EXPERIMENT NO: 1 (a)

AIM : To Study Unix/Linux general purpose utility command list man,who,cat,cd,cp,ps,ls,mv,rm,mkdir,rmdir,echo,more,date,time,kil, History,chmod,chown,finger,pwd,cal,logout,shutdown.

DESCRIPTION :

man :

man command in Linux is used to display the user manual of any command that we can run on the terminal. It provides a detailed view of the command which includes NAME, SYNOPSIS, DESCRIPTION, OPTIONS, EXIT STATUS, RETURN VALUES, ERRORS, FILES, VERSIONS, EXAMPLES, AUTHORS and SEE ALSO.

who :

In Unix, "who" command allows to show or print the number of users who has been logged into your Unix computer system currently. The main usage of who command in Unix without command-line parameter is to show the name of the users who are logged in currently. Also, based on the Unix system, we can also get the information to print the terminal and time at which they have logged in. The who command is used to print the information about all the users who are currently logged in.

cat :

The **cat** command reads one or more files and prints their contents to standard output. Files are read and output in the order they appear in the command arguments.

cd :

The change directory (**cd**) command is built into the system shell and changes the current working directory. The cd command can be used to either change to a directory that is relative to the the location of the current working directory or to an absolute location in the filesystem.

cp :

The CP command is most important command in unix which is used to copy the file from one location to another location. In windows systems we have direct GUI to copy the files and folders. But in unix there is no such GUI provided. Unix has the command named 'CP' which will copy the files from source and paste it to destination or target.

ps :

Report a snapshot of the current processes. **ps** displays information about a selection of the active processes. If you want a repetitive update of the selection and the displayed information, use *top*(1) instead.

ls :

ls List information about the FILEs (the current directory by default). Sort entries alphabetically if none of -cftuvSUX nor --sort is specified. Mandatory arguments to long options are mandatory for short options too.

mv :

mv stands for **move**. mv is used to move one or more files or directories from one place to another in file system like UNIX. It has two distinct functions:
(i) It rename a file or folder.
(ii) It moves group of files to different directory.
No additional space is consumed on a disk during renaming. This command normally **works silently** means no prompt for confirmation.

rm :

rm removes each specified file. By default, it does not remove directories.

rm stands for remove.

mkdir :

The **mkdir** command creates a single directories or multiple directories.

rmdir :

The rmdir command stands for remove directory.

This command removes a single directory or multiple directories.

echo :

The echo command helps to display the line of text.

more :

The **more** command displays the file called *name* in the screen. The RETURN key displays the next line of the file. The spacebar displays the next screen of the file.

date :

The **date** command is used to print out, or change the value of, the system's time and date information.

time :

The **time** command runs the specified program *command* with the given arguments.
When *command* finishes, **time** writes a message to standard error giving timing statistics about this program run.

kill :

The Kill command in unix or linux operating system is used to send a signal to the specified process or group. If we dont specify any signal, then the kill command passes the SIGTERM signal. We mostly use the kill command for terminating or killing a process. However we can also use the kill command for running a stopped process

history :

The history Command. In its simplest form, you can use the history command by just typing its name: history. The list of previously used commands is then written to the terminal window. The commands are numbered, with the most recently used (those with the highest numbers) at the end of the list.

chmod :

chmod changes the permissions of each given file according to mode, where mode describes the permissions to modify. Mode can be specified with octal numbers or with letters.

chown :

chown - To change owner, change the user and/or group ownership of each given File to a new Owner. Chown can also change the ownership of a file to match the user/group of an existing reference file.

finger :

In Unix , finger is a program you can use to find information about computer users. It usually lists the login name, the full name, and possibly other details about the user you are fingering. These details may include the office location and phone number (if known), login time, idle time, time mail was last read, and the user's plan and project files. The information listed varies, and you may not be able to get any information from some sites.

pwd :

print name of current/working directory.

cal : To display a calendar.

logout :

logout command allows you to programmatically logout from your session. causes the session manager to take the requested action immediately.

shutdown :

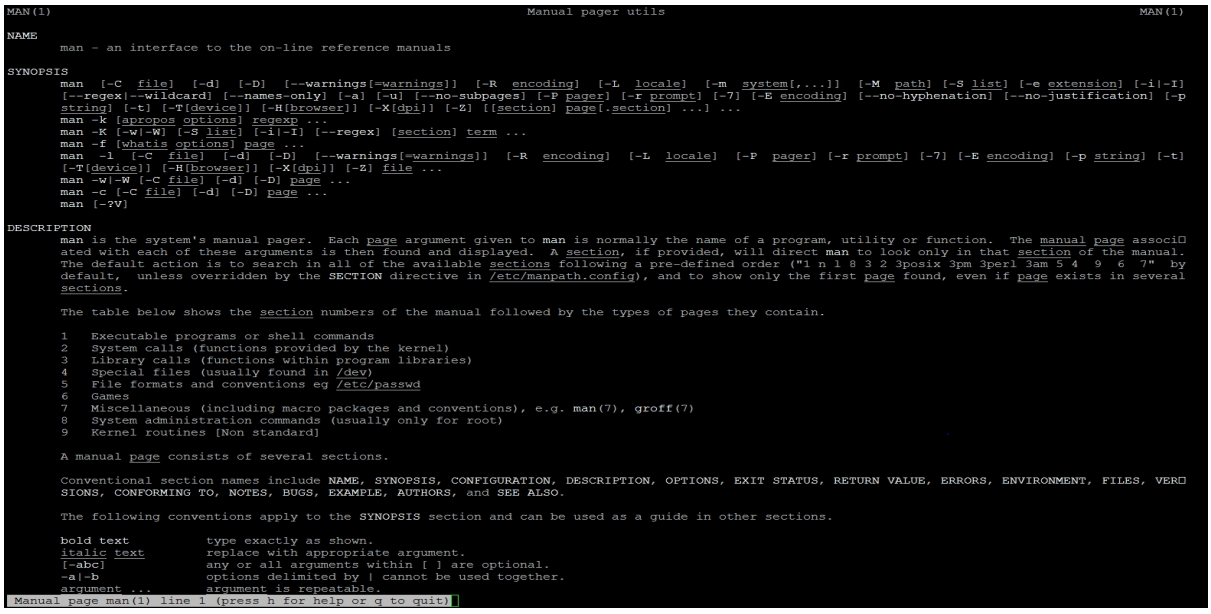
shutdown brings the system down in a secure way. All logged-in users are notified that the system is going down, and **login**(1) is blocked. It is possible to shut the system down immediately or after a specified delay.

OUTPUT SCREEN SHOTS:

man command and its options :

man command:

syntax : man man



man command options :

-f option : One may not be able to remember the sections in which a command is present. So this option gives the section in which the given command is present.

Syntax : man -f [command name]

```
ubuntu@ip-172-31-31-109:~$ man -f ls
ls (1)               - list directory contents
ubuntu@ip-172-31-31-109:~$
```

-a option: This option helps us to display all the available intro manual pages in succession.

Syantax : man -a [command name]

```
ubuntu@ip-172-31-31-109:~$ man -a
What manual page do you want?
```

-k option: This option searches the given command as a regular expression in all the manuals and it returns the manual pages with the section number in which it is found.

Syntax : man -k [command name]

```
ubuntu@ip-172-31-31-109:~$ man -k ls
add-shell (8)        - add shells to the list of valid login shells
blockdev (8)         - call block device ioctls from the command line
credentials (7)      - process identifiers
dircolors (1)        - color setup for ls
eatmydata (1)        - transparently disable fsync() and other data-to-disk synchronization calls
false (1)            - do nothing, unsuccessfully
git-credential (1)   - Retrieve and store user credentials
git-credential-cache--daemon (1) - Temporarily store user credentials in memory
git-credential-store (1) - Helper to store credentials on disk
git-difftool (1)     - Show changes using common diff tools
git-ls-files (1)     - Show information about files in the index and the working tree
git-ls-remote (1)    - List references in a remote repository
git-ls-tree (1)      - List the contents of a tree object
git-mailsplit (1)    - Simple UNIX mbox splitter program
git-mergetool (1)    - Run merge conflict resolution tools to resolve merge conflicts
git-mktree (1)       - Build a tree-object from ls-tree formatted text
gitcredentials (7)   - providing usernames and passwords to Git
grub-menulst2cfg (1) - transform legacy menu.lst into grub.cfg
initramfs-tools (8)  - an introduction to writing scripts for mkinitramfs
intro (2)            - introduction to system calls
ip (8)               - show / manipulate routing, network devices, interfaces and tunnels
libvmtools (3)       - vmware shared library
logrotate (5)        - rotates, compresses, and mails system logs
logrotate (8)        - rotates, compresses, and mails system logs
logrotate.conf (5)   - rotates, compresses, and mails system logs
ls (1)               - list directory contents
lsattr (1)           - list file attributes on a Linux second extended file system
lsb_release (1)      - print distribution-specific information
lsblk (8)            - list block devices
lscpu (1)            - display information about the CPU architecture
lshw (1)             - list hardware
```

-w option: This option returns the location in which the manual page of a given command is present.

Syntax : man -w [command name]

```
ubuntu@ip-172-31-31-109:~$ man -w ls
/usr/share/man/man1/ls.1.gz
ubuntu@ip-172-31-31-109:~$
```

who command and its options:

who command :

```
ubuntu@ip-172-31-31-109:~$ who
ubuntu   pts/0        2020-12-11 06:56 (157.47.68.67)
ubuntu@ip-172-31-31-109:~$
```

who command options :

-H option : This option -H, is used to display the headings of the columns that were displayed in who command.

```
ubuntu@ip-172-31-31-109:~$ who -H
NAME     LINE         TIME             COMMENT
ubuntu   pts/0        2020-12-11 06:56 (157.47.68.67)
ubuntu@ip-172-31-31-109:~$
```

-q option : This option will allow to display the login names and total number of users logged on the system.

```
ubuntu@ip-172-31-31-109:~$ who -q
ubuntu
# users=1
ubuntu@ip-172-31-31-109:~$
```

-r option : To check the current processing run-level.

```
ubuntu@ip-172-31-31-109:~$ who -r
         run-level 5  2020-12-11 06:51
```

-b option : To see the user and time at which the system was last booted . When -b is used with -c it will allow for listing of users logged in the output.

```
ubuntu@ip-172-31-31-109:~$ who -b
         system boot  2020-12-11 06:50
ubuntu@ip-172-31-31-109:~$
```

cat command and its options

cat command :

```
ubuntu@ip-172-31-31-109:~$ cat file2
This is file2
this is created using cat command
ubuntu@ip-172-31-31-109:~$
```

cat command options :

-n option : number of output lines.

```
ubuntu@ip-172-31-31-109:~$ cat -n file2
     1  This is file2
     2  this is created using cat command
ubuntu@ip-172-31-31-109:~$
```

-s option : suppress repeated output lines that are empty.

```
ubuntu@ip-172-31-31-109:~$ cat -s file2
This is file2
this is created using cat command
ubuntu@ip-172-31-31-109:~$
```

cd command and its utilities:

cd command :

```
ubuntu@ip-172-31-31-109:~$ mkdir directory
ubuntu@ip-172-31-31-109:~$ cd directory
ubuntu@ip-172-31-31-109:~/directory$
```

cd ~ :

Your home directory is the directory you're placed in, by default, when you open a new terminal session. It's the directory that holds all your settings, your mail, your default documents and downloads folder, and other personal items. It has a special representation: a tilde ("**~**").

```
ubuntu@ip-172-31-31-109:~/directory$ cd ~
ubuntu@ip-172-31-31-109:~$
```

cd .. :

The parent directory of the current directory — in other words, the directory one level up from the current directory, which contains the directory we're in now — is represented by two dots ("**..**").

```
ubuntu@ip-172-31-31-109:~/directory$ cd ..
ubuntu@ip-172-31-31-109:~$ cd directory
```

cp command and its options :

cp command :

```
ubuntu@ip-172-31-31-109:~$ cat file1
File is created
ubuntu@ip-172-31-31-109:~$ cat file2
This is file2
this is created using cat command
ubuntu@ip-172-31-31-109:~$ cp file1 file2
ubuntu@ip-172-31-31-109:~$ cat file2
File is created
```

cp command options :

-r option : It helps to copy files recursively.

```
ubuntu@ip-172-31-31-109:~$ cp -r file1 file2
ubuntu@ip-172-31-31-109:~$ cat file1
File is created
ubuntu@ip-172-31-31-109:~$
```

-rt option: It copies whole directory into another directory.

```
ubuntu@ip-172-31-31-109:~$ cp -rt /home/ubuntu/directory /home/ubuntu/directory2
ubuntu@ip-172-31-31-109:~$ cd directory2
ubuntu@ip-172-31-31-109:~/directory2$ ls
ubuntu@ip-172-31-31-109:~/directory2$ cd ..
ubuntu@ip-172-31-31-109:~$ cd directory
ubuntu@ip-172-31-31-109:~/directory$ ls
directory2  file3
ubuntu@ip-172-31-31-109:~/directory$
```

ps command and its options :

ps command :

```
ubuntu@ip-172-31-31-109:~/directory$ ps
  PID TTY          TIME CMD
 1811 pts/0    00:00:00 bash
 1975 pts/0    00:00:00 ps
ubuntu@ip-172-31-31-109:~/directory$
```

ps command options :

-a option : Displays all processes on a terminal, with the exception of group leaders.

```
ubuntu@ip-172-31-31-109:~/directory$ ps -a
  PID TTY          TIME CMD
 1980 pts/0    00:00:00 ps
```

-c option : Displays scheduler data.

```
ubuntu@ip-172-31-31-109:~/directory$ ps -c
  PID CLS PRI TTY          TIME CMD
 1811 TS   19 pts/0    00:00:00 bash
 1981 TS   19 pts/0    00:00:00 ps
ubuntu@ip-172-31-31-109:~/directory$
```

-d option : Displays all processes with the exception of session leaders.

```
ubuntu@ip-172-31-31-109:~/directory$ ps -d
  PID TTY          TIME CMD
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 rcu_gp
    4 ?        00:00:00 rcu_par_gp
    6 ?        00:00:00 kworker/0:0H-kb
    9 ?        00:00:00 mm_percpu_wq
   10 ?        00:00:00 ksoftirqd/0
   11 ?        00:00:00 rcu_sched
   12 ?        00:00:00 migration/0
   13 ?        00:00:00 cpuhp/0
   14 ?        00:00:00 kdevtmpfs
   15 ?        00:00:00 netns
   16 ?        00:00:00 rcu_tasks_kthre
   17 ?        00:00:00 kauditd
   18 ?        00:00:00 xenbus
   19 ?        00:00:00 xenwatch
   20 ?        00:00:00 khungtaskd
   21 ?        00:00:00 oom_reaper
   22 ?        00:00:00 writeback
   23 ?        00:00:00 kcompactd0
   24 ?        00:00:00 ksmd
   25 ?        00:00:00 khugepaged
   71 ?        00:00:00 kintegrityd
   72 ?        00:00:00 kblockd
   73 ?        00:00:00 blkcg_punt_bio
   74 ?        00:00:00 tpm_dev_wq
   75 ?        00:00:00 ata_sff
```

-j option : Displays the process group ID and session ID.

```
ubuntu@ip-172-31-31-109:~/directory$ ps -j
  PID  PGID   SID TTY          TIME CMD
 1811  1811  1811 pts/0    00:00:00 bash
 1988  1988  1811 pts/0    00:00:00 ps
ubuntu@ip-172-31-31-109:~/directory$
```

ls command and its options :

ls command :

```
ubuntu@ip-172-31-31-109:~/directory$ ls
directory2  file3
ubuntu@ip-172-31-31-109:~/directory$
```

ls command options :

-a option : It shows all the files including the current directory (.) and parent directory (..).

```
ubuntu@ip-172-31-31-109:~/directory$ ls -a
.  ..  directory2  file3
ubuntu@ip-172-31-31-109:~/directory$
```

-l option : use a long listing format.

```
ubuntu@ip-172-31-31-109:~/directory$ ls -l
total 8
drwxrwxr-x 2 ubuntu ubuntu 4096 Dec 11 07:38 directory2
-rw-rw-r-- 1 ubuntu ubuntu   63 Dec 11 07:36 file3
ubuntu@ip-172-31-31-109:~/directory$
```

-r option : It displays reverse order while sorting.

```
ubuntu@ip-172-31-31-109:~/directory$ ls -r
file3  directory2
ubuntu@ip-172-31-31-109:~/directory$
```

-t option : It displays list sorted by modification time.

```
ubuntu@ip-172-31-31-109:~/directory$ ls -t
directory2  file3
ubuntu@ip-172-31-31-109:~/directory$
```

mv command and its options :

mv command :

```
ubuntu@ip-172-31-31-109:~$ ls
directory  directory2  file1  file2
ubuntu@ip-172-31-31-109:~$ mv file1 directory2
ubuntu@ip-172-31-31-109:~$ ls
directory  directory2  file2
ubuntu@ip-172-31-31-109:~$ ls /directory2
ls: cannot access '/directory2': No such file or directory
ubuntu@ip-172-31-31-109:~$ ls directory2
file1
ubuntu@ip-172-31-31-109:~$
```

mv command options :

-i option: Like in cp ,the -i option makes the command ask the user for confirmation before moving a file that would overwrite an existing file, you have to press **y** for confirm moving, any other key leaves the file as it is.

```
ubuntu@ip-172-31-31-109:~$ mv -i directory2/file1 directory
ubuntu@ip-172-31-31-109:~$ ls directory2
ubuntu@ip-172-31-31-109:~$ ls directory
directory2  file1  file3
ubuntu@ip-172-31-31-109:~$
```

-f option: With -f option we can move files into directories.

```
ubuntu@ip-172-31-31-109:~$ mv -f directory/file1 /home/ubuntu
ubuntu@ip-172-31-31-109:~$ ls
directory  directory2  file1  file2
ubuntu@ip-172-31-31-109:~$
```

rm command and its options :

rm command :

```
ubuntu@ip-172-31-31-109:~$ rm file1
ubuntu@ip-172-31-31-109:~$ ls
directory  directory2  file2
ubuntu@ip-172-31-31-109:~$ cat file1
cat: file1: No such file or directory
ubuntu@ip-172-31-31-109:~$
```

rm command options :

-f option : You will not be prompted, even if the file is write-protected; if rm can delete the file, it will.

```
ubuntu@ip-172-31-31-109:~$ ls
directory  directory2  file2
ubuntu@ip-172-31-31-109:~$ rm -f file2
ubuntu@ip-172-31-31-109:~$ ls
directory  directory2
ubuntu@ip-172-31-31-109:~$
```

mkdir command :

```
ubuntu@ip-172-31-31-109:~$ ls
directory  directory2
ubuntu@ip-172-31-31-109:~$ mkdir directory3
ubuntu@ip-172-31-31-109:~$ ls
directory  directory2  directory3
ubuntu@ip-172-31-31-109:~$
```

rmdir command :rmdir is used to remove empty directory

```
ubuntu@ip-172-31-31-109:~$ rmdir directory3
ubuntu@ip-172-31-31-109:~$ ls
directory  directory2
ubuntu@ip-172-31-31-109:~$
```

-p option: This option is used to remove directory and its ancestors.

```
ubuntu@ip-172-31-31-109:~$ mkdir a
ubuntu@ip-172-31-31-109:~$ cd a
ubuntu@ip-172-31-31-109:~/a$ mkdir b
ubuntu@ip-172-31-31-109:~/a$ cd b
ubuntu@ip-172-31-31-109:~/a/b$ mkdir c
ubuntu@ip-172-31-31-109:~/a/b$ cd c
ubuntu@ip-172-31-31-109:~/a/b/c$ cd ..
ubuntu@ip-172-31-31-109:~/a/b$ cd ~
```

```
ubuntu@ip-172-31-31-109:~$ rmdir -p a/b/c
ubuntu@ip-172-31-31-109:~$ ls
directory  directory2
ubuntu@ip-172-31-31-109:~$
```

echo command and its options :

echo command :

```
ubuntu@ip-172-31-31-109:~$ echo "HelloWorld"
HelloWorld
```

echo command option :

-n option : This option is used to omit trailing newline.

```
ubuntu@ip-172-31-31-109:~$ echo -n "HelloWorld"
HelloWorldubuntu@ip-172-31-31-109:~$
```

-e option : Enable interpretation of backslash escape sequences.

```
ubuntu@ip-172-31-31-109:~$ echo -e "hello\nworld"
hello
world
ubuntu@ip-172-31-31-109:~$ echo -e "hello\tworld"
hello   world
ubuntu@ip-172-31-31-109:~$ echo -e "hello\cworld"
helloubuntu@ip-172-31-31-109:~$ echo -e "hello\rworld"
world
ubuntu@ip-172-31-31-109:~$ echo -e "hello\bworld"
hellworld
ubuntu@ip-172-31-31-109:~$
```

more command and its options :

more command:

```
ubuntu@ip-172-31-31-109:~$ more subjectlist
Operting system
Unix Programming
Object Oriented analasys and design
Database management system
Compiler Design
ubuntu@ip-172-31-31-109:~$
```

more command options :

-p option : Do not scroll. Instead, clear the whole screen and then display the text.

```
ubuntu@ip-172-31-31-109: ~
Operting system
Unix Programming
Object Oriented analasys and design
Database management system
Compiler Design
ubuntu@ip-172-31-31-109:~$
```

-c option : Do not scroll. Instead, paint each screen from the top, clearing the remainder of each line as it is displayed.

```
ubuntu@ip-172-31-31-109: ~
Operting system
Unix Programming
Object Oriented analasys and design
Database management system
Compiler Design
ubuntu@ip-172-31-31-109:~$
```

date command and its options :

date command :

```
ubuntu@ip-172-31-31-109:~$ date
Fri Dec 11 08:21:39 UTC 2020
ubuntu@ip-172-31-31-109:~$
```

date command options :

-u option : Displays the time in GMT(Greenwich Mean Time)/UTC(Coordinated Universal Time)time zone.

```
ubuntu@ip-172-31-31-109:~$ date -u
Fri Dec 11 08:22:07 UTC 2020
ubuntu@ip-172-31-31-109:~$
```

--date option : Displays the time in GMT(Greenwich Mean Time)/UTC(Coordinated Universal Time)time zone.

```
ubuntu@ip-172-31-31-109:~$ date --date="12/12/2020"
Sat Dec 12 00:00:00 UTC 2020
ubuntu@ip-172-31-31-109:~$
```

time command :time command is used to display time for any command execution time.

```
ubuntu@ip-172-31-31-109:~$ time sleep 5

real    0m5.002s
user    0m0.001s
sys     0m0.000s
ubuntu@ip-172-31-31-109:~$
```

kill command :kill command is used to kill an process which is running using process id.

```
ubuntu@ip-172-31-31-109:~$ sleep 5 &
[1] 2099
ubuntu@ip-172-31-31-109:~$ ps
  PID TTY          TIME CMD
 1811 pts/0    00:00:00 bash
 2099 pts/0    00:00:00 sleep
 2100 pts/0    00:00:00 ps
ubuntu@ip-172-31-31-109:~$ kill 2099
-bash: kill: (2099) - No such process
[1]+  Done                    sleep 5
ubuntu@ip-172-31-31-109:~$ ps
  PID TTY          TIME CMD
 1811 pts/0    00:00:00 bash
 2101 pts/0    00:00:00 ps
ubuntu@ip-172-31-31-109:~$
```

history command : This command displays list of commands used till now in a session on unix os.

```
ubuntu@ip-172-31-31-109:~$ history
    1  ls
    2  cat>file1
    3  ls
    4  cat file1
    5  ls
    6  clear
    7  man man
    8  man -f ls
    9  man -a
   10  ls
   11  man -k ls
   12  man -w ls
   13  who
   14  who -H
   15  who -q
   16  who -r
   17  who -b
   18  ls
   19  cat>file2
   20  cat file2
   21  cat -n file2
   22  cat -s file2
   23  mkdir directory
   24  cd directory
   25  cd .
   26  cd ..
   27  cd directory
   28  cd~
   29  cd ~
   30  ls
   31  cat file1
   32  cat file2
   33  cp file1 file2
   34  cat file2
   35  cat file1
   36  cp -r file1 file2
   37  cat file1
   38  cd directory
   39  ls
   40  cd ..
```

chmod command and its options :

chmod command : This command is used to change the permissions of a file

rwx

000-No permission-0

100-Only read permission-4

101-Read and execute permission-5

chmod oga filename

O-Owner permissions

G-group permissions

A-All users permissions

```
ubuntu@ip-172-31-31-109:~$ ls -l subjectlist
-rw-rw-r-- 1 ubuntu ubuntu 112 Dec 11 08:18 subjectlist
ubuntu@ip-172-31-31-109:~$ chmod 777 subjectlist
ubuntu@ip-172-31-31-109:~$ ls -l
total 12
drwxrwxr-x 3 ubuntu ubuntu 4096 Dec 11 07:56 directory
drwxrwxr-x 2 ubuntu ubuntu 4096 Dec 11 07:54 directory2
-rwxrwxrwx 1 ubuntu ubuntu  112 Dec 11 08:18 subjectlist
ubuntu@ip-172-31-31-109:~$ ls -l subjectlist
-rwxrwxrwx 1 ubuntu ubuntu 112 Dec 11 08:18 subjectlist
ubuntu@ip-172-31-31-109:~$
```

chmod command options :

a-x option : This option will remove execute permission for all(i.e)owner,group,all users

```
ubuntu@ip-172-31-31-109:~$ ls -l subjectlist
-rwxrwxrwx 1 ubuntu ubuntu 112 Dec 11 08:18 subjectlist
ubuntu@ip-172-31-31-109:~$ chmod a-x subjectlist
ubuntu@ip-172-31-31-109:~$ ls -l subjectlist
-rw-rw-rw- 1 ubuntu ubuntu 112 Dec 11 08:18 subjectlist
ubuntu@ip-172-31-31-109:~$
```

chown command: This command is used to change the owner to a file

```
ubuntu@ip-172-31-31-109:~$ ls -l subjectlist
-rw-rw-rw- 1 ubuntu ubuntu 112 Dec 11 08:18 subjectlist
ubuntu@ip-172-31-31-109:~$ chown root subjectlist
chown: changing ownership of 'subjectlist': Operation not permitted
ubuntu@ip-172-31-31-109:~$
```

pwd command :it displays the present working directory

```
ubuntu@ip-172-31-31-109:~$ pwd
/home/ubuntu
ubuntu@ip-172-31-31-109:~$
```

cal command and its options :

cal command :

```
ubuntu@ip-172-31-31-109:~$ cal
   December 2020
Su Mo Tu We Th Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

ubuntu@ip-172-31-31-109:~$
```

cal command options :

-j option : Display julian dates

```
ubuntu@ip-172-31-31-109:~$ cal -j
       December 2020
 Su   Mo   Tu   We   Th   Fr   Sa
         336  337  338  339  340
341  342  343  344  345  346  347
348  349  350  351  352  353  354
355  356  357  358  359  360  361
362  363  364  365  366

ubuntu@ip-172-31-31-109:~$
```

-y option : Display current year calendar.

```
ubuntu@ip-172-31-31-109:~$ cal -y
                              2020
      January               February                March
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
          1  2  3  4                     1   1  2  3  4  5  6  7
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   8  9 10 11 12 13 14
12 13 14 15 16 17 18   9 10 11 12 13 14 15  15 16 17 18 19 20 21
19 20 21 22 23 24 25  16 17 18 19 20 21 22  22 23 24 25 26 27 28
26 27 28 29 30 31     23 24 25 26 27 28 29  29 30 31


       April                  May                   June
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
          1  2  3  4                  1  2      1  2  3  4  5  6
 5  6  7  8  9 10 11   3  4  5  6  7  8  9   7  8  9 10 11 12 13
12 13 14 15 16 17 18  10 11 12 13 14 15 16  14 15 16 17 18 19 20
19 20 21 22 23 24 25  17 18 19 20 21 22 23  21 22 23 24 25 26 27
26 27 28 29 30        24 25 26 27 28 29 30  28 29 30
                      31

        July                 August              September
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
          1  2  3  4                     1         1  2  3  4  5
 5  6  7  8  9 10 11   2  3  4  5  6  7  8   6  7  8  9 10 11 12
12 13 14 15 16 17 18   9 10 11 12 13 14 15  13 14 15 16 17 18 19
19 20 21 22 23 24 25  16 17 18 19 20 21 22  20 21 22 23 24 25 26
26 27 28 29 30 31     23 24 25 26 27 28 29  27 28 29 30
                      30 31

      October              November             December
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
             1  2  3   1  2  3  4  5  6  7         1  2  3  4  5
 4  5  6  7  8  9 10   8  9 10 11 12 13 14   6  7  8  9 10 11 12
11 12 13 14 15 16 17  15 16 17 18 19 20 21  13 14 15 16 17 18 19
18 19 20 21 22 23 24  22 23 24 25 26 27 28  20 21 22 23 24 25 26
25 26 27 28 29 30 31  29 30                 27 28 29 30 31

ubuntu@ip-172-31-31-109:~$
```

cal year: It displays the complete calender of the year specified in the command

```
ubuntu@ip-172-31-31-109:~$ cal 2018
                                2018
      January               February                March
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6                  1  2  3                  1  2  3
 7  8  9 10 11 12 13   4  5  6  7  8  9 10   4  5  6  7  8  9 10
14 15 16 17 18 19 20  11 12 13 14 15 16 17  11 12 13 14 15 16 17
21 22 23 24 25 26 27  18 19 20 21 22 23 24  18 19 20 21 22 23 24
28 29 30 31           25 26 27 28           25 26 27 28 29 30 31


       April                  May                   June
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7         1  2  3  4  5                  1  2
 8  9 10 11 12 13 14   6  7  8  9 10 11 12   3  4  5  6  7  8  9
15 16 17 18 19 20 21  13 14 15 16 17 18 19  10 11 12 13 14 15 16
22 23 24 25 26 27 28  20 21 22 23 24 25 26  17 18 19 20 21 22 23
29 30                 27 28 29 30 31        24 25 26 27 28 29 30


        July                 August              September
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7            1  2  3  4                     1
 8  9 10 11 12 13 14   5  6  7  8  9 10 11   2  3  4  5  6  7  8
15 16 17 18 19 20 21  12 13 14 15 16 17 18   9 10 11 12 13 14 15
22 23 24 25 26 27 28  19 20 21 22 23 24 25  16 17 18 19 20 21 22
29 30 31              26 27 28 29 30 31     23 24 25 26 27 28 29
                                            30


      October              November              December
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6               1  2  3                     1
 7  8  9 10 11 12 13   4  5  6  7  8  9 10   2  3  4  5  6  7  8
14 15 16 17 18 19 20  11 12 13 14 15 16 17   9 10 11 12 13 14 15
21 22 23 24 25 26 27  18 19 20 21 22 23 24  16 17 18 19 20 21 22
28 29 30 31           25 26 27 28 29 30     23 24 25 26 27 28 29
                                            30 31
ubuntu@ip-172-31-31-109:~$
```

logout command :

No output is displayed on the screen but the user gets out of the current session.

shutdown command :

```
ubuntu@ip-172-31-31-109: ~
ubuntu@ip-172-31-31-109:~$ shutdown
Failed to set wall message, ignoring: Interactive authentication required.
Failed to call ScheduleShutdown in logind, proceeding with immediate shutdown: Interactive authentication required.
Failed to set wall message, ignoring: Interactive authentication required.
Failed to power off system via logind: Interactive authentication required.
Failed to open /dev/initctl: Permission denied
Failed to talk to init daemon.
ubuntu@ip-172-31-31-109:~$
```

EXPERIMENT NO: 1 (b)

AIM : Study of vi editor

DESCRIPTION :

Vi Editor:

- An editor allows the users to see a portion of a file on the screen and to modify characters and lines by simply typing at the cursor position.
- The vi editor represents,
- Vi stands for visual
- It is a full screen editor and allows the user to view and edit the entire document at the same time.
- Vi is case sensitive.
- It has powerful undo features.

Modes of Vi editor:

i) Command mode:

- In this mode all the keys pressed by the user are interpreted to be editor commands.
- No text is displayed on the screen even if corresponding keys is pressed on the keyboard.

ii) Insert mode:

- This mode permits to insert a new text, editing and replacement of existing text.
- When vi editor is in insert mode the letters typed at the keyboard are echoed on the screen.

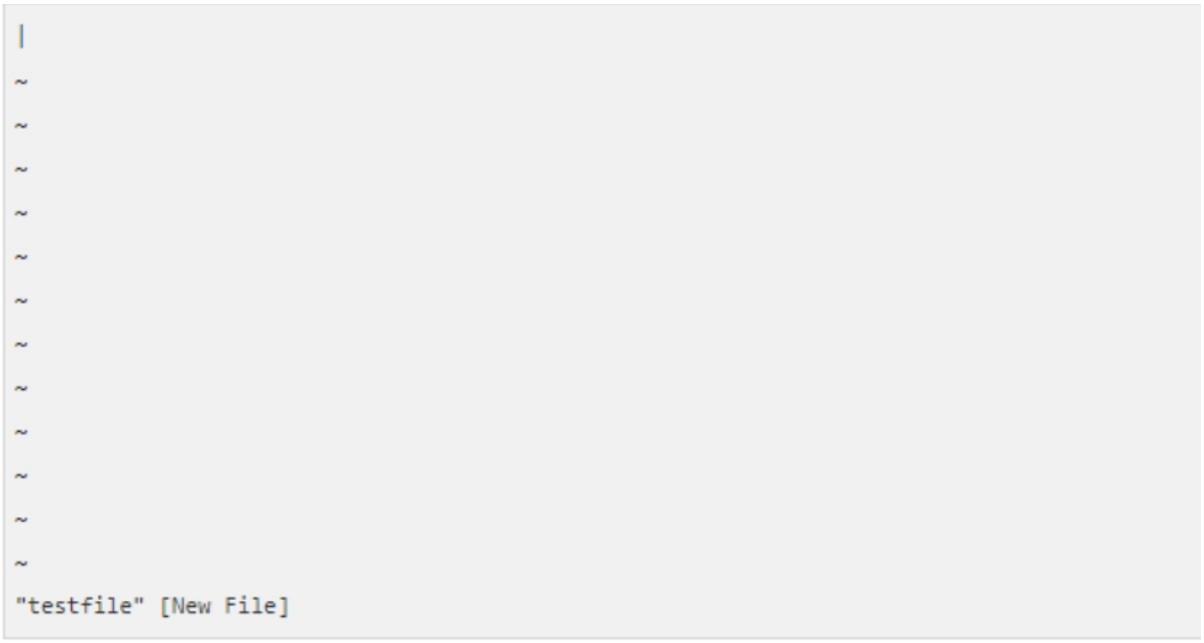
iii) Escape mode:

- Commands typed at the command line.

Starting with vi editor:

Syntax:

$vi filename



Editing the file:

- Open the file using $ vi filename
- To add text at the end of the file, position the cursor at the last character of the file.
- Switch from command mode to text input mode by pressing 'a'.
- Here 'a' stands for append.
- Inserting text in the middle of the file is possible by pressing 'i'.
- The editor accepts and inserts the typed character until Esc key is pressed.

Saving text:

:w – save the file and remains in edit mode
:wq – save the file and quits from edit mode
:q – quit without changes from edit mode

Quitting vi:

Press zz or ':wq' in command mode.

EXPERIMENT NO: 1 (c)

AIM : Study of Bash shell, Bourne shell and C shell in Unix/Linux operating system.

DESCRIPTION :

Types of Shells:

- The Shells are 4 types

a) The Bourne Shell (sh)
b) The C Shell (csh)
c) The Korn Shell (ksh)
d) The Bourne-Again Shell (bash)

a) The Bourne Shell (sh):

- This is the most common shell available on Unix Systems.
- The first major shell developed by Stephen Bourne at AT&T Bell labs.
- This shell is widely used.
- This shell is distributed as the standard shell on almost all Unix Systems.

b) The C Shell (csh):

- It is developed by Bill Joy at UCB as part of the BSD release.
- Its syntax and usage is very similar to the C programming language.
- This shell is not available on all machines.
- Shell scripts written in the C shell are not compatible with the Bourne Shell.
- A version of it called tcsh is available free of cost under Linux.

c) The Korn Shell (ksh):

- This shell was developed by David Korn at AT&T Bell labs.
- Basically it is built on the Bourne Shell.
- It also incorporates certain features of the C shell.
- At present it is one of the widely used shells.
- It can run Bourne shell scripts without any modification.
- One of its versions, the Public Domain Korn Shell (pdksh), comes with Linux free of cost.

d) The Bourne-Again Shell:

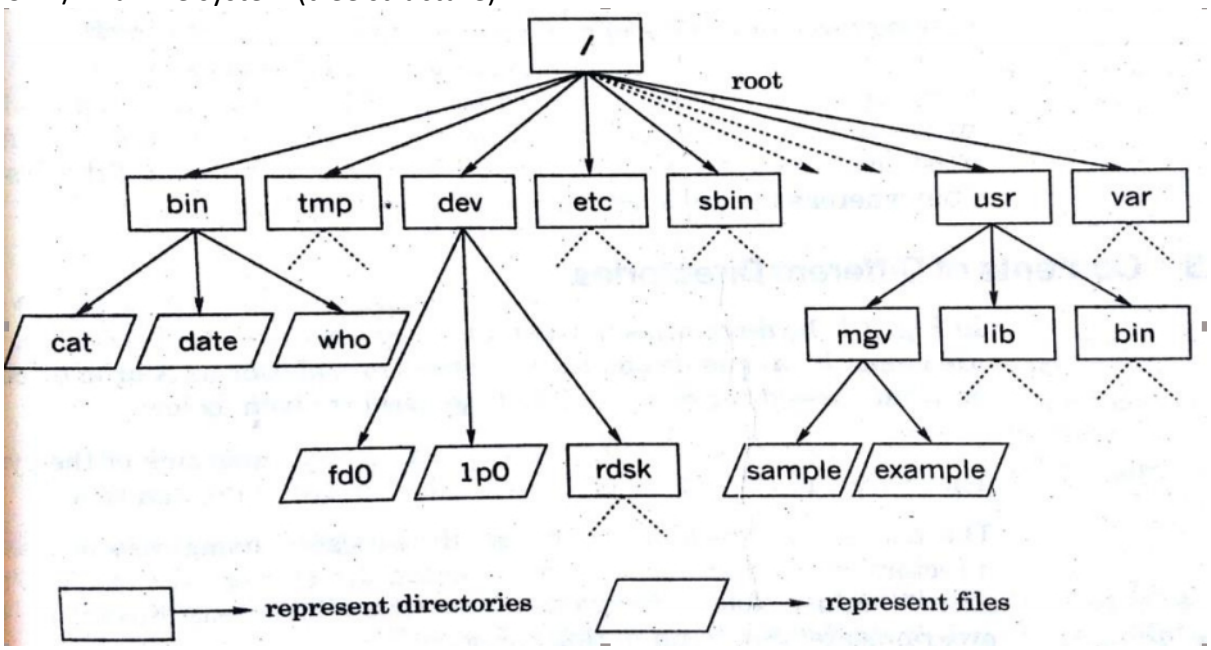
- It was developed by B Fox and C Ramey at Free Software Foundation.
- Certain Linux operating system variants come with this shell as its default shell.
- This is clearly a free ware shell.

EXPERIMENT NO: 1 (d)

AIM : Study of Unix/Linux file system (tree structure).

DESCRIPTION :

Unix/Linux file system (tree structure):



- It is also called as UNIX File System, because it is used to organize UNIX files.
- The different directories in the Directory Hierarchy are,

DIRECTORY	FILE STRUCTURE
root (/)	Root of the file system
/bin	Essential programs in executable form (binaries)
/tmp	Temporary files; cleaned when system is restarted
/dev	Device files
/etc	System miscellany
/sbin	System binaries

/usr	User file system
/var	Variables
/usr/lib	Libraries for C, FORTRAN. Etc
/usr/bin	User binaries

EXPERIMENT NO: 1 (e)

AIM : Study of .bashrc, /etc/bashrc and Environment variables.

DESCRIPTION :

.bashrc

.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

• Like .bash_profile you will also commonly see a .bashrc file in your home directory. This file is meant for setting command aliases and functions used by bash shell users.

• Just like the /etc/profile is the system wide version of .bash_profile. 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 a Interactive Shell (aka Login Shell)

Environment Variables

Variable	**Purpose**
PATH	The PATH variable holds a list of directories in a certain order. In this list colon (:) separate different directories
HOME	When a user logs in, he or she will be automatically placed in the home directory. This directory is decided by the system administrator at the time of opening an account for a user.This

	directory is stored in the file /etc/passwd
IFS	This variable holds tokens used by the shell commands to parse a string into substrings such as a word or a record into its individual fields. The default tokens are the three white space tokens Space, Tab, New line
MAIL	This variable holds the absolute pathname of the file where user's mail is kept. Usually the name of this file is the user's login name
SHELL	This variable contains the name of the users shell program in the form of absolute pathname. System administrator sets the default shell If required, user can change it
TERM	This variable holds the information regarding the type of the terminal being used. If TERM is not set properly, utilities like vi editor will not work

EXPERIMENT NO:2

AIM: TowriteaCprogramthatmakesacopyofafileusingstandardI/O,andsystemcalls.

DESCRIPTION:

In the following program we use the I/O and system calls to copy content of one file to another.To
Achieve this use some headerfiles.They are unistd.h,fcntl.h,stdio.h,stdlib.h.
The methods used are:
stdio.h-->perror()
unistd.h-->read(),write()
fcntl.h-->open()
stdlib.h-->exit()
fcntl.h-filecontroloptions
unistd.h-standardsymbolicconstantsandtypes
stdlib.h-standardlibrarydefinitions
stdio.h-standardbufferedinput/output

LIBRARIESUSED:

stdio.h,unistd.h,fcntl.h,stdlib.h
All these header files included libraries to execute different systemcalls required to write this program

SYNTAX:

Syntax for using open :

int open(const char *_path_, int_oflag_, ...);

The *open*() function shall establish the connection between a file and a file descriptor.It shall create an open file description that refers to a file and a file descriptor that refers to that open file description.The file descriptor is used by other I/O functions to refer to that file.The path argument points to a pathname naming the file.
The *open*() function shall return a file descriptor for the named file that is the lowest file descriptor not currently open for that process. The open file description is new, and therefore the file descriptor shall not share it with any other process in the system.The FD_CLOEXEC file descriptor flag associated with the new file descriptor shall be cleared.

O_RDONLY:
Open for reading only.
O_WRONLY:
Open for writing only.
O_CREAT:
If the file exists,this flag has no effect.Otherwise,the file shall be created;the userID of the file shall be set to the effective userID of the process;the groupID of the file shall be set to the group ID of the file's parent directory or to the effective groupID of the process; and the access permission bits of the file mode shall be set to

the value of the third argument taken as type **mode_t** modified.
O_TRUNC:

If the file exists and is a regular file, and the file is successfully opened O_RDWR or O_WRONLY,its length shall be truncated to 0,and the mode and owner shall be unchanged.It shall have no effecton FIFO special files or terminal device files.Its effect on other file types is implementation-defined.The result of using O_TRUNC with O_RDONLY is undefined.

Syntax for using read,write :

ssize_t read(int, void *,size_t);

ssize_t write(int fildes, const void *buf, size_tnbyte);

The write() function shall attempt to write *n* bytes from the buffer pointed to by buf to the file associated with the open file descriptor,fildes.
Before any action is taken,and if nbyte is zero and the file is a regular file,the write() function may detect and return error.In the absence of errors,or if error detection is not performed,the write() function shall return zero and have no other results.If nbyte is zero and the file is not a regular file, the results are unspecified.

On a regular file or other file capable of seeking, the actual writing of data shall proceed from the position in the file indicated by the file off set associated with fildes.Before successful return from write(), the file offset shall be incremented by the number of bytes actually written. On a regular file, if the position of the last byte written is greater than or equal to the length of the file,the length of the file shall be set to this position plus one.

The read() function shall attempt to read *nbyte* bytes from the file associated with the openfile descriptor,fildes,into the buffer pointed to by buf. The behavior of multiple concurrent reads on the same pipe, FIFO, or terminal device isunspecified.Before any action is taken,and if nbyte is zero,the read() function may detect and return errors. In the absence of errors,or if error detection is not performed,the *read*()function shall return zero and have no other results.

On files that support seeking(for example,a regular file),the *read*() shall start at a position in the file given by the file offset associated with fildes.The file offset shall be incremented by the number of bytes actually read.

Syntax for using exit:

void exit(int status):

The value of status may be 0,EXIT_SUCCESS,EXIT_FAILURE,or any other value,though only the least significant 8bits(that is,status&0377)shall be available from wait() and waitpid(); the full value shall be available from waitid() and in the siginfo_t passed to a signal handler for SIGCHLD.

Syntax for using perror :

void perror(const char *s);

The *perror*() function shall map the error number accessed through the symbol *errno* to a language- dependent error message,which shall be written to the standard error stream as follows:

*First (if *s* is not a null pointer and the character pointed to by *s* is not the nullbyte), the string pointed to by *s* followed by a <colon> and a<space>.

*Then an error message string followed by a<newline>.The contents of the error message strings shall be the same as those returned by strerror() with argument *errno*.

PROGRAMMING LANGUAGE USED:C

PROGRAM:

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<stdlib.h>
intmain(intargc,char*argv[])
{
intf1,f2;
charbuff[50];
longintn;
if(((f1=open(argv[1],O_RDONLY))==-1||((f2=open(argv[2],O_CREAT|
O_WRONLY|O_TRUNC,0700))==1)))
{
perror("probleminfile");
exit(1);
}
while((n=read(f1,buff,50))>0)
if(write(f2,buff,n)!=n)
{
perror("probleminwriting");
exit(3);
}
if(n==-1)
{
perror("probleminreading");
exit(2);
```

```
}
close(f2);
exit(0);
}
```

OUTPUT:

```
$gccfilecopy.c
$./a.outfilecopy.cdest.txt
$catdest.txt
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<stdlib.h>
intmain(intargc,char*argv[])
{
intf1,f2;
charbuff[50];
longintn;
if(((f1=open(argv[1],O_RDONLY))==-1||((f2=open(argv[2],O_CREAT|O_WRONLY|
O_TRUNC,0700))==1)))
{
perror("probleminfile");
exit(1);
}
while((n=read(f1,buff,50))>0)
if(write(f2,buff,n)!=n)
{
perror("probleminwriting");
exit(3);
}
if(n==-1)
{
perror("probleminreading");
exit(2);
}
close(f2);
exit(0);
}
```

OUTPUT SCREENSHOT:

```
$ gcc filecopy.c
$ ./a.out filecopy.c dest.txt
$ cat dest.txt
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
int f1, f2;
char buff[50];
long int n;
if(((f1 = open(argv[1], O_RDONLY)) == -1 || ((f2=open(argv[2], O_CREAT | O_WRONLY | O_TRUNC, 0700))== 1
)))
{
perror("problem in file");
exit(1);
}
while((n=read(f1, buff, 50))>0)
if(write(f2, buff, n)!=n)
{
perror("problem in writing");
exit(3);
}
if(n==-1)
{
perror("problem in reading");
exit(2);
}
close(f2);
exit(0);
}
$
```

EXPERIMENTNO:3

AIM: TowriteaCprogramtoemulatetheUNIXls–lcommand.

DESCRIPTION:

In this program we are supposed to print the long listing descriptoion of files in file system hyerarchy.

In UNIX when we enter ls -l command a long list of all files available in current / working are displayed.It is displayed in 7 fields.Each field contain some information about that file.

- The first field is the filepermissions.
- Second one is noof links to that file(soft or hardlinks)
- The third one isUser
- Fourth isgroup
- Fifth is size offile
- Sixth is date created or lastmodified
- The last one is the file nameitself

To obtain some of the file properties we use some headers and inbuild functionalities

In this program we used the following header files.

- stdio.h
- fcntl.h
- time.h
- sys/stat.h

We already discussed about stdio.h and fcntl.h in the previous program to copy file

So we shall see time.h and sys/stat.h for now.

time.h --> time types

sys/stat.h -->data returned by the stat() function

From time header we use ctime function.

From sys/stst we use some

symbolicconstants.

LIBRARIESUSED:

stdio.h,sys/stat.h,fcntl.h,time.h

All these header files included libraries to execute different systemcalls required to write this program

SYNTAX:

Syntax for using ctime:

```
char *ctime(consttime_t *);
```

Itshallreturnvaluesinoneoftwostaticobjects:abroken-downtimestructureandanarrayof char.Executionofanyofthefunctionsmayoverwritetheinformationreturnedineitherofthese objects by any of the otherfunctions.

The *ctime*() function need not be thread-safe.

The*ctime_r*()functionshallconvertthecalendartimepointedtoby*clock*tolocaltimeinexactly the same form as *ctime*() and put the string into the array pointed to by *buf*(which

shall be at least 26 bytes in size) and return*buf*.

The *ctime*() function shall return the pointer.

It contains tm structure which contains the members:

tm_sec,tm_min,tm_hour,tm_mday,...,tm_isdst.

In the program we use ctime function.

Constants in sys/stat.h:

Name	Numeric Value	Description
S_IRWXU	0700	Read, write, execute/search by owner.
S_IRUSR	0400	Read permission, owner.
S_IWUSR	0200	Write permission, owner.
S_IXUSR	0100	Execute/search permission, owner.
S_IRWXG	070	Read, write, execute/search by group.
S_IRGRP	040	Read permission, group.
S_IWGRP	020	Write permission, group.
S_IXGRP	010	Execute/search permission, group.
S_IRWXO	07	Read, write, execute/search by others.
S_IROTH	04	Read permission, others.
S_IWOTH	02	Write permission, others.
S_IXOTH	01	Execute/search permission, others.
S_ISUID	04000	Set-user-ID on execution.
S_ISGID	02000	Set-group-ID on execution.
S_ISVTX	01000	On directories, restricted deletion flag.

Also to check the type of file we use:

S_ISBLK(*m*)
Test for a block special file.
S_ISCHR(*m*)
Test for a character special file.
S_ISDIR(*m*)
Test for a directory.
S_ISFIFO(*m*)
Test for a pipe or FIFO special file.
S_ISREG(*m*)
Test for a regular file.
S_ISLNK(*m*)
Test for a symbolic link.
S_ISSOCK(*m*)
Test for a socket.

PROGRAMMING LANGUAGE USED:C
PROGRAM:

```
#include<stdio.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<time.h>
intmain(intargc,char*argv[])
{
char*at,*mt,*ct;inti;
structstatbuf;
for(i=1;i<argc;i++)
{
stat(argv[i],&buf);
printf("\n_____%s_____\n",argv[i]);
printf("theinodenois:%d\n",buf.st_ino);
if(S_ISDIR(buf.st_mode))
printf("itisadirectory");
if(S_ISREG(buf.st_mode))
printf("itisaregularfile\n");
printf("thenooflinksis%d\n",buf.st_nlink);
printf("sizeofthefile=%d\n",buf.st_size);
at=ctime(&buf.st_atime);
printf("thetimeoflastaccessis%s\n",at);
mt=ctime(&buf.st_mtime);
printf("themodificationtimeis:%s\n",mt);
ct=ctime(&buf.st_ctime);
printf("timeofcreationis:%s\n",ct);
```

```
if((buf.st_mode&S_IRUSR)==S_IRUSR)
printf("userhasreadpermission\n");
if((buf.st_mode&S_IWUSR)==S_IWUSR)
printf("userhaswritepermission\n");
if((buf.st_mode&S_IXUSR)==S_IXUSR)
printf("userhasexecutepermission\n");
if((buf.st_mode&S_IRGRP)==S_IRGRP)
printf("grouphasreadpermission\n");
if((buf.st_mode&S_IWGRP)==S_IWGRP)
printf("grouphaswritepermission\n");
if((buf.st_mode&S_IXGRP)==S_IXGRP)
printf("grouphasexecutepermission\n");
if((buf.st_mode&S_IROTH)==S_IROTH)
printf("othershasreadpermission\n");
if((buf.st_mode&S_IWOTH)==S_IWOTH)
printf("othershaswritepermission\n");
if((buf.st_mode&S_IXOTH)==S_IXOTH)
printf("othershasexecutepermission\n");
}
}
```

OUTPUT:

```
$gccexp3.c
exp3.c:13:33:warning:formatspecifiestype'int'
buttheargumenthastype'ino_t'(aka
'unsignedlong')[-Wformat]
printf("theinodenois:%d\n",buf.st_ino);
~~ ^~~~~~~~~~
%lu
exp3.c:19:32:warning:formatspecifiestype'int'
buttheargumenthastype'off_t'
(aka'long')[-Wformat]
printf("sizeofthefile=%d\n",buf.st_size);
~~ ^~~~~~~~~~~
%ld
2warningsgenerated.
$./a.outfile1file4
___________file1____________
theinodenois:2662462
itisaregularfile
thenooflinksis1
sizeofthefile=138
thetimeoflastaccessisThuOct123:11:302020
themodificationtimeis:ThuOct123:11:462020
timeofcreationis:TueDec2217:21:422020
```

userhasreadpermission
userhaswritepermission
userhasexecutepermission
grouphasreadpermission
grouphaswritepermission
othershasreadpermission
othershasexecutepermission
_____file4_____
theinodenois:2663307
itisaregularfile
thenooflinksis1
sizeofthefile=59
thetimeoflastaccessisSunAug2321:29:452020
themodificationtimeis:MonAug2409:37:552020
timeofcreationis:TueDec2217:21:422020
userhasreadpermission
userhaswritepermission
userhasexecutepermission
grouphasreadpermission
grouphaswritepermission
othershasreadpermission
othershasexecutepermission

OUTPUTSCREENSHOT:

```
$ gcc exp3.c
exp3.c:13:33: warning: format specifies type 'int'
      but the argument has type 'ino_t' (aka
      'unsigned long') [-Wformat]
printf("the inode no is : %d\n",buf.st_ino);
                         ~~    ^~~~~~~~~
                         %lu
exp3.c:19:32: warning: format specifies type 'int'
      but the argument has type 'off_t'
      (aka 'long') [-Wformat]
printf("size of the file=%d\n",buf.st_size);
                        ~~     ^~~~~~~~~~
                        %ld
2 warnings generated.
$ ./a.out file1 file4

_____file1_____
the inode no is : 2662462
it is a regular file
the no of links is 1
size of the file=138
the time of last access is Thu Oct  1 23:11:30 2020

the modification time is :Thu Oct  1 23:11:46 2020

time of creation is : Tue Dec 22 17:21:42 2020

user has read permission
user has write permission
user has execute permission
group has read permission
group has write permission
others has read permission
others has execute permission

_____file4_____
the inode no is : 2663307
it is a regular file
the no of links is 1
size of the file=59
the time of last access is Sun Aug 23 21:29:45 2020

the modification time is :Mon Aug 24 09:37:55 2020

time of creation is : Tue Dec 22 17:21:42 2020

user has read permission
user has write permission
user has execute permission
group has read permission
group has write permission
others has read permission
others has execute permission
$
```

EXPERIMENTNO:4

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

DESCRIPTION : In computer programming, especially in UNIX operating systems, a pipe is a technique for passing information from one program process to another. Unlike other forms of interprocess communication (IPC), a pipe is one-way communication only. Basically, a pipe passes a parameter such as the output of one process to another process which accepts it as input. The system temporarily holds the piped information until it is read by the receiving process.

Using a UNIX shell (the UNIX interactive command interface), a pipe is specified in a command line as a simple vertical bar (|) between two command sequences. The output or result of the first command sequence is used as the input to the second command sequence. The *pipe* system call is used in a similar way within a program.

LIBRARIES USED :

#include <stdio.h>

SYNTAX :

FILE-While doing file handling we often use FILE for declaring the pointer in order to point to the file we want to read from or to write on.

popen() – First, each time popen is called, we have to remember the process ID of the child that we create and either its file descriptor or FILE pointer. We choose to save the child's process ID in the array childpid, which we index by the file descriptor.

fgets() –The C library function char *fgets(char *str, int n, FILE *stream) reads a line from the specified stream and stores it into the string pointed to by str. It stops when either (n-1) characters are read, the newline character is read, or the end-of-file is reached, whichever comes first.

fputs() –The C library function int fputs(const char *str, FILE *stream) writes a string to the specified stream up to but not including the null character.

pclose() -This way, when pclose is called with the FILE pointer as its argument, we call the standard I/O function fileno to get the file descriptor, and then have the child process ID for the call to waitpid. Since it's possible for a given process to call popen more than once, we dynamically allocate the childpid array (the first time popen is called), with room for as many children as there are file descriptors

PROGRAM:

```
#include<stdio.h>
intmain()
{
FILE*p=NULL;
FILE*w=NULL;
charbuf[1024];
p=popen("ls-l","r");
w=popen("sort","w");
while(fgets(buf,1024,p)!=NULL)
fputs(buf,w);
pclose(p);
```

```
pclose(w);
}
```

OUTPUT:

```
$ls-l|sort
-rw-------1u0_a232u0_a232 14Sep309:27file5
-rw-------1u0_a232u0_a232 17Oct118:58abc
-rw-------1u0_a232u0_a232 36Dec2819:44file2
-rw-------1u0_a232u0_a232 98Aug2412:56temp2
-rw-------1u0_a232u0_a232 98Aug2710:51temp4
-rw-------1u0_a232u0_a232 157Dec1320:57temp
-rw-------1u0_a232u0_a232 200Jan410:43exp4.c
-rw-------1u0_a232u0_a232 240Sep811:33fs3
-rw-------1u0_a232u0_a232 279Aug2506:55temp3
-rw-------1u0_a232u0_a232 475Dec2217:26exp2.c
-rw-------1u0_a232u0_a232 475Dec2217:27filecopy.c
-rw-------1u0_a232u0_a232 570Jan420:04shmc.c
-rw-------1u0_a232u0_a232 845Jan420:03shms.c
-rw-------1u0_a232u0_a2321386Dec2216:29exp3.c
-rw-------1u0_a232u0_a2321386Dec2219:38experiment3.c
-rw-------1u0_a232u0_a2321661Jan420:42temp5
-rw-------1u0_a232u0_a23212288Aug2619:26typescript
-rwx------1u0_a232u0_a232 475Dec2217:41dest.txt
-rwx------1u0_a232u0_a2328280Jan419:35a.out
-rwx--x--x1u0_a232u0_a232 0Sep2109:24fxile
-rwx--x--x1u0_a232u0_a232 0Sep2109:29fxile2
-rwxr-xr-x1u0_a232u0_a232 47Sep1508:50f1
-rwxrw-r-x1u0_a232u0_a232 59Aug2409:37file4
-rwxrw-r-x1u0_a232u0_a232 138Oct123:11file1
drwx------2u0_a232u0_a2324096Dec2913:43downloads
drwx------2u0_a232u0_a2324096Oct312:43backups
drwx------3u0_a232u0_a2324096Oct110:36assignments
drwx------3u0_a232u0_a2324096Oct613:30a
drwx------4u0_a232u0_a2324096Oct110:31letters
drwx------5u0_a232u0_a2324096Oct110:35reports
total248
$gccexp4.c
$./a.out
-rw-------1u0_a232u0_a232 14Sep309:27file5
-rw-------1u0_a232u0_a232 17Oct118:58abc
-rw-------1u0_a232u0_a232 36Dec2819:44file2
-rw-------1u0_a232u0_a232 98Aug2412:56temp2
-rw-------1u0_a232u0_a232 98Aug2710:51temp4
-rw-------1u0_a232u0_a232 157Dec1320:57temp
-rw-------1u0_a232u0_a232 200Jan410:43exp4.c
```

-rw-------1u0_a232u0_a232 240Sep811:33fs3
-rw-------1u0_a232u0_a232 279Aug2506:55temp3
-rw-------1u0_a232u0_a232 475Dec2217:26exp2.c
-rw-------1u0_a232u0_a232 475Dec2217:27filecopy.c
-rw-------1u0_a232u0_a232 570Jan420:04shmc.c
-rw-------1u0_a232u0_a232 845Jan420:03shms.c
-rw-------1u0_a232u0_a2321386Dec2216:29exp3.c
-rw-------1u0_a232u0_a2321386Dec2219:38experiment3.c
-rw-------1u0_a232u0_a2321661Jan420:42temp5
-rw-------1u0_a232u0_a23212288Aug2619:26typescript
-rwx------1u0_a232u0_a232 475Dec2217:41dest.txt
-rwx------1u0_a232u0_a2328280Jan518:06a.out
-rwx--x--x1u0_a232u0_a232 0Sep2109:24fxile
-rwx--x--x1u0_a232u0_a232 0Sep2109:29fxile2
-rwxr-xr-x1u0_a232u0_a232 47Sep1508:50f1
-rwxrw-r-x1u0_a232u0_a232 59Aug2409:37file4
-rwxrw-r-x1u0_a232u0_a232 138Oct123:11file1
drwx------2u0_a232u0_a2324096Dec2913:43downloads
drwx------2u0_a232u0_a2324096Oct312:43backups
drwx------3u0_a232u0_a2324096Oct110:36assignments
drwx------3u0_a232u0_a2324096Oct613:30a
drwx------4u0_a232u0_a2324096Oct110:31letters
drwx------5u0_a232u0_a2324096Oct110:35reports
total248

OUTPUTSCREENSHOT:

```
$ ls -l|sort
-rw------- 1 u0_a232 u0_a232    14 Sep  3 09:27 file5
-rw------- 1 u0_a232 u0_a232    17 Oct  1 18:58 abc
-rw------- 1 u0_a232 u0_a232    36 Dec 28 19:44 file2
-rw------- 1 u0_a232 u0_a232    98 Aug 24 12:56 temp2
-rw------- 1 u0_a232 u0_a232    98 Aug 27 10:51 temp4
-rw------- 1 u0_a232 u0_a232   157 Dec 13 20:57 temp
-rw------- 1 u0_a232 u0_a232   200 Jan  4 10:43 exp4.c
-rw------- 1 u0_a232 u0_a232   240 Sep  8 11:33 fs3
-rw------- 1 u0_a232 u0_a232   279 Aug 25 06:55 temp3
-rw------- 1 u0_a232 u0_a232   475 Dec 22 17:26 exp2.c
-rw------- 1 u0_a232 u0_a232   475 Dec 22 17:27 filecopy.c
-rw------- 1 u0_a232 u0_a232   570 Jan  4 20:04 shmc.c
-rw------- 1 u0_a232 u0_a232   845 Jan  4 20:03 shms.c
-rw------- 1 u0_a232 u0_a232  1386 Dec 22 16:29 exp3.c
-rw------- 1 u0_a232 u0_a232  1386 Dec 22 19:38 experiment3.c
-rw------- 1 u0_a232 u0_a232  1661 Jan  4 20:42 temp5
-rw------- 1 u0_a232 u0_a232 12288 Aug 26 19:26 typescript
-rwx------ 1 u0_a232 u0_a232   475 Dec 22 17:41 dest.txt
-rwx------ 1 u0_a232 u0_a232  8280 Jan  4 19:35 a.out
-rwx--x--x 1 u0_a232 u0_a232     0 Sep 21 09:24 fxile
-rwx--x--x 1 u0_a232 u0_a232     0 Sep 21 09:29 fxile2
-rwxr-xr-x 1 u0_a232 u0_a232    47 Sep 15 08:50 f1
-rwxrw-r-x 1 u0_a232 u0_a232    59 Aug 24 09:37 file4
-rwxrw-r-x 1 u0_a232 u0_a232   138 Oct  1 23:11 file1
drwx------ 2 u0_a232 u0_a232  4096 Dec 29 13:43 downloads
drwx------ 2 u0_a232 u0_a232  4096 Oct  3 12:43 backups
drwx------ 3 u0_a232 u0_a232  4096 Oct  1 10:36 assignments
drwx------ 3 u0_a232 u0_a232  4096 Oct  6 13:30 a
drwx------ 4 u0_a232 u0_a232  4096 Oct  1 10:31 letters
drwx------ 5 u0_a232 u0_a232  4096 Oct  1 10:35 reports
total 248
$ gcc exp4.c
$ ./a.out
-rw------- 1 u0_a232 u0_a232    14 Sep  3 09:27 file5
-rw------- 1 u0_a232 u0_a232    17 Oct  1 18:58 abc
-rw------- 1 u0_a232 u0_a232    36 Dec 28 19:44 file2
-rw------- 1 u0_a232 u0_a232    98 Aug 24 12:56 temp2
-rw------- 1 u0_a232 u0_a232    98 Aug 27 10:51 temp4
-rw------- 1 u0_a232 u0_a232   157 Dec 13 20:57 temp
-rw------- 1 u0_a232 u0_a232   200 Jan  4 10:43 exp4.c
-rw------- 1 u0_a232 u0_a232   240 Sep  8 11:33 fs3
-rw------- 1 u0_a232 u0_a232   279 Aug 25 06:55 temp3
-rw------- 1 u0_a232 u0_a232   475 Dec 22 17:26 exp2.c
-rw------- 1 u0_a232 u0_a232   475 Dec 22 17:27 filecopy.c
-rw------- 1 u0_a232 u0_a232   570 Jan  4 20:04 shmc.c
-rw------- 1 u0_a232 u0_a232   845 Jan  4 20:03 shms.c
-rw------- 1 u0_a232 u0_a232  1386 Dec 22 16:29 exp3.c
-rw------- 1 u0_a232 u0_a232  1386 Dec 22 19:38 experiment3.c
-rw------- 1 u0_a232 u0_a232  1661 Jan  4 20:42 temp5
-rw------- 1 u0_a232 u0_a232 12288 Aug 26 19:26 typescript
-rwx------ 1 u0_a232 u0_a232   475 Dec 22 17:41 dest.txt
-rwx------ 1 u0_a232 u0_a232  8280 Jan  5 18:06 a.out
-rwx--x--x 1 u0_a232 u0_a232     0 Sep 21 09:24 fxile
-rwx--x--x 1 u0_a232 u0_a232     0 Sep 21 09:29 fxile2
-rwxr-xr-x 1 u0_a232 u0_a232    47 Sep 15 08:50 f1
-rwxrw-r-x 1 u0_a232 u0_a232    59 Aug 24 09:37 file4
-rwxrw-r-x 1 u0_a232 u0_a232   138 Oct  1 23:11 file1
drwx------ 2 u0_a232 u0_a232  4096 Dec 29 13:43 downloads
drwx------ 2 u0_a232 u0_a232  4096 Oct  3 12:43 backups
drwx------ 3 u0_a232 u0_a232  4096 Oct  1 10:36 assignments
drwx------ 3 u0_a232 u0_a232  4096 Oct  6 13:30 a
drwx------ 4 u0_a232 u0_a232  4096 Oct  1 10:31 letters
drwx------ 5 u0_a232 u0_a232  4096 Oct  1 10:35 reports
total 248
$
```

EXPERIMENT NO:5A

AIM : To write a C program that illustrates two processes communicating using shared memory.

DESCRIPTION : Shared memory is a memory shared between two or more processes. To reiterate, each process has its own address space, if any process wants to communicate with some information from its own address space to other processes, then it is only possible with IPC (inter process communication) techniques. As we are already aware, communication can be between related or unrelated processes.

Usually, inter-related process communication is performed using Pipes or Named Pipes. Unrelated processes (say one process running in one terminal and another process in another terminal) communication can be performed using Named Pipes or through popular IPC techniques of Shared Memory and Message Queues.

SYNTAX :

ftok() –The *ftok*() function shall return a key based on *path* and *id* that is usable in subsequent calls to *msgget*(), *semget*(), and *shmget*(). The application shall ensure that the *path* argument is the pathname of an existing file that the process is able to *stat*().

shmdt() –shmdt() detaches the shared memory segment located at the address specified by shmaddrfrom the address space of the calling process.

shmctl() -Theshmctl() function provides a variety of shared memory control operations as specified by *cmd*. The following commands are available:

LIBRARIES USED:

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

These contains server client

operations.

PROGRAM:

//shared memory

server

#include<stdlib.h>

```
#include<sys/ipc.h>

#include<sys/shm.h>

#include<stdio.h>

#include<unistd.h>

#define NOT_READY -1

#define FILLED 0

#define TAKEN 1

struct Memory

{

int status;

int data[4];

};

void main(int argc,char *argv[])

{

key_t ShmKEY;

int ShmID,i;

struct Memory *ShmPTR;

//prepare for a shared memory

ShmKEY=ftok("./",'x');

ShmID=shmget(ShmKEY,sizeof(struct

Memory),IPC_CREAT|0666); //Returns the starting address of

shared memory
```

```
ShmPTR=(struct Memory *)shmat(ShmID,NULL,0);

//shared memory not ready

ShmPTR->status=NOT_READY;
//filling the data

for(i=0;i<4;i++) ShmPTR-

>data[i]=atoi(argv[i+1]); //wait

until the data is taken ShmPTR-

>status=FILLED; while(ShmPTR-

>status!=TAKEN){ sleep(1); /sleep

for 1second/

}

//detach and remove shared

memory shmdt((void *) ShmPTR);

shmctl(ShmID,IPC_RMID,NULL);

exit(0);

}


//shared memory client

#include<stdlib.h>

#include<sys/ipc.h>

#include<sys/shm.h>

#include<stdio.h>

#define NOT_READY -1

#define FILLED 0#define
```

```
TAKEN 1

struct Memory

{

int status;
int data[4];

};

void main(void)

{

key_t ShmKEY;

int ShmID;

struct Memory *ShmPTR;

//prepare for a shared memory

ShmKEY=ftok("./",'x');

ShmID=shmget(ShmKEY,sizeof(struct

Memory),0666); ShmPTR=(struct Memory

*)shmat(ShmID,NULL,0); while(ShmPTR-

>status!=FILLED)

;

printf("%d %d %d %d\n",ShmPTR->data[0],ShmPTR->data[1],ShmPTR-
>data[2],ShmPTR- >data[3]);

ShmPTR->status=TAKEN;

shmdt((void *) ShmPTR);

exit(0);

}
```

OUTPUT:

ubuntu@ip-172-31-31-109:~$ gcc ipcserver.c

ubuntu@ip-172-31-31-109:~$ ./a.out

4 6 8 2

ubuntu@ip-172-31-31-109:~$

ubuntu@ip-172-31-31-109:~$ gcc ipcclient.c

ubuntu@ip-172-31-31-109:~$ ./a.out 4 6 8 2

ubuntu@ip-172-31-31-109:~$

OUTPUT SCREENSHOTS:

EXPERIMENT-5B

AIM: i) To write a C program to create a one way pipe using single process.

DESCRIPTION: write() writes up to count bytes to the file referenced by the file descriptor fd from the buffer starting at buf. POSIX requires that a read() which can be proved to occur after a write() has returned returns the new data. Note that not all file systems are POSIX conforming.

LIBRARIES USED:

```
#include<stdio.h>

#include<stdlib.h>

#include<errno.h>

#include<unistd.h>
```

These are the libraries used to work with pipe using single process.

SYNTAX:

```
int main(){}
ssize_t  write(int fd, const void *buf, size_t count);
```

PROGRAM:

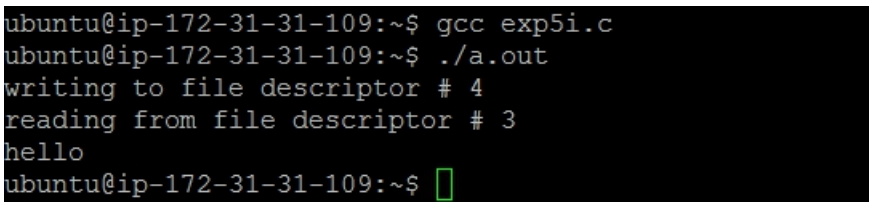
```
#include<stdio.h>

#include<stdlib.h>

#include<errno.h>

#include<unistd.h>

int main(void)

{

int pfds[2];

char buf[30];

if(pipe(pfds)==-1)

{

perror("pipe");
```

```
exit(1);

}

printf("writing to file descriptor # %d\n", pfds[1]);

write(pfds[1],"hello",6);

printf("reading from file descriptor # %d\n",pfds[0]);

read(pfds[0],buf,6);

printf("%s \n",buf);

return 0;

}
```

OUTPUT:

ubuntu@ip-172-31-31-109:~**$** gcc exp5i.c
ubuntu@ip-172-31-31-109:~**$** ./a.out
Writing to file descriptor # 4
Reading from file descripto # 3
hello

OUTPUT SCREENSHOTS:



AIM: ii) Write a C program to create child & parent process and making them communicate by using pipe().

DESCRIPTION:

pipe() creates a pair of file descriptors, pointing to a pipe inode, and places them in the array pointed to by filedes. filedes[0] is for reading, filedes[1] is for writing.

LIBRARIES USED :

```
#include<stdio.h>

#include<stdlib.h>

#include<errno.h>

#include<sys/types.h>

#include<unistd.h>
```

These are the libraries used to work with pipe() using child and parent process.

SYNTAX:

```
int pipe(int filedes[2]);
```

PROGRAM:

```
#include<stdio.h>

#include<stdlib.h>

#include<errno.h>

#include<sys/types.h>

#include<unistd.h>

int main(void)

{

int pfds[2];

char buf[30];

pipe(pfds);
```

```
if(!fork())

{

printf("Child: Writing to the pipe\n");

write(pfds[1],"test",5);

printf("Child:exiting\n");

exit(0);

}

else

{

printf("Parent: reading from pipe\n");

read(pfds[0],buf,5);

printf("parent reads: %s \n",buf);

wait(NULL);

}

return 0;

}
```

OUTPUT:

```
ubuntu@ip-172-31-31-109:~$ gcc exp5ii.c
exp5ii.c: In function 'main':
exp5ii.c:23:1: warning: implicit declaration of function 'wait'; did you mean 'main'? [-
Wimplicit-function-declaration]
 wait(NULL);
 ^~~~
 main
ubuntu@ip-172-31-31-109:~$ ./a.out
Parent: reading from pipe
Child: Writing to the pipe
parent reads: test
Child:exiting
```

OUTPUT SCREENSHOTS:

```
ubuntu@ip-172-31-31-109:~$ gcc exp5ii.c
exp5ii.c: In function 'main':
exp5ii.c:23:1: warning: implicit declaration of function 'wait'; did you mean 'm
ain'? [-Wimplicit-function-declaration]
 wait(NULL);
 ^~~~
 main
ubuntu@ip-172-31-31-109:~$ ./a.out
Parent: reading from pipe
Child: Writing to the pipe
parent reads: test
Child:exiting
ubuntu@ip-172-31-31-109:~$
```

AIM: iii) Write a C program to create a two-way pipes between two process

DESCRIPTION: read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf.

If count is zero, read() returns zero and has no other results. If count is greater than SSIZE_MAX, the result is unspecified.

fork() creates a child process that differs from the parent process only in its PID and PPID, and in the fact that resource utilizations are set to 0. File locks and pending signals are not inherited.

LIBRARIES USED:

#include<stdio.h>

#include<stdlib.h>

These are the libraries used to work with pipe() using two process.

SYNTAX:
ssize_t read(int fd, void *buf, size_t count);
PROGRAM:

```
#include<stdio.h>

#include<stdlib.h>

main()

{

int p1[2],p2[2],n,pid;

char buf1[25],buf2[25];

pipe(p1); pipe(p2);

printf("\n readfds=%d %d\n",p1[0],p2[0]);

printf("\n writefds=%d %d\n",p1[1],p2[1]);

pid=fork();
```

```
if(pid==0)
{
close(p1[0]);
printf("\n Child process sending data \n");
write(p1[1]," INDIA",6);
close(p2[1]);
read(p2[0],buf1,25);
printf("reply from parent:%s\n",buf1);
sleep(2);
}
else
{
close(p1[1]);
printf("\n parent process receiving data\n");
n=read(p1[0],buf2,sizeof(buf2));
printf("\n data received from child through pipe:%s\n",buf2);
sleep(3);
close(p2[0]);
write(p2[1],"EARTH",6);
printf("\n reply send\n");
}
}
```

OUTPUT:

ubuntu@ip-172-31-31-109:~$ ./a.out

readfds=3 5

writefds=4 6

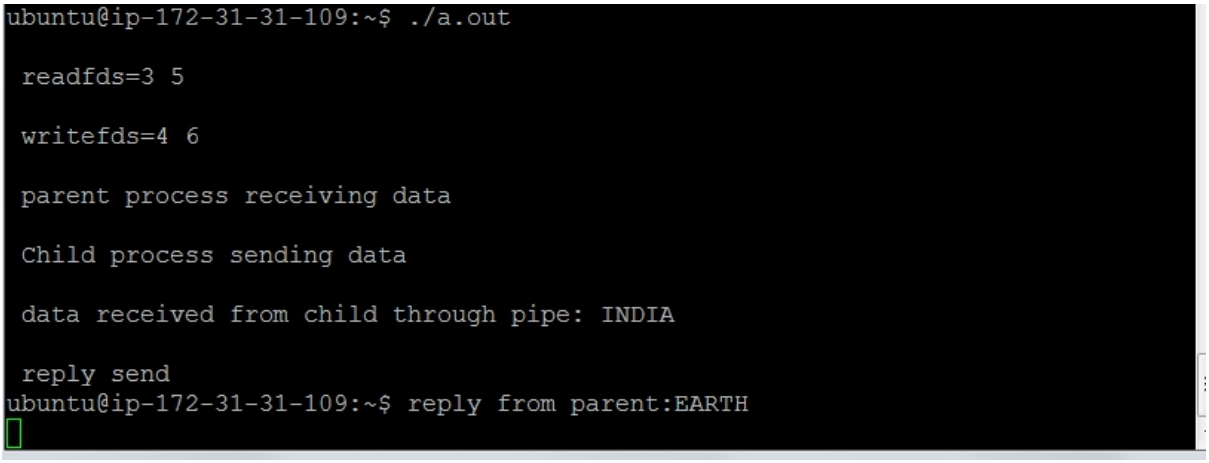
parent process receiving data

Child process sending data

data received from child through pipe: INDIA

reply send
ubuntu@ip-172-31-31-109:~$ reply from parent:EARTH

OUTPUT SCREEENSHOTS:



EXPERIMENT-5C

AIM: Write a C program to perform inter process communication using fifo's.
DESCRIPTION:

mkfifo() makes a FIFO special file with name pathname. Mode specifies the FIFO's permissions. It is modified by the process's umask in the usual way: the permissions of the created file are (mode & ~umask).

A FIFO special file is similar to a pipe, except that it is created in a different way. Instead of being an anonymous communications channel, a FIFO special file is entered into the file system by calling mkfifo().Once you have created a FIFO special file in this way, any process can open it for reading or writing, in the same way as an ordinary file. However, it has to be open at both ends simultaneously before you can proceed to do any input or output operations on it. Opening a FIFO for reading normally blocks until some other process opens the same FIFO for writing, and vice versa.

LIBRARIES USED:

```
#include<stdio.h>

#include<unistd.h>

#include<fcntl.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<string.h>
```

These are the libraries used to communicate through fifo's
SYNTAX:
mkfifo - make a FIFO special file
int close(int fd);
PROGRAM:

SERVER:

```
#include<stdio.h>

#include<unistd.h>
```

```
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<string.h>
int main()
{
char fname[25]="";
char fcontent[100]="";
int fd,fd1,fd2;
mkfifo("fifo1",0600);
mkfifo("fifo2",0600);
fd=open("fifo1",O_RDONLY);
fd1=open("fifo2",O_WRONLY);
read(fd,fname,25);
fd2=open(fname,O_RDONLY);
while(read(fd2,fcontent,100)!=0)
{
printf("%s\n",fcontent);
if(fd<0)
write(fd1,"file not exit",14);
else
write(fd1,fcontent,strlen(fcontent));
}
```

```
close(fd);

close(fd1);

close(fd2);

}
```

CLIENT:

```
#include<stdio.h>

#include<unistd.h>

#include<fcntl.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<string.h>

int main()

{

char s[100]="";

char s1[1000]="";

int fd,fd1;

fd=open("fifo1",O_WRONLY);

fd1=open("fifo2",O_RDONLY);

printf("\nEnter the file name:");

scanf("%s",s);

write(fd,s,strlen(s));

while(read(fd1,s1,1000)!=0)

{
```

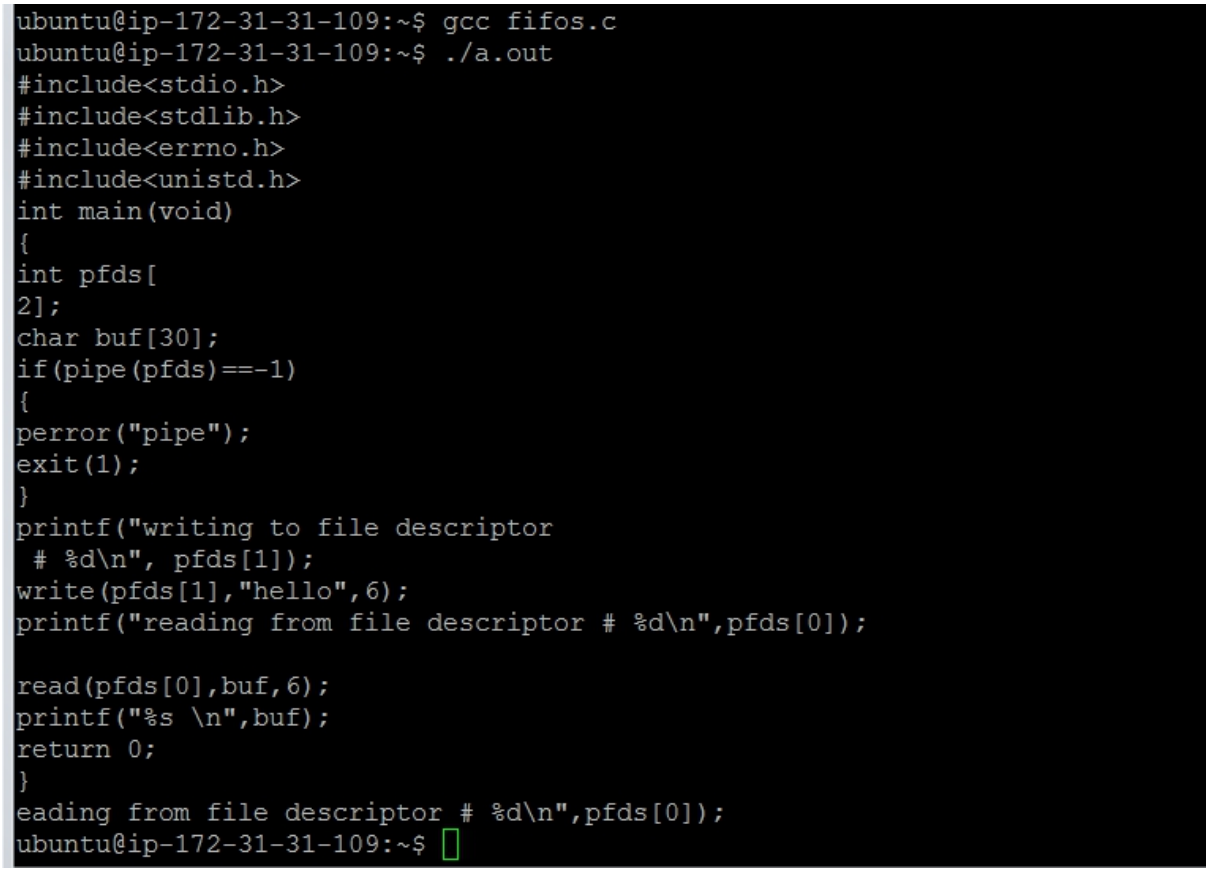
```
printf("File Content :%s",s1);

}

}
```

OUTPUT:

```
ubuntu@ip-172-31-31-109:~$ gcc fifos.c
ubuntu@ip-172-31-31-109:~$ ./a.out
#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<unistd.h>
int main(void)
{
int pfds[
2];
char buf[30];
if(pipe(pfds)==-1)
{
perror("pipe");
exit(1);
}
printf("writing to file descriptor
 # %d\n", pfds[1]);
write(pfds[1],"hello",6);
printf("reading from file descriptor # %d\n",pfds[0]);

read(pfds[0],buf,6);
printf("%s \n",buf);
return 0;
}
eading from file descriptor # %d\n",pfds[0]);
ubuntu@ip-172-31-31-109:~$

ubuntu@ip-172-31-31-109:~$ gcc fifoc.c
ubuntu@ip-172-31-31-109:~$ ./a.out

Enter the file name:exp5i.c
File Content :#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<unistd.h>
int main(void)
```

```
{
int pfds[2];
char buf[30];
if(pipe(pfds)==-1)
{
perror("pipe");
exit(1);
}
printf("writing to file descriptor # %d\n", pfds[1]);
write(pfds[1],"hello",6);
printf("reading from file descriptor # %d\n",pfds[0]);
read(pfds[0],buf,6);
printf("%s \n",buf);
return 0;
}
ubuntu@ip-172-31-31-109:~$
```

OUTPUT SCREENSHOTS:



```
ubuntu@ip-172-31-31-109:~$ gcc fifoc.c
ubuntu@ip-172-31-31-109:~$ ./a.out

Enter the file name:exp5i.c
File Content :#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<unistd.h>
int main(void)
{
int pfds[2];
char buf[30];
if(pipe(pfds)==-1)
{
perror("pipe");
exit(1);
}
printf("writing to file descriptor # %d\n", pfds[1]);
write(pfds[1],"hello",6);
printf("reading from file descriptor # %d\n",pfds[0]);
read(pfds[0],buf,6);
printf("%s \n",buf);
return 0;
}
ubuntu@ip-172-31-31-109:~$
```

EXPERIMENT-5D

AIM: Write a C program to perform inter process communication using message queues

DESCRIPTION: A message queue is a linked list of messages stored within the kernel and identified by a message queue identifier. A new queue is created or an existing queue opened by **msgget()**.

New messages are added to the end of a queue by **msgsnd()**. Every message has a positive long integer type field, a non-negative length, and the actual data bytes (corresponding to the length), all of which are specified to msgsnd() when the message is added to a queue. Messages are fetched from a queue by **msgrcv()**. We don't have to fetch the messages in a first-in, first-out order. Instead, we can fetch messages based on their type field.

LIBRARIES USED:

```
#include <string.h>

#include <sys/msg.h>

#include<stdio.h>
```

These are the libraries used to communicate using message queues.

SYNTAX:

Strcpy();
msgsnd();
msgrcv();

PROGRAM:

Sender:

```
#include <string.h>

#include <sys/msg.h>

int main() {

int msqid = 0;

struct message {

long type;

char text[20];

} msg;

msg.type = 1;

strcpy(msg.text, "This is message 1");
```

```
msgsnd(msqid, (void *) &msg, sizeof(msg.text), IPC_NOWAIT);

strcpy(msg.text, "This is message 2");

msgsnd(msqid, (void *) &msg, sizeof(msg.text), IPC_NOWAIT);

return 0;

}
```

Receiver:

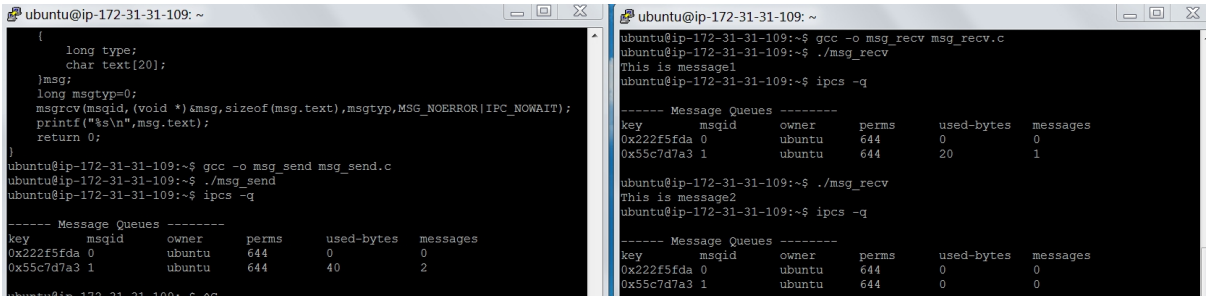
```
#include <stdio.h>

#include <sys/msg.h>

int main() {

int msqid = 65536;

struct message {

long type;

char text[20];

} msg;

long msgtyp = 0;

msgrcv(msqid, (void *) &msg, sizeof(msg.text), msgtyp, MSG_NOERROR | IPC_NOWAIT);

printf("%s \n", msg.text);

return 0;

}
```

OUTPUT:

```
ubuntu@ip-172-31-31-109:~$ gcc -o msg_send msg_send.c
ubuntu@ip-172-31-31-109:~$ ./msg_send
ubuntu@ip-172-31-31-109:~$ ipcs -q

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
```

```
0x222f5fda 0        ubuntu    644       0          0
0x55c7d7a3 1        ubuntu    644       40         2

ubuntu@ip-172-31-31-109:~$
ubuntu@ip-172-31-31-109:~$ gcc -o msg_recv msg_recv.c
ubuntu@ip-172-31-31-109:~$ ./msg_recv
This is message1
ubuntu@ip-172-31-31-109:~$ ipcs -q

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x222f5fda 0          ubuntu     644        0            0
0x55c7d7a3 1          ubuntu     644        20           1

ubuntu@ip-172-31-31-109:~$ ./msg_recv
This is message2
ubuntu@ip-172-31-31-109:~$ ipcs -q

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x222f5fda 0          ubuntu     644        0            0
0x55c7d7a3 1          ubuntu     644        0            0

ubuntu@ip-172-31-31-109:~$
```

OUTPUT SCREENSHOTS:



EXPERIMENT NO-6

AIM : Write a C program to stimulate producer and consumer problem using semaphores
DESCRIPTION :The producer consumer problem is a synchronization problem. We have a buffer of fixed size. A producer can produce an item and can place in the buffer. A consumer can pick items and can consume them. We need to ensure that when a producer is placing an item in the buffer, then at the same time consumer should not consume any item. In this problem, buffer is the critical section. **Semaphore** : A semaphore S is an integer variable that can be accessed only through two standard operations :

- wait() - The wait() operation reduces the value of semaphore by 1
- signal() - The signal() operation increases its value by 1.

```
 wait(S){
 while(S<=0); // busy waiting
S--;
 }
signal(S){
 S++;
}
```

Semaphores are of two types:

- **Binary Semaphore** – This is similar to mutex lock but not the same thing. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problem with multiple processes.
- **Counting Semaphore** – Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

Mutual Exclusion: One or more than one resource are non-sharable (Only one process can use at a time). A mutex provides mutual exclusion, either producer or consumer can have the key (mutex) and proceed with their work. As long as the buffer is filled by producer, the consumer needs to wait, and vice versa.

```
 Initialization of semaphores
 mutex = 1 Full = 0 // Initially, all slots are empty. Thus full slots are 0
 Empty = n // All slots are empty initially
```

Solution for Producer :

```
do{
 //produce an item
wait(empty);
 wait(mutex);
 //place in buffer
 signal(mutex);
 signal(full);
}while(true)
```

When producer produces an item then the value of "empty" is reduced by 1 because one slot will be filled now. The value of mutex is also reduced to prevent consumer to access the buffer. Now, the producer has placed the item and thus the value of "full" is increased by 1. The value of mutex is also increased by 1 beacuse the task of producer has been completed

and consumer can access the buffer. Solution for Consumer: do{ wait(mutex); //remove item from buffer Signal(mutex); Signal(empty); //consumes item }while(true) As the consumer is removing an item from buffer, therefore the value of "full" is reduced by 1 and the value is mutex is also reduced so that the producer cannot access the buffer at this moment. Now, the consumer has consumed the item, thus increasing the value of "empty" by 1. The value of mutex is also increased so that producer can access the buffer now.

LIBRARIES USED:

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

Program:

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
int main()
{
 int n;
 void producer();
 void consumer();
 int wait(int);
 int signal(int);
 printf("\n1.Producer\n2.Consumer\n3.Exit");
 while(1)
 {
 printf("\nEnter your choice:");
 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;
 }
}
 return 0;
}
```

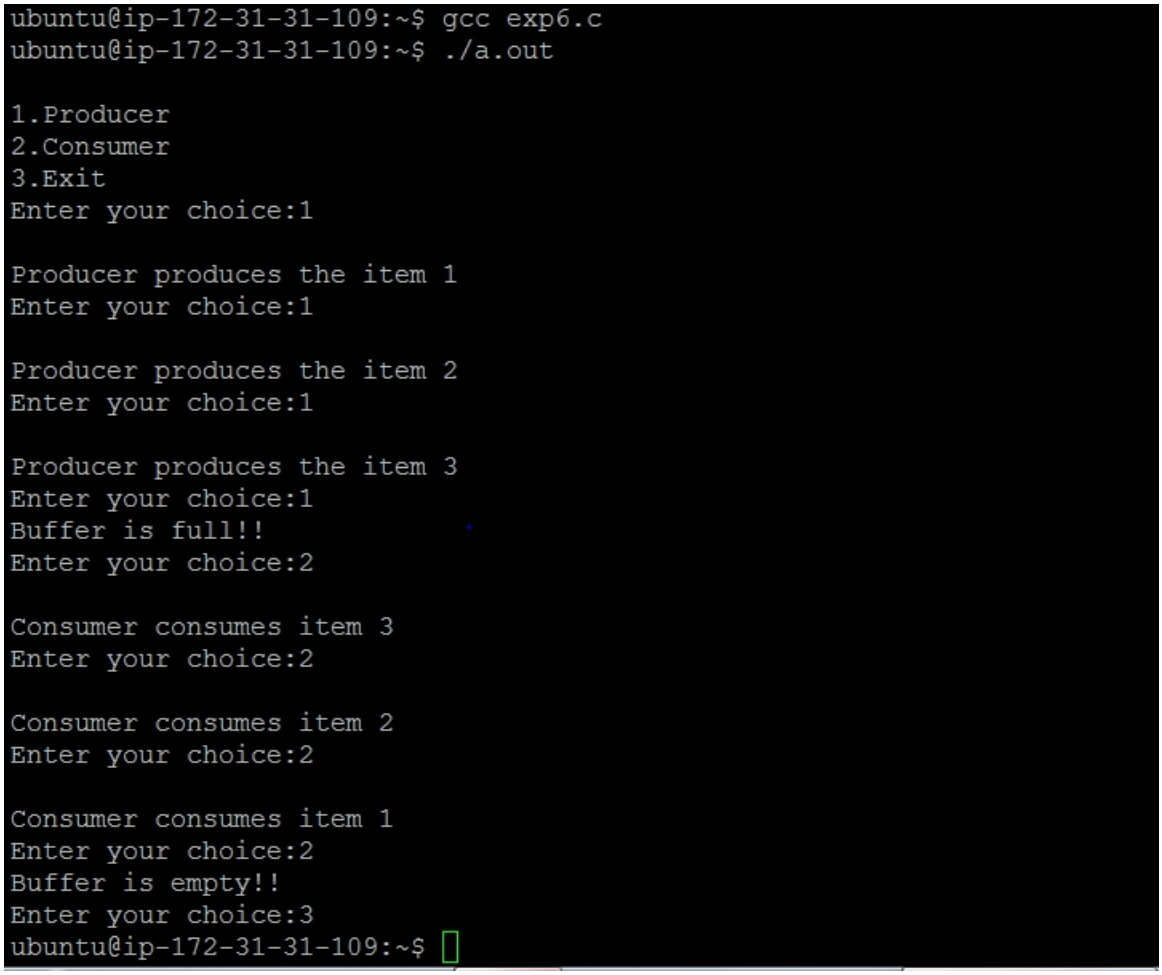
```
int wait(int s)
{
 return (--s);
}
int signal(int s)
{
 return(++s);
}
void producer()
{
 mutex=wait(mutex);
 full=signal(full);
 empty=wait(empty);
 x++;
 printf("\nProducer produces the item %d",x);
 mutex=signal(mutex);
}
void consumer()
{
 mutex=wait(mutex);
 full=wait(full);
 empty=signal(empty);
 printf("\nConsumer consumes item %d",x);
 x--;
 mutex=signal(mutex);
}
```

OUTPUT:

```
ubuntu@ip-172-31-31-109:~$gccexp6.c
ubuntu@ip-172-31-31-109:~$./a.out
1.Producer
2.Consumer
3.Exit
Enteryourchoice:1
Producerproducestheitem1
Enteryourchoice:1
Producerproducestheitem2
Enteryourchoice:1
Producerproducestheitem3
Enteryourchoice:1
Bufferisfull!!
Enteryourchoice:2
Consumerconsumesitem3
Enteryourchoice:2
```

Consumerconsumesitem2
Enteryourchoice:2
Consumerconsumesitem1
Enteryourchoice:2
Bufferisempty!!
Enteryourchoice:3
ubuntu@ip-172-31-31-109:~$

OUTPUTSCREENSHOT:



EXPERIMENT NO: 7

AIM : To write C program to create a thread using pthreads library and let it run its function.

DESCRIPTION :POSIX.1 specifies a set of interfaces (functions, header files for threaded programming commonly known as POSIX threads, or Pthreads. A single process can contain multiple threads, all of which are executing the same program. These threads share the same global memory (data and heap segments), but each thread has its own stack (automatic variables).

The pthread_create() function is used to create a new thread, with attributes specified by attr, within a process. If attr is NULL, the default attributes are used. If the attributes specified by attr are modified later, the thread's attributes are not affected. Upon successful completion, pthread_create() stores the ID of the created thread in the location referenced by thread.

The pthread_join() function suspends execution of the calling thread until the target thread terminates, unless the target thread has already terminated. On return from a successful pthread_join() call with a non-NULL value_ptr argument, the value passed to pthread_exit() by the terminating thread is made available in the location referenced by value_ptr. When a pthread_join() returns successfully, the target thread has been terminated. The results of multiple simultaneous calls to pthread_join() specifying the same target thread are undefined. If the thread calling pthread_join() is canceled, then the target thread will not be detached.

SYNTAX:

int pthread_create(pthread_t *_thread_, const pthread_attr_t *_attr_,
  void *(*_start_routine_)(void*), void *_arg_);
int pthread_join(pthread_t _thread_, void **_value_ptr_);
int main(){}

LIBRARIES USED:

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>
```

All these libraries are used to implement pthreads and to run it . pthread library is present in pthread.h

PROGRAM:

```
#include <stdio.h>

#include <stdlib.h>
```

```
#include <unistd.h>

#include <pthread.h>

void *myThreadFun(void *vargp)

{

sleep(1);

printf("Printing inside Thread \n");

return NULL;

}

int main()

{

pthread_t thread_id;

printf("Before Thread creation\n");

pthread_create(&thread_id, NULL, myThreadFun, NULL);

pthread_join(thread_id, NULL);

printf("After Thread creation\n");

exit(0);

}
```

OUTPUT:

```
ubuntu@ip-172-31-31-109:~/os$ gcc exp7.c -lpthread
ubuntu@ip-172-31-31-109:~/os$ ./a.out
Before Thread creation
Printing inside Thread
After Thread creation
ubuntu@ip-172-31-31-109:~/os$
```

OUTPUT SCREEN SHOTS:

```
ubuntu@ip-172-31-31-109:~/os$ gcc exp7.c -lpthread
ubuntu@ip-172-31-31-109:~/os$ ./a.out
Before Thread creation
Printing inside Thread
After Thread creation
```

EXPERIMENT NO: 8

AIM : To write a C program to illustrate concurrent execution of threads using pthreads library.

DESCRIPTION : POSIX Threads, usually referred to as **pthreads**, is an execution model that exists independently from a language, as well as a parallel execution model. Each thread will share the same static variable which is mostly likely a global variable. The scenario where some threads can have wrong value is the race condition (increment isn't done in one single execution rather it is done in 3 assembly instructions, load, increment, store)

The pthread_create() function is used to create a new thread, with attributes specified by attr, within a process. If attr is NULL, the default attributes are used. If the attributes specified by attr are modified later, the thread's attributes are not affected. Upon successful completion, pthread_create() stores the ID of the created thread in the location referenced by thread.

The *pthread_exit()* function terminates the calling thread and makes the value *value_ptr* available to any successful join with the terminating thread. Any cancellation cleanup handlers that have been pushed and not yet popped are popped in the reverse order that they were pushed and then executed. After all cancellation cleanup handlers have been executed, if the thread has any thread-specific data, appropriate destructor functions will be called in an unspecified order. Thread termination does not release any application visible process resources, including, but not limited to, mutexes and file descriptors, nor does it perform any process level cleanup actions, including, but not limited to, calling any *atexit()* routines that may exist.

SYNTAX:

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
   void *(*start_routine)(void*), void *arg);
void pthread_exit(void *value_ptr);

int main(){}
```

LIBRARY USED:

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>
```

All these libraries are used to implement pthreads and to run it . pthread library is present in pthread.h

PROGRAM:

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

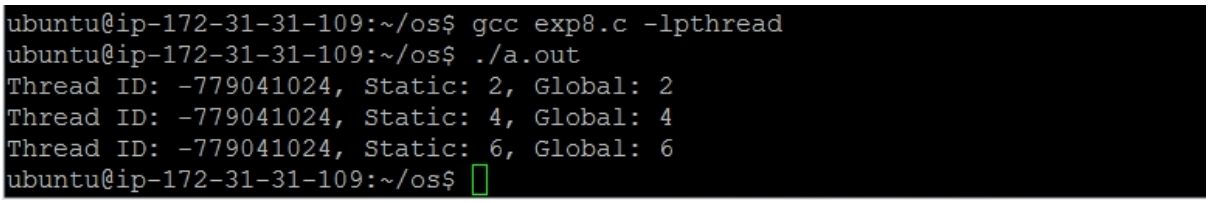
```
#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

// Let us create a global variable to change it in threads

int g = 0;

// The function to be executed by all threads

void *myThreadFun(void *vargp)

{

// Store the value argument passed to this thread

int *myid = (int *)vargp;

// Let us create a static variable to observe its changes

static int s = 0;

// Change static and global variables

++s; ++g;

// Print the argument, static and global variables

printf("Thread ID: %d, Static: %d, Global: %d\n", *myid, ++s, ++g);

}

int main()

{

int i;

pthread_t tid;

// Let us create three threads

for (i = 0; i < 3; i++)
```

```
pthread_create(&tid, NULL, myThreadFun, (void *)&tid);

pthread_exit(NULL);

return 0;

}
```

OUTPUT:

```
ubuntu@ip-172-31-31-109:~/os$ gcc exp8.c -lpthread
ubuntu@ip-172-31-31-109:~/os$ ./a.out
Thread ID: -779041024, Static: 2, Global: 2
Thread ID: -779041024, Static: 4, Global: 4
Thread ID: -779041024, Static: 6, Global: 6
ubuntu@ip-172-31-31-109:~/os$
```

OUTPUT SCREENSHOTS:



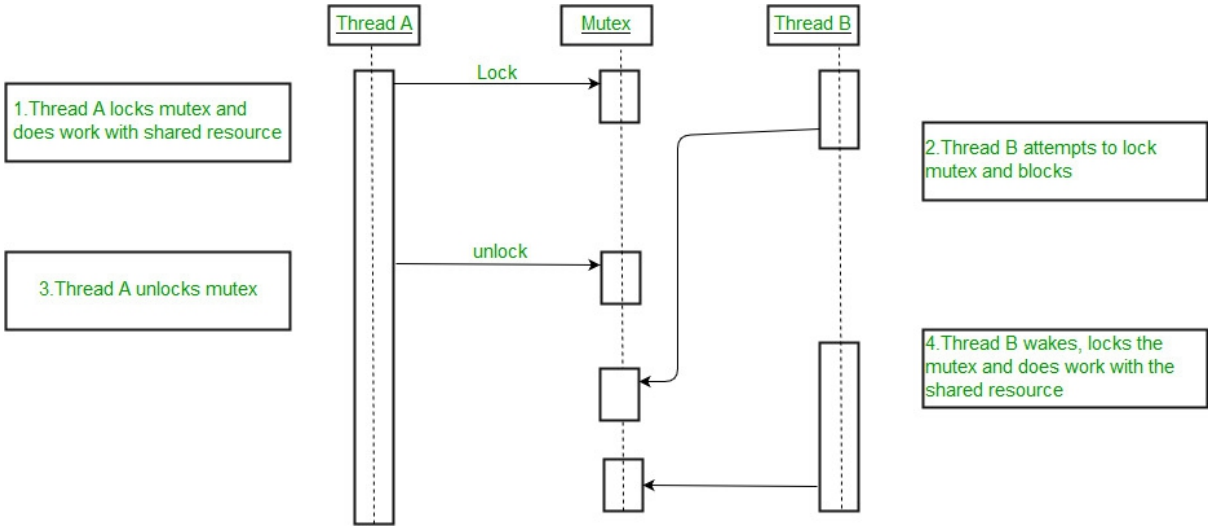
EXPERIMENT NO: 9

AIM: To write a C program to illustrate Mutex lock for Linux Thread Synchronization

DESCRIPTION: Thread synchronization is defined as a mechanism which ensures that two or more concurrent processes or threads do not simultaneously execute some particular program segment known as a critical section. Processes' access to critical section is controlled by using synchronization techniques. When one thread starts executing the critical section (a serialized segment of the program) the other thread should wait until the first thread finishes. If proper synchronization techniques are not applied, it may cause a race condition where the values of variables may be unpredictable and vary depending on the timings of context switches of the processes or threads.

The mutex object referenced by *mutex* is locked by calling *pthread_mutex_lock()*. If the mutex is already locked, the calling thread blocks until the mutex becomes available. This operation returns with the mutex object referenced by *mutex* in the locked state with the calling thread as its owner.

If the mutex type is PTHREAD_MUTEX_NORMAL, deadlock detection is not provided. Attempting to relock the mutex causes deadlock. If a thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, undefined behaviour results.



SYNTAX:

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_mutex_init(pthread_mutex_t *mutex,
   const pthread_mutexattr_t *attr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_join(pthread_t thread, void **value_ptr);
int main(){}
```

LIBRARIES USED:

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

```
#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#include <string.h>
```

All these libraries are used to implement pthreads and to run it . pthread library is present in pthread.h

PROGRAM:

```
#include<stdio.h>

#include<string.h>

#include<pthread.h>

#include<stdlib.h>

#include<unistd.h>

pthread_t tid[2];

int counter;

pthread_mutex_t lock;

void* trythis(void *arg)

{

pthread_mutex_lock(&lock);

unsigned long i = 0;

counter += 1;

printf("\n Job %d has started\n", counter);

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

printf("\n Job %d has finished\n", counter);

pthread_mutex_unlock(&lock);
```

```
return NULL;
}
int main(void)
{
int i = 0;
int error;
if (pthread_mutex_init(&lock, NULL) != 0)
{
printf("\n mutex init has failed\n");
return 1;
}
while(i < 2)
{
error = pthread_create(&(tid[i]), NULL, &trythis, NULL);
if (error != 0)
printf("\nThread can't be created :[%s]", strerror(error));
i++;
}
pthread_join(tid[0], NULL);
pthread_join(tid[1], NULL);
pthread_mutex_destroy(&lock);
return 0;
}
```

OUTPUT:

```
ubuntu@ip-172-31-31-109:~/os$ gcc exp9.c -lpthread
ubuntu@ip-172-31-31-109:~/os$ ./a.out

 Job 1 has started

 Job 1 has finished

 Job 2 has started

 Job 2 has finished
ubuntu@ip-172-31-31-109:~/os$
```

OUTPUT SCREENSHOTS: