [ 1 -le $c ]
do
c=`expr $c - 1`
shift $c
res=$res' '$1
set $x
done
echo Arguments in reverse order $res
```

Output:

```
sh 1prg.sh a b c
No of arguments are 3
Arguments in reverse order c b a
```

Write a shell script to find the smallest of three numbers that are read from the keyboard.

Script:

```
echo enter first number
read a
echo enter second number
read b
echo enter third number
read c
if [ $a -eq $b -a $b -eq $c ]
then
echo all numbers are equal
exit
fi
if [ $a -lt $b ]
then
s1=$a
s2=$b
else
s1=$b
s2=$a
fi
if [ $s1 -gt $c ]
then
s2=$s1
s1=$c
fi
echo "smallest number is " $s1
```

Output:

```
enter first number:54
enter second number:67
enter third number :22
smallest number is 22
```

Write a shell script that displays a list of all the files in the current directory to which the user has read, write and execute permissions.

Script:

```
echo "The name of all files having all permissions :"
 for file in *
do
# check if it is a file
if [ -f $file ]
then
# check if it has all permissions
if [ -r $file -a -w $file -a -x $file ]
then
# print the complete file name with -l option
ls -l $file
# closing second if statement
fi
# closing first if statement
fi
done
```

Output:

```
The name of all files having all permissions:
-rwxrwxrwx. 1 CSE CSE 16 Jan 31 20:03  program1.txt
-rwxrwxrwx. 1 CSE CSE 16 Jan 31 20:03  program2.txt
```

Write a shell script that reports the logging in of a specified user within one minute after he/she logs in. The script automatically terminates if the specified user does not login during a specified period of time.

Script:

```
echo _Enter the login name of the user:‖
read user
period=0
while [ true]
do
var=`who | grep –w ―$user‖`
len=`echo ―$var | wc –c`
if [ $len –gt 1 ]
then
echo ―$user logged in $tm seconds‖
exit
else
sleep 1
tm=`expr $tm + 1`
fi
if [ $tm –eq 61 ]
then
echo ―$user did not login within 1 minute‖
exit
fi
done
```

Output:

```
Enter the login name of the user :cse517
cse517 logged in 25 seconds
Enter the login name of the user :cse554
Cse554 did not login within 1 minute
```

Write a C program that takes one or more file or directory names as command line input and reports the following information on the file:

i) File type ii) Number of links

iii) Read, write and execute permissions iv) Time of last access

Program:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<sys/syscall.h>
#include<fcntl.h>
void main()
{
int fd;
struct stat buf;
fd=open("f1",O_RDONLY|O_CREAT,600);
if(fd!=-1)
{
 if(fstat(fd,&buf)==0)
 {
 printf("mode of file is %u",buf.st_mode);
 printf("\n size of the file is %u",buf.st_size);
 printf("\n device name %u",buf.st_dev);
 printf("\n inode of file is %u",buf.st_ino);
 printf("\n no. of links are %u",buf.st_nlink);
 printf("\n owner of a file is %u",buf.st_uid);
 printf("\n no.of blocks is %u",buf.st_blocks);
 printf("\n group owner is %u",buf.st_gid);
 printf("\n blocks size of the file is %u",buf.st_blksize);
 printf("\n time of last modified is %u",buf.st_ctime);
 }
 else
 printf("error in fstat() syscall");
}
else
 printf("error in open() sys call");
}
```

Output:

mode of file is 33188

size of the file is 23

device name 2049

inode of file is 1194371

no. of links are 2

owner of a file is 1001

no.of blocks is 8

group owner is 1002

blocks size of the file is 4096

time of last modified is 1489657990

Write C programs that simulate the unix mv command

Program:

```
#include<sys/types.h>
#include<sys/stat.h>
#include<stdio.h>
#include<fcntl.h>
main( int argc,char *argv[] )
{
int i,fd1,fd2;
char *file1,*file2,buf[2];
file1=argv[1];
file2=argv[2];
printf("file1=%s   \n  file2=%s",file1,file2);
fd1=open(file1,O_RDONLY,0777);
fd2=creat(file2,0777);
while(i=read(fd1,buf,1)>0)
write(fd2,buf,1);
if(remove(file1)==0)
printf("\n File Copied and File f6 was deleted.");
else
printf("\n File coping unsuccessful");
close(fd1);
close(fd2);
}
```

Output:

```
vi cp2.c
cc cp2.c
./a.out f6 ff5
file1=f6
file2=ff5
File Copied and File f6 was deleted.
```

Write C programs that simulate the unix cat command

Program:

```
#include<sys/types.h>
#include<sys/stat.h>
#include<stdio.h>
#include<fcntl.h>
main( int argc,char *argv[] )
{
int fd,i;
char buf[2];
fd=open(argv[1],O_RDONLY,0777);
if(fd==-argc)
{
printf("file open error");
}
else
{
while((i=read(fd,buf,1))>0)
{
printf("%c",buf[0]);
}
close(fd);
}
}
```

Output:

```
cc cp3.c
./a.out ff5
cse
ece
mech
civil
```

Write C programs that simulate the unix cp command

Program:

```
#include<sys/types.h>
#include<sys/stat.h>
#include<stdio.h>
#include<fcntl.h>
main( int argc,char *argv[] )
{
int i,fd1,fd2,flag=0;
char *file1,*file2,buf[2];
file1=argv[1];
file2=argv[2];
printf(" Source File=%s \n  Destination File=%s",file1,file2);
fd1=open(file1,O_RDONLY,0777);
if(fd1==-1)
{
printf("\n Source File Doesn't exist ");
}
else
{
fd2=creat(file2,0777);
while(i=read(fd1,buf,1)>0)
{
if(write(fd2,buf,1)!=-1)
{
flag=1;
}
}
if(flag==1)
printf("\n Content copied \n");
else
printf("\n Not copied \n");
close(fd1);
close(fd2);     }    }
```

Output:

```
cc cp4.c
./a.out dpk1 dpk2
Source File=dpk1
Destination File=dpk2
Content copied
```

Write C programs that simulate the unix ls command

Program:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<dirent.h>
int main()
{
DIR *dp;
struct dirent *dirp;
dp=opendir(".");
printf("\n The List of files in the current working Directory: \n");
while((dirp=readdir(dp))!=NULL)
{
if(dirp->d_ino==0)
continue;
else
printf("%s \n",dirp->d_name);
}
}
```

Output:

```
cc cp5.c
./a.out
The List of files in the current working Directory:
fp.c
f3
cp5.c
f1
```