

**1.Aim: Shell programming**

**Description:** Linux is a Unix-Like operating system. All the Linux/Unix commands are run in the terminal provided by the Linux system. Linux/Unix commands are case-sensitive. The terminal can be used to accomplish all administrative tasks. This includes package installation, file manipulation, and user management. Linux terminal is user-interactive. The terminal outputs the results of commands which are specified by the user itself. Execution of typed command is done only after you press the Enter key.

**1.Script program to print factorial of a given number**

```
read -p "Enter a Number " n
```

```
fact=1
```

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

```
fact=$((fact*i))
```

```
echo $fact
```

**Output:**

```
Enter a Number 5
```

```
120
```

**2.Script program to print fibonacci series**

```
read -p "Enter a number " n
```

```
a1=0
```

```
a2=1
```

```
echo "Fibonacci series: "
```

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

```
    echo -n "$a1 "
```

```
    temp=$((a1+a2))
```

```
    a1=$a2
```

```
    a2=$temp
```

```
}
```

**Output:**

```
Enter a number 5
```

```
Fibonacci series:
```

```
0 1 1 2 3
```

**3.Script program to add any five numbers taken as command line arguments**

```
read -p "Enter number1: " n1
read -p "Enter number2: " n2
read -p "Enter number3: " n3
read -p "Enter number4: " n4
read -p "Enter number5: " n5
echo "Sum is: "
echo $((n1+n2+n3+n4+n5))
```

**Output:**

```
Enter number1: 1
Enter number2: 2
Enter number3: 3
Enter number4: 4
Enter number5: 5
Sum is: 15
```

**4.Demonstration of 1.While loop**

```
read -p "Enter number" n
i=0
echo "First n digits are: "
while [ $i -lt $n ];
do
    echo $i
    i=$((i+1))
done
```

**Output:**

```
Enter number 5
First n digits are:
0 1 2 3 4
```

**2.Until loop**

```
read -p "Enter number: " n
```

```
c=0
echo "First n even numbers"
until [ $n == 0 ]
do
    echo $c
    c=$((c+2))
    n=$((n-1))
done
```

**Output:**

Enter number: 5

First n even numbers

0 2 4 6 8

**5.Shell program to check whether a number even or odd**

```
read -p "Enter number: " n
if [ $((n%2)) -eq 0 ];
then
    echo "The given number is even";
else
    echo "The given number is odd";
fi
```

**Output:**

Enter number: 5

The given number is odd

**6.Execution of file handling operations**

```
file="factorial.sh"
if [ -r $file ]
then
    echo "File has read access"
else
```

```
    echo "File does not have read access"
fi
if [ -w $file ]
then
    echo "File has write permission"
else
    echo "File does not have write permission"
fi
if [ -x $file ]
then
    echo "File has execute permission"
else
    echo "File does not have execute permission"
fi
if [ -f $file ]
then
    echo "File is an ordinary file"
else
    echo "This is a special file"
fi
if [ -d $file ]
then
    echo "File is a directory"
else
    echo "File is not a directory"
fi
```

```
if [ -e $file ]
then
    echo "File exists"
else
    echo "File does not exist"
fi
if [ -s $file ]
then
    echo "File size is not zero"
else
    echo "File size is zero"
fi
```

**Output:**

File has read access  
File has write permission  
File has execute permission  
File is an ordinary file  
File is not a directory  
File exists  
File size is not zero

**7.Shell program to check whether a number is palindrome or not**

```
echo "Enter Number: "
read n
num=$n
rev=0
while [ $num -gt 0 ]
do
    rev=$((rev*10 + num%10))
    num=$((num/10))
done
```

```
done
```

```
if [ $rev -eq $n ]
```

```
then
```

```
    echo "$n is a Palindrome"
```

```
else
```

```
    echo "$n is not a Palindrome"
```

```
fi
```

**Output:**

Enter Number:454

454 is a Palindrome

**8.Shell program to check whether a number is Armstrong number or not.**

```
echo -n "Enter a number: "
```

```
read n
```

```
t=$n
```

```
sum=0
```

```
while [ $n -gt 0 ]
```

```
do
```

```
    rem=$((n%10))
```

```
    cube=$((rem*rem*rem))
```

```
    sum=$((sum+cube))
```

```
    n=$((n/10))
```

```
done
```

```
if [ $sum -eq $t ]
```

```
then
```

```
    echo "$t is an Armstrong Number"
```

```
else
```

```
    echo "$t is not an Armstrong Number"
```

```
fi
```

**Output:**

Enter a number: 153

153 is an Armstrong Number

**9.Shell program to convert a number from decimal to binary**

echo "Enter any decimal no:"

read num

rem=1

bno=" "

while [ \$num -gt 0 ]

do

rem=`expr \$num % 2`

bno=\$bno\$rem

num=`expr \$num / 2`

done

i=\${#bno}

final=" "

while [ \$i -gt 0 ]

do

rev=`echo \$bno | awk '{ printf substr( \$0,\$i,1 ) }`

final=\$final\$rev

i=\$(( \$i - 1 ))

done

echo "Equivalent Binary no:" \$final

**Output:**

Enter any decimal no:10

Equivalent Binary no: 1010

**10. Script program to convert a number from decimal to base 5.**

```
echo -n "Enter any decimal no: "  
read num  
rem=1  
bno=""  
while [ $num -gt 0 ]  
do  
    rem=$((num%5))  
    bno=$rem$bno  
    num=$((num/5))  
done  
echo "The converted base 5 number: $bno"
```

**Output:**

```
Enter any decimal no: 10  
The converted base 5 number: 20
```

**11. Script program to convert a number from base 5 to decimal.**

```
echo -n "Enter Base 5 number: "  
read bno  
dec=0  
pow=1  
while [ $bno -gt 0 ]  
do  
    val=$((bno%10))  
    dec=$((dec+val*pow))  
    bno=$((bno/10))  
    pow=$((pow*5))  
done  
echo "The Decimal number is: $dec"
```



**Output:**

Enter Base 5 number: 20

The Decimal number is: 10

**4.Aim:** Demonstration of Linux/Unix file related system calls.**Description:**

**1.pwd command:** The pwd command is used to display the location of the current working directory.

**Syntax:** pwd

```
cbit@cbit-VirtualBox:~/Desktop/vidya$ pwd
/home/cbit/Desktop/vidya
```

**2.mkdir command:** The mkdir command is used to create a new directory under any directory.

**Syntax:** mkdir <directory name>

```
cbit@cbit-VirtualBox:~/Desktop$ mkdir vidya
cbit@cbit-VirtualBox:~/Desktop$ cd vidya
cbit@cbit-VirtualBox:~/Desktop/vidya$ pwd
/home/cbit/Desktop/vidya
```

**3.rmdir command:** The rmdir command is used to delete a directory.

**Syntax:** rmdir<directoryname>

```
cbit@cbit-VirtualBox:~/Desktop/vidya$ ls
abc Cars Cities fruits Names number State student
cbit@cbit-VirtualBox:~/Desktop/vidya$ rmdir abc
cbit@cbit-VirtualBox:~/Desktop/vidya$ ls
Cars Cities fruits Names number State student
cbit@cbit-VirtualBox:~/Desktop/vidya$
```

**4.ls command:** The ls command is used to display a list of content of a directory.

**Syntax:** ls

```
cbit@cbit-VirtualBox:~/Desktop/vidya$ ls
fruits Names
```

**5.cd command:** The cd command is used to change the current directory.

**Syntax:** cd <directory name>

```
cbit@cbit-VirtualBox:~/Desktop/vidya$ pwd
/home/cbit/Desktop/vidya
cbit@cbit-VirtualBox:~/Desktop/vidya$ cd /home/cbit/Desktop
cbit@cbit-VirtualBox:~/Desktop$ pwd
/home/cbit/Desktop
```

**6.touch command:** The touch command is used to create empty files. We can create multiple empty files by executing it once.

**Syntax:** touch <file name>, touch <file1> <file2> ....

```
cbit@cbi-VirtualBox:~/Desktop/vidya$ ls
Cars Cities file1 fruits Names number State student
cbit@cbi-VirtualBox:~/Desktop/vidya$ touch file2
cbit@cbi-VirtualBox:~/Desktop/vidya$ touch file3
```

**7.cat command:** The cat command is a multi-purpose utility in the Linux system. It can be used to create a file, display content of the file, copy the content of one file to another file, and more.

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

```
vidya@vidya-VirtualBox:~/Desktop$ cat file
good morning
```

**8.rm command:** The rm command is used to remove a file.

**Syntax:** rm <file name>

```
cbit@cbi-VirtualBox:~/Desktop/vidya$ rm States
cbit@cbi-VirtualBox:~/Desktop/vidya$ ls
Cars fruits Names
```

**9.cp command:** The cp command is used to copy a file or directory.

**Syntax:** cp <existing file name> <new file name>

```
cbit@cbi-VirtualBox:~/Desktop/vidya$ cp fruits Cars
```

**10.mv command:** The mv command is used to move a file or a directory from one location to another location.

**Syntax:** mv <file name> <directory path>

```
cbit@cbi-VirtualBox:~/Desktop/vidya$ mkdir sarika
cbit@cbi-VirtualBox:~/Desktop/vidya$ pwd
/home/cbit/Desktop/vidya
cbit@cbi-VirtualBox:~/Desktop/vidya$ mv sarika student
cbit@cbi-VirtualBox:~/Desktop/vidya$ ls
Cars fruits Names student
cbit@cbi-VirtualBox:~/Desktop/vidya$
```

**11.rename command:** The rename command is used to rename files. It is useful for renaming a large group of files.

**Syntax:** rename 's/old-name/new-name/' files

**12.head command:** head command is used to display the content of a file. It displays the first 10 lines of a file.

**Syntax:** head <file name>1

```
cbit@cbit-VirtualBox:~/Desktop/vidya$ cat State
Telangana
Maharashtra
Karnataka
Gujarat
Arunachal Pradesh
Jammu & Kashmir
Punjab
Jharkhand
Orissa
Assam
Mizoram
Sikkim
Goa
cbit@cbit-VirtualBox:~/Desktop/vidya$ head -5 State
Telangana
Maharashtra
Karnataka
Gujarat
Arunachal Pradesh
```

**13.tail command:** The tail command is similar to the head command. The difference between both commands is that it displays the last ten lines of the file content. It is useful for reading the error message.

**Syntax:** tail <file name>

```
cbit@cbit-VirtualBox:~/Desktop/vidya$ tail -5 State
Orissa
Assam
Mizoram
Sikkim
Goa
```

**14.tac command:** The tac command is the reverse of cat command, as its name specified. It displays the file content in reverse order (from the last line).

**Syntax:** tac <file name>

```
file2: ASCII text
vidya@vidya-VirtualBox:~/Desktop$ tac file2
hello how are you
```

**15.Sudo command:** superuser do elevates a users permission to administrator or root

**Syntax:** sudo <command>

```
bhavya@LAPTOP-93P1453S:/mnt/c/Users/BHUYA HARINI/os$ sudo apt install ploacte
[sudo] password for bhavya:
Reading package lists... Done
Building dependency tree... Done
```

**16.Diff command:** Compares two files and prints difference

**Syntax:** diff <file1> <file2>

```
vidya@vidya-VirtualBox:~/Desktop$ diff file file2
1c1
< good morning
---
> hello how are you
```

**17.Chmod:** change mode command to change file and directory permissions

**Syntax:**chmod <permission> <file>

Chmod 777 file2.txt

**18.Ps:** Lists the current running process on system

**Syntax:**ps

```
vidya@vidya-VirtualBox:~/Desktop/vidya$ ps
  PID TTY          TIME CMD
 14961 pts/0    00:00:00 bash
 14975 pts/0    00:00:00 ps
```

**19.echo:**Echo command prints arguments to terminal

**Syntax:**echo <argument>

```
cbit@cbit-VirtualBox:~/Desktop/vidya$ echo Good Morning!
Good Morning!
```

**20.hostname:** To check DNS name of current machine

**Syntax:**hostname

```
vidya@vidya-VirtualBox:~/Desktop/vidya$ hostname
vidya-VirtualBox
```

**21.file:**Provides information about a file,printing filetype and content types

**Syntax:** file <filename>

```
vidya@vidya-VirtualBox:~/Desktop$ file file2
file2: ASCII text
```

**22.Whoami command:**Used to show the currently \* in user

**Syntax:**whoami

```
vidya@vidya-VirtualBox:~/Desktop/vidya$ whoami
vidya
```

**23.Psswd command:**To alter password from terminal

**Syntax:** psswd



```
/usr/bin/cat
```

```
vidya@vidya-VirtualBox:~/Desktop$ passwd
Changing password for vidya.
```

**24.Reboot command:** Reboot command restarts the system immediately from terminal

**Syntax :**reboot

**25.Which command:**Shows the path of executable program

**Syntax:**which <command>

```
vidya@vidya-VirtualBox:~/Desktop$ which cat
/usr/bin/cat
```

**26.Vim command:**Linux text editor that runs in the terminal.Creating and executing file

**Syntax:**vim <file>

```
vidya@vidya-VirtualBox:~/Desktop$ vim file2
Command 'vim' not found, but can be installed with:
sudo apt install vim          # version 2:8.2.3995-1ubuntu2.11, or
sudo apt install vim-tiny     # version 2:8.2.3995-1ubuntu2.11
sudo apt install neovim       # version 0.6.1-3
sudo apt install vim-athena   # version 2:8.2.3995-1ubuntu2.11
sudo apt install vim-gtk3     # version 2:8.2.3995-1ubuntu2.11
sudo apt install vim-nox      # version 2:8.2.3995-1ubuntu2.11
```

**27.What is command:**Quick way to determine what a command does

**Syntax:**what is <command>

```
vidya@vidya-VirtualBox:~/Desktop$ whatis cat
```

**28.Alias and unalias command:**Used for customization of commands

**Syntax:**alias <name>=command

```
vidya@vidya-VirtualBox:~/Desktop$ alias meow=cat
vidya@vidya-VirtualBox:~/Desktop$ meow file
good morning
```

**29.Cmp command:**Tells us the line number which is different

**Syntax:**cmp <file1> <file2>

```
cbit@cbit-VirtualBox:~/Desktop/vidya$ cmp fruits Cars
fruits Cars differ: byte 1, line 1
```

**30.useradd command:** The [useradd](#) command is used to add or remove a user on a Linux server.

**Syntax:** useradd username

```
vidya@vidya-VirtualBox:~/Desktop$ sudo useradd vidya1
[passwd] password for vidya:
```

**31.groupadd command:**

Syntax: groupadd <group name>

**32.grep command:**The [grep](#) is the most powerful and used filter in a Linux system. The 'grep' stands for "global regular expression print."

Syntax: Command | grep <search word>

```
cbit@cbit-VirtualBox:~/Desktop/vidya$ grep A State
Arunachal Pradesh
Assam
Andhra Pradesh
```

**33.comm command:** The '[comm](#)' command is used to compare two files or streams.

Syntax: comm <file1> <file2>

```
vidya@vidya-VirtualBox:~/Desktop$ comm file file2
good morning
hello how are you
```

**34.tr command:** Used to translate the file content like from lower case to upper case.

Syntax: command | tr <'old'> <'new'>

```
vidya@vidya-VirtualBox:~/Desktop$ cat file2 | tr 'l' 'L'
heLLo how are you
```

**35.uniq command:** The [uniq](#) command is used to form a sorted list in which every word will occur only once.

Syntax: Command <filename> | uniq

```
vidya@vidya-VirtualBox:~/Desktop$ uniq file2
hello how are you
```

**36.wc command:**The [wc](#) command is used to count the lines, words, and characters in a file.

Syntax: wc <file name>

```
vidya@vidya-VirtualBox:~/Desktop$ wc file
1 2 13 file
```

**37.od command:**The [od](#) command is used to display the content of a file in different s, such as hexadecimal, octal, and ASCII characters.

Syntax: od -b <filename> //octal format

od -t x1 <filename> //hexa decimal format

od -c <filename> //ASCII character format

```
vidya@vidya-VirtualBox:~/Desktop$ od -b file2
0000000 150 145 154 154 157 040 150 157 167 040 141 162 145 040 171 157
0000020 165 012
0000022
```

**38.sort command:**The [sort](#) command is used to sort files in alphabetical order.

**Syntax:** sort <file name>

```
vidya@vidya-VirtualBox:~/Desktop$ sort file
good morning
```

**39.gzip command:**The [gzip](#) command is used to truncate the file size. It is a compressing tool. It replaces the original file by the compressed file having '.gz' extension.

**Syntax:** gzip <file1> <file2> <file3>....

```
vidya@vidya-VirtualBox:~/Desktop$ gzip file file2
```

**40.gunzip command:**The [gunzip](#) command is used to decompress a file. It is a reverse operation of gzip command.

**Syntax:** gunzip <file1> <file2> <file3>

```
vidya@vidya-VirtualBox:~/Desktop$ gunzip file file2
```

**41.find command:** The [find](#) command is used to find a particular file within a directory. It also supports various options to find a file such as byname, by type, by date, and more.

**Syntax:** find . -name <filename>

```
vidya@vidya-VirtualBox:~/Desktop$ find -name file
```

**42.locate command:**The [locate](#) command is used to search a file by file name. It is quite similar to find command; the difference is that it is a background process

**Syntax:** locate <file name>

**43.date command:**The [date](#) command is used to display date, time, time zone, and more.

**Syntax:** date

```
cbit@cbit-VirtualBox:~/Desktop/vidya$ date
Friday 08 September 2023 11:43:55 AM IST
```

**44.cal command:**The [cal](#) command is used to display the current month's calendar with the current date highlighted.

**Syntax:** cal year

**45.sleep command:**The [sleep](#) command is used to hold the terminal by the specified amount of time. By default, it takes time in seconds.



**Syntax:** sleep <time>

```
vidya@vidya-VirtualBox:~/Desktop/vidya$ sleep 5
```

**46.time command:**The [time](#) command is used to display the time to execute a command.

**Syntax:** time

```
vidya@vidya-VirtualBox:~/Desktop/vidya$ time
real    0m0.000s
user    0m0.000s
sys     0m0.000s
```

**47.zcat command:**The zcat command is used to display the compressed files.

**Syntax:** zcat <filename>

```
vidya@vidya-VirtualBox:~/Desktop$ gzip file
vidya@vidya-VirtualBox:~/Desktop$ zcat file
good morning
```

**48.df command:**The [df](#) command is used to display the disk space used in the file system. It displays the output as in the number of used blocks, available blocks, and the mounted directory.

**Syntax:** df

```
vidya@vidya-VirtualBox:~/Desktop/vidya$ df
Filesystem      1K-blocks    Used Available Use% Mounted on
tmpfs           503212      1572    501640   1% /run
/dev/sda3       25600228 14561532   9712924  60% /
tmpfs           2516052      0    2516052   0% /dev/shm
tmpfs           5120         4      5116    1% /run/lock
/dev/sda2       524252      6216   518036    2% /boot/efi
tmpfs           503208      112   503096    1% /run/user/1000
```

**49.du:**Disk uage commad helps show how much space a file or directory takes up.

**Syntax:** du

```
vidya@vidya-VirtualBox:~/Desktop/vidya$ du
4
vidya@vidya-VirtualBox:~/Desktop/vidya$ df
```

**50.exit command:**Linux [exit](#) command is used to exit from the current shell. It takes a parameter as a number and exits the shell with a return of status number.

**Syntax:** exit

```
bhavya@LAPTOP-93P1453S:/mnt/c/Users/BHUYA HARINI/os$ exit
logout
```



**51.clear command:**Linux clear command is used to clear the terminal screen.

**Syntax:** clear

**52.ip command:**Linux [ip](#) command is an updated version of the ipconfig command. It is used to assign an IP address, initialize an interface, disable an interface.

**Syntax:** ip a or ip addr

**53.w command:**Shows a list of currently logged in users

**Syntax:**w

```
vidya@vidya-VirtualBox:~/Desktop/vidya$ w
 16:48:15 up 21 min,  1 user,  load average: 1.45, 1.64, 1.42
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU WHAT
vidya     tty2     tty2            16:27    21:23  0.02s  0.02s /usr/libexec/g
```

**54.type command:**Provides type of specific command

**Syntax:**type name

```
vidya@vidya-VirtualBox:~/Desktop/vidya$ type echo
echo is a shell builtin
```

**55. tee command:** The [tee](#) command is quite similar to the cat command. The only difference between both filters is that it puts standard input on standard output and also write them into a file.

**Syntax:** cat <filename> | tee <newFile> | cat or tac | .....

```
bhavya@LAPTOP-93P1453S:/mnt/c/Users/BHVVYA HARINI/os$ cat file3.txt | tee
new.txt |tac
OS recordbhavya@LAPTOP-93P1453S:/mnt/c/Users/BHVVYA HARINI/os$ |
```

**55.wait command:**It waits for a process to change its state i.e. it waits for any running process to complete and returns the exit status.

**Syntax:**wait PID

**56.ifconfig command:**It gives the list of all network interfaces along with the ip address,mac address and other information.

**Syntax:**~→>ifconfig

**57.userdel command:**Removes a user from system,sudo enables privileges

**Syntax:** sudo userdel <username>

```
vidya@vidya-VirtualBox:~/Desktop/vidya$ sudo userdel vidya
```

**58.history command:**Terminal sessions keep history log of commands

**Syntax:**history

**C B I T**

**66.close:**The `close()` function deallocates the file descriptor indicated by `fd`. To deallocate means to make the file descriptor available for return by subsequent calls to `open(2)` or other functions that allocate file descriptors. Syntax: `int close(int fd);`

**67.read:**The `read()` function attempts to read `nbyte` bytes from the file associated with the open file descriptor, `fd`, into the buffer pointed to by `buf`. Syntax: `ssize_t read(int fd, void *buf, size_t nbyte);`

**68.write**The `write()` function attempts to write `nbyte` bytes from the buffer pointed to by `buf` to the file associated with the open file descriptor, `fd`. Syntax: `ssize_t write(int fd, const void *buf, size_t nbyte);`

**69.lseek:**`lseek()` repositions the file offset of the open file description associated with the file descriptor `fd` to the argument `offset` according to the directive `whence`. Syntax: `off_t lseek(int fd, off_t offset, int whence);`

**70.stat**The `stat()` function obtains information about the file pointed to by `path`. Read, write, or execute permission of the named file is not required, but all directories listed in the path name leading to the file must be searchable. Syntax: `int stat(const char *restrict path, struct stat *restrict buf);`

**71.sync:**The `sync()` function writes all information in memory that should be on disk, including modified super blocks, modified inodes, and delayed block I/O. Syntax: `void sync(void);`


#### Demonstration of read,write,lseek,stat,sync

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<stdlib.h>
#include<sys/ioctl.h>
#include<sys/stat.h>
#define WR_VALUE_IOW('a','a',int*)
#define RD_VALUE_IOR('a','b',int*)
int main()
{
    int fd = open("sample.txt", O_RDONLY | O_CREAT);
    int fd2 = open("file2.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
    int fd3 = open("file3.txt", O_RDWR);
    char c[20];
    read(fd, &c, 20);
    printf("Text in file is: %s\n",c);
    write(fd2, c, read(fd, &c, 20));
    lseek(fd, 5, SEEK_CUR);
}
```


```
char c1[20];
read(fd, &c1, 20);
printf("Text after lseek 10 characters: %s", c1);
printf("Enter a value to write in the file: ");
int number;
scanf("%d", &number);
ioctl(fd3, WR_VALUE, (int*) &number);
struct stat sfile;
stat("stat.c", &sfile);
printf("st_mode = %o\n", sfile.st_mode);
close(fd);
close(fd2);
close(fd3);
return 0;
}
```

**OUTPUT:**


```
Text in file is: This is a sample textf
Text after lseek 10 characters: This is third line
Enter a value to write in the file: 12
st_mode = 400
```

 sample - Notepad

File Edit Format View Help  
This is a sample text.  
This is second line  
This is third line  
This is fourth line

 file2 - Notepad

File Edit Format View Help  
t.  
This is second 1

 file3 - Notepad

File Edit Format View Help  
12

**5.Aim:** Demonstration of Linux/Unix process related system calls: fork, wait, exec, exit, getpid, getuid, setuid, brk, nice, sleep.

**Description:**

**fork system call:** The Fork system call is used for creating a new process in Linux, and Unix systems, which is called the child process, which runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call. Negative value: The creation of a child process was unsuccessful. Zero: Returned to the newly created child process. Positive value: Returned to parent. The value contains the process ID of the newly created child process.

**wait system call:** A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent continues its execution after wait system call instruction.

**exec system call:** The exec system call is used to execute a file which is residing in an active process. When exec is called the previous executable file is replaced and new file is executed.

**Getpid:** getpid() is an inbuilt function defined in unistd.h library. Returns the process ID of the calling process. This is often used by routines that generate unique temporary filenames.

**Getuid:** The getuid() is an unbuild function in unistd.h library which returns the real user ID of the calling process. The real user ID identifies the person who is logged in.

**Setuid:** The setuid() function sets the real user ID, effective user ID, and saved user ID of the calling process. The setgid() function sets the real group ID, effective group ID, and saved group ID of the calling process. The setegid() and seteuid() functions set the effective group and user IDs respectively for the calling process. See [Intro\(2\)](#) for more information on real, effective, and saved user and group IDs.

**Brk:** brk() change the location of the program break, which defines the end of the process's data segment. brk() sets the end of the data segment to the value specified by addr, when that value is reasonable, the system has enough memory, and the process does not exceed its maximum data size.

**Nice:** nice() adds inc to the nice value for the calling thread. The range of the nice value +19(high priority) to -20(high priority).

**Sleep:** Sleep is a computer program and when you call this method, then it sets the process to wait until the specified amount of time proceeds, then goes and finds some other process to run.

**Programs:**

**Demonstration of fork system call**

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

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

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

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

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



```
int globalVariable=2;
int main()
{
    int localVariable=20,i=1;
    pid_t pID=fork();
    if (pID==0)
    {
        printf("In child process: ");
        globalVariable++;
        localVariable++;
    }
    else if (pID<0)
    {
        printf("Failed to fork");
        exit(1);
    }
    else
    {
        printf("Parent Process: ");
        printf("\nglobalVariable:%d",globalVariable);
        printf("\nStack variable:%d",localVariable);
        return 0;
    }
}
```

**Output:**

Parent Process:

globalVariable:2

Stack variable:20In child process:

globalVariable:3

Stack variable:21

**Demonstration of getpid and getppid**

```
#include<stdio.h>
#include<string.h>
#include<sys/types.h>
#include<unistd.h>
#include<stdlib.h>
int main()
{
    int pid;
    printf("\nWelcome to CBIT\n");
    pid=fork();
    if (pid==0)
    {
        printf("\nI am in child process....");
        printf("and my PID=%d and my parent PID=%d\n",getpid(),getppid());
    }
    else
    {
        printf("\nI am in parent process my PID=%d",getpid());
        printf("and my child PID=%d\n",pid);
    }
    return 0;
}
```

**Output:**

Welcome to CBIT

I am in parent process my PID=117and my child PID=118

I am in child process....and my PID=118 and my parent PID=117

**Demonstration of fork system call using wait, getpid, geppid and sleep**

```
#include<sys/types.h>
#include<errno.h>
#include<sys/wait.h>
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
int main()
{
    pid_t childpid;
    int retval,status;
    childpid=fork();
    if(childpid>=0)
    {
        if(childpid==0)
        {
            printf("child:Iam the Child process\n");
            printf("child:Here's my PID:%d\n",getpid());
            printf("child:My parent's pid is %d\n",getppid());
            printf("child:The value of my copy of childpid is %d\n",childpid);
            printf("child:sleeping for 10 seconds\n"); sleep(10);
            printf("child:Enter an exit value");
            scanf("%d",&retval);
            printf("child:Goodbye!\n");
            exit(retval);
        }
        else
        {
            printf("parent:Iam the parent process\n");
```



```
printf("parent:Here's my PID:%d\n",getpid());
printf("parent:The value of my copy of childpid is %d\n",childpid);
printf("parent:I will now wait for my child process to exit\n");
wait(&status);
if ( WIFEXITED(status) )
    printf("Parent: child has exited with status %d.\n", WEXITSTATUS(status));
printf("\nparent:Goodbye!\n");
exit(0);
}
}
else
{
    perror("fork");
    exit(0);
}
}
```

**Output:**

```
parent:Iam the parent process
parent:Here's my PID:135
parent:The value of my copy of childpid is 136
parent:I will now wait for my child process to exit
child:Iam the Child process
child:Here's my PID:136
child:My parent's pid is 135
child:The value of my copy of childpid is 0
child:sleeping for 10 seconds
child:Enter an exit value6
child:Goodbye!
Parent: child has exited with status 6.
```

parent:Goodbye!

**Demonstration of fork() using exec()**

```
#include<sys/wait.h>
#include<string.h>
#include<stdlib.h>
int main(int argc, char *args[])
{
    int pid ;
    pid=fork();
    if(pid<0)
    {
        fprintf(stderr, "fork failed");
        exit(-1);
    }
    else if(pid==0)
        execlp("/bin/ls", "ls", NULL);
    else
    {
        wait(NULL);
        printf("child complete");
        exit(0);
    }
}
```

**Output:**

a.out execdemo.c forkdemo.c getpid.c waitsleep. child complete

**Aim:** Write a Program to demonstrate First Come First Serve CPU scheduling

**Description :** First in, first out (FIFO), also known as first come, first served (FCFS), is the simplest scheduling algorithm. FIFO simply queues processes in the order that they arrive in the ready queue. In this, the process that comes first will be executed first and next process starts only after the previous gets fully executed.

**Program:**

```
#include<stdio.h>
int main()
{ int p[10],at[10],bt[10],ct[10],tat[10],wt[10],i,j,temp=0,n;
float awt=0,atat=0;
printf("enter no of process you want:");
scanf("%d",&n);
printf("enter %d process:",n);
for(i=0;i<n;i++)
scanf("%d",&p[i]);
printf("enter %d arrival time:",n);
for(i=0;i<n;i++)
scanf("%d",&at[i]);
printf("enter %d burst time:",n);
for(i=0;i<n;i++)
scanf("%d",&bt[i]);
for(i=0;i<n;i++)
{ for(j=0;j<(n-i);j++)
{ if(at[j]>at[j+1])
{temp=p[j+1];
p[j+1]=p[j];
p[j]=temp;
temp=at[j+1];
at[j+1]=at[j];
at[j]=temp;
temp=bt[j+1];
bt[j+1]=bt[j];
bt[j]=temp;
}}}
ct[0]=at[0]+bt[0];
for(i=1;i<n;i++)
{ temp=0;
if(ct[i-1]<at[i])
temp=at[i]-ct[i-1];
ct[i]=ct[i-1]+bt[i]+temp;
}
printf("\n p\t A.T\t B.T\t C.T\t TAT\t WT");
for(i=0;i<n;i++)
{
```

```
tat[i]=ct[i]-at[i];
wt[i]=tat[i]-bt[i];
atat+=tat[i];
awt+=wt[i];
}
atat=atat/n;
awt=awt/n;
for(i=0;i<n;i++)
{printf("\nP%d\t %d\t %d\t %d\t %d\t %d",p[i],at[i],bt[i],ct[i],tat[i],wt[i]);}
printf("\naverage turnaround time is %f",atat);
printf("\naverage wating timme is %f\n",awt);
return 0; }
```

**OUTPUT:**

```
enter no of proccess you want:3
enter 3 process:1 2 3
enter 3 arrival time:0 1 0
enter 3 burst time:4 2 3

p      A.T      B.T      C.T      TAT      WT
P1      0        4        4        4        0
P3      0        3        7        7        4
P0      0        0        7        7        7
average turnaround time is 6.000000
average wating timme is 3.666667
```

**Aim:** Write a Program to demonstrate Shortest Job First or Shortest Job next CPU scheduling

**Description:** The CPU scheduling algorithm Shortest Job First (SJF), allocates the CPU to the processes according to the process with smallest execution time. SJF uses both preemptive and non-preemptive scheduling. The preemptive version of SJF is called SRTF

**Program:(preemptive)**

```
#include <stdio.h>
int main()
{ int arrival_time[10], burst_time[10], temp[10];
  int i, smallest, count = 0, time, limit;
  double wait_time = 0, turnaround_time = 0, end;
  float average_waiting_time, average_turnaround_time;
  printf("\nEnter the Total Number of Processes:\t");
  scanf("%d", &limit);
  printf("\nEnter Details of %d Processes\n", limit);
  for(i = 0; i < limit; i++)
  {printf("\nEnter Arrival Time:\t");

  scanf("%d", &arrival_time[i]);
  printf("Enter Burst Time:\t");
  scanf("%d", &burst_time[i]);
  temp[i] = burst_time[i];}
  burst_time[9] = 9999;
  for(time = 0; count != limit; time++)
  {smallest = 9;
  for(i = 0; i < limit; i++)
  {if(arrival_time[i] <= time && burst_time[i] < burst_time[smallest] && burst_time[i] > 0)
  smallest = i;}
  burst_time[smallest]--;
  if(burst_time[smallest] == 0)
  {count++;
  end = time + 1;
  wait_time = wait_time + end - arrival_time[smallest] - temp[smallest];
  turnaround_time = turnaround_time + end - arrival_time[smallest]; }}
  average_waiting_time = wait_time / limit;
  average_turnaround_time = turnaround_time / limit;
  printf("\n\nAverage Waiting Time:\t%f\n", average_waiting_time);
  printf("Average Turnaround Time:%f\n", average_turnaround_time);
  return 0; }
```

**OUTPUT:**

```
Enter the Total Number of Processes:    3

Enter Details of 3 Processes

Enter Arrival Time:      1
Enter Burst Time:        2

Enter Arrival Time:      2
Enter Burst Time:        3

Enter Arrival Time:      0
Enter Burst Time:        2

Average Waiting Time:    1.000000
Average Turnaround Time:3.333333
```

**AIM:** Program to demonstrate the use of setuid(), setgid(), seteuid(), getgid(), geteuid(), getegid()

**PROCEDURE:** setuid() and setgid() change the real user and group IDs, while seteuid() sets the effective user ID, influencing process permissions. getuid(), getgid(), geteuid(), getegid() retrieve user and group identity information for the current process in Unix-like systems.

**CODE:**

```
#include<stdio.h>
#include<unistd.h>
int main()
{uid_t uid;
printf("Before setting the uid and gid\nreal user id: %d\nreal grp id: %d\neffective user id: %d\neffective grp id: %d\n",getuid(),getgid(),geteuid(),getegid());
setuid(1003);
setgid(1002);
```

```
printf("After setting the uid and gid\nreal user id: %d\nreal grp id: %d\neffective user id: %d\neffective grp id: %d\n",getuid(),getgid(),geteuid(),getegid());
return 0; }
```

**OUTPUT:**

```
Before setting the uid and gid
real user id: 1000
real grp id: 1000
effective user id: 1000
effective grp id: 1000
After setting the uid and gid
real user id: 1000
real grp id: 1000
effective user id: 1000
effective grp id: 1000
```

**Aim:** Write a Program to demonstrate the use of atexit() exit handler

**Description :** The atexit function in C registers a function to be called at normal program termination, allowing for cleanup or finalization tasks before the program exits.

**PROGRAM :**

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
int fd1;
int fd2;
void functionA () { // Exit Handler
close(fd1);
printf("We are in functionA and closed the file \n");}
void functionB () { // Exit Handler
close(fd2);
printf("We are in functionB and closed the file \n");}
int main () {
char buf[10];
fd1=open("blank.txt",O_RDONLY,742); // kk.txt should present in current folder
fd2=open("blank2.txt",O_RDONLY,S_IRWXU|S_IRGRP|S_IXOTH);
read(fd1,buf,10);
write(1,buf,10);
printf("\nCompleted the operation A \n");
atexit(functionA );
read(fd2,buf,10);
write(1,buf,10);
printf("\nCompleted the operation B \n");
atexit(functionB );
printf("Starting main Program...\n");
```



```
printf("Exiting main Program...\n");  
return(0);}
```

**OUTPUT:**

```
Completed the operation A  
  
Completed the operation B  
Starting main Program...  
Exiting main Program...  
We are in functionB and closed the file  
We are in functionA and closed the file
```

**AIM:** A program to demonstrate paging

**DESCRIPTION:**

In Operating Systems, Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages. The main idea behind the paging is to divide each

process in the form of pages. The main memory will also be divided in the form of frames. One page of the

process is to be stored in one of the frames of the memory. The pages can be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes. Pages of the

process are brought into the main memory only when they are required otherwise they reside in the secondary storage.

**CODE:**

```
#include<stdio.h>  
#define MAX 50  
int main()  
{int page[MAX],i,n,f,ps,off,pno;  
int choice=0;  
printf("\nEnter the no of pages in memory: ");  
scanf("%d",&n);  
printf("\nEnter page size: ");  
scanf("%d",&ps);  
printf("\nEnter no of frames: ");  
scanf("%d",&f);  
for(i=0;i<n;i++)  
page[i]=-1;  
printf("\nEnter the page table\n");  
printf("(Enter frame no as -1 if that page is not present in any frame)\n\n");  
printf("\npageno\tframeno\n-----\t-----");  
for(i=0;i<n;i++)  
{ printf("\n\n%d\t\t",i);  
scanf("%d",&page[i]); }
```



```
do { printf("\n\nEnter the logical address(i.e,page no & offset:");
scanf("%d%d",&pno,&off);
if(page[pno]==-1)
printf("\n\nThe required page is not available in any of frames");
else
printf("\n\nPhysical address(i.e,frame no & offset):%d,%d",page[pno],off);
printf("\nDo you want to continue(1/0)?");
scanf("%d",&choice);
}while(choice==1);
return 1; }
```

**OUTPUT:**

Enter the no of pages in memory: 3

Enter page size: 4

Enter no of frames: 3

Enter the page table

(Enter frame no as -1 if that page is not present in any frame)

pageno	frameno
-----	-----

0	2
---	---

1	3
---	---

2	1
---	---

Enter the logical address(i.e,page no & offset):4 42000

Physical address(i.e,frame no & offset):0,42000

Do you want to continue(1/0?):1

Enter the logical address(i.e,page no & offset):1 8500

Physical address(i.e,frame no & offset):3,8500

Do you want to continue(1/0?):0

**AIM:** A program to demonstrate the Linked File allocation techniques

**PROCEDURE:** It is easy to allocate the files because allocation is on an individual block basis. Each block contains a pointer to the next free block in the chain. Here also the file allocation table consists of a single entry for each file. Using this strategy any free block can be added to a chain very easily. There is a link between one block to another block, that's why it is said to be linked allocation.

**CODE :**

```
#include<stdio.h>
#include<stdlib.h>
int main()
{int f[50], p,i, st, len, j, c, k, a;
for(i=0;i<50;i++)
f[i]=0;
printf("Enter how many blocks already allocated: ");
scanf("%d",&p);
printf("Enter blocks already allocated: ");
for(i=0;i<p;i++)
{scanf("%d",&a);
f[a]=1; }
x: printf("Enter index starting block and length: ");
scanf("%d%d", &st,&len);
k=len;
if(f[st]==0)
{for(j=st;j<(st+k);j++)
{if(f[j]==0)
{f[j]=1;
printf("%d----->%d\n",j,f[j]);}
else
{printf("%d Block is already allocated \n",j);
k++; }}}
else
printf("%d starting block is already allocated \n",st);
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
return 0;}
```

**OUTPUT:**

```
Enter how many blocks already allocated: 3
Enter blocks already allocated: 1 0 2
Enter index starting block and length: 0 2
0 starting block is already allocated
Do you want to enter more file(Yes - 1/No - 0)1
Enter index starting block and length: 3 5
3----->1
4----->1
5----->1
6----->1
7----->1
Do you want to enter more file(Yes - 1/No - 0)0
```

**AIM:** A program to demonstrate the contiguous file allocation techniques.

**PROCEDURE:**

In this allocation strategy, each file occupies a set of contiguous blocks on the disk. This strategy is best suited for sequential files, the file allocation table consists of a single entry for each file. It shows the filenames, starting block of the file and size of the file. The main problem with this strategy is, it is difficult to find the contiguous free blocks in the disk and some free blocks could happen between two files.

**CODE:**

```
#include<stdio.h>
#include<stdlib.h>
int main()
{int f[50], i, st, len, j, c, k, count = 0;
for(i=0;i<50;i++)
f[i]=0;
printf("Files Allocated are : \n");
x: count=0;
printf("Enter starting block and length of files: ");
scanf("%d%d", &st,&len);
for(k=st;k<(st+len);k++)
if(f[k]==0)
count++;
if(len==count)
{for(j=st;j<(st+len);j++)
if(f[j]==0)
{f[j]=1;
printf("%d\t%d\n",j,f[j]);}}
```

```
if(j!=(st+len-1))
printf(" The file is allocated to disk\n");
}else
printf(" The file is not allocated \n");
printf("Do you want to enter more file(Yes - 1/No - 0)");
scanf("%d", &c);
if(c==1)
goto x;
else
exit(0);
return 0;}
```

**OUTPUT :**

```
Files Allocated are :
Enter starting block and length of files: 0 5
0      1
1      1
2      1
3      1
4      1
The file is allocated to disk
Do you want to enter more file(Yes - 1/No - 0)1
Enter starting block and length of files: 2 3
The file is not allocated
Do you want to enter more file(Yes - 1/No - 0)0
```

**AIM:** A program to demonstrate segmentation

**PROCEDURE:**

Segmentation is a memory management technique that splits up the virtual address space of an application into chunks. By splitting the memory up into manageable chunks, the operating system can track which parts of the memory are in use and which parts are free. This makes allocating and deallocating memory much faster and simpler for the operating system. The segments are of unequal size and are not placed in a contiguous way.

**CODE:**

```
#include<stdio.h>
int main()
{ int a[10][10],b[100],i,j,n,x,base,size,seg,off;
printf("Enter the segments count\n");
scanf("%d",&n);
```

```
for(i=0;i<n;i++) {
printf("Enter the %d size \n",i+1);
scanf("%d",&size);
a[i][0]=size;
printf("Enter the base address\n");
scanf("%d",&base);
a[i][1]=base;
for(j=0;j<size;j++)
{
x=0;
scanf("%d",&x);
base++;
b[base]=x;
}}
printf("Enter the segment number and offset value \n");
scanf("%d%d",&seg,&off);
if(off<a[seg][0]) {
int abs=a[seg][1]+off;
printf("the offset is less tha %d",a[seg][0]);
printf("\n %d + %d = %d\n",a[seg][1],off,abs);
printf("the element %d is at %d ",b[abs+1],abs);
}else
{ printf("Error in locating\n"); }}
```

**OUTPUT :**

```
Enter the segments count
2
Enter the 1 size
3
Enter the base address
45
12
15
14
Enter the 2 size
2
Enter the base address
78
9
6
Enter the segment number and offset value
2 120
Error in locating
```

**AIM:** A program to demonstrate the use of signal – IPC

**DESCRIPTION:**

A signal is a software generated interrupt that is sent to a process by the OS because of when user press ctrl-c or another process tell something to this process. There are fix set of signals that can be sent to a process. signal are identified by integers. Signal number have symbolic names. For example SIGCHLD is number of the signal sent to the parent process when child terminates.

**CODE:**

```
#include<stdio.h>
#include<signal.h>
void handle_sigint(int sig){
printf("caught signal %d\n",sig);}
int main(){ //signal is a function
signal(SIGINT,handle_sigint);
while(1)
{ printf("hello world\n");
sleep(1);} //press ctrl+c to stop
return 0; }
```

**OUTPUT:**



```
hello world
hello world
hello world
hello world
^Ccaught signal 2
hello world
hello world
hello world
^Ccaught signal 2
hello world
hello world
^Z
[2]+  Stopped                  ./a.out
```

**AIM:** A program to demonstrate the use of SIGCHLD

**DESCRIPTION:**

The name "SIGCHLD" stands for Signal Child. When a process forks a child process using the fork() system call, the parent process can receive a SIGCHLD signal when the child process terminates. This signal allows the parent process to perform actions such as cleaning up resources associated with the child process or obtaining information about the exit status of the child.

**CODE:**

```
#include<stdio.h>
#include<signal.h>
#include<unistd.h> //press ctrl+c to provoke function
void handle_sigchild(int sig)
{printf("inside child signal %d\n",sig);}
int main()
{ signal(SIGCHLD,handle_sigchild);
  int p = fork();
  if(p==0){
    printf("inside child process \n");
    printf("end of child process \n"); }
  else if(p>=0)
  { printf("inside parent process\n");
    wait();
    printf("end of parent process\n");}
  return 0; }
```

**OUTPUT:**

```
inside parent process
inside child process
end of child process
inside child signal 17
end of parent process
```



**3. WAP to demonstrate the use of shared memory – IPC**

**AIM:** A program to demonstrate the use of shared memory

**DESCRIPTION:** Shared Memory is the fastest inter-process communication (IPC) method. The operating system maps a memory segment in the address space of several processes so that those processes can read and write in that memory segment. Two functions: `shmget()` and `shmat()` are used for IPC using shared memory. `shmget()` function is used to create the shared memory segment while `shmat()` function is used to attach the shared segment with the address space of the process.

**CODE:**

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{ int i;
void *shared_mem; // a void ptr, not a function like signal()
char buff[100];
int shmid;
shmid=shmget((key_t)2345,1024,0666|IPC_CREAT);
//created with key=2345,1024 bytes, ipc created with 0666 permission
printf("key of shared memory = %d\n",shmid);
shared_mem = shmat(shmid,NULL,0);
printf("process attached at %p\n",shared_mem);
//enter data
printf("enter data to write for shared memory\n");
read(0,buff,100);
strcpy(shared_mem,buff);
printf("data entered : %s\n",(char *)shared_mem);
return 0;}
```

**OUTPUT:**

```
key of shared memory = 0
process attached at 0x7f8d5a292000
enter data to write for shared memory
sky is blue
data entered : sky is blue
```



**4. WAP to demonstrate the use of semaphore – IPC**

**AIM:** A program to demonstrate the use of semaphore – IPC

**DESCRIPTION:**

Semaphore is an integer variable which is accessed or modified by using two atomic operations:

wait() and

signal(). In C program the corresponding operations are sem\_wait() and sem\_post(). Here, we write a

Program for Process Synchronization using Semaphores to understand the implementation of

sem\_wait()

and sem\_signal() to avoid a race condition.

**CODE:**

```
#include<stdio.h>
#include<unistd.h>
#include<semaphore.h>
#include<pthread.h>
void *f1();
void *f2();
int shared=1;
sem_t s;
int main()
{ sem_init(&s,0,1); //address of semaphore,no of processes,initial value
pthread_t thread1,thread2;
pthread_create(&thread1,NULL,f1,NULL);
pthread_create(&thread2,NULL,f2,NULL);
pthread_join(thread1,NULL);
pthread_join(thread2,NULL);
printf("final value of shared = %d\n",shared);}
void *f1()
{int x;
sem_wait(&s);
x=shared;
printf("thread1 :\n\treads value as %d\n",x);
x++;
printf("\tlocal updation\tvalue = %d\n",x);
sleep(1);
shared=x;
printf("val of shared var updated by thread1 = %d\n",shared);
sem_post(&s);}
void *f2()
{ int y;
sem_wait(&s);
y=shared;
printf("thread2 :\n\treads value as %d\n",y);
y--;
printf("\tlocal updation\tvalue = %d\n",y);
```

```
sleep(1);
shared=y;
printf("val of shared var updated by thread1 = %d\n",shared);
sem_post(&s);}
```

**OUTPUT :**

```
thread1 :
    reads value as 1
    local updation value = 2
val of shared var updated by thread1 = 2
thread2 :
    reads value as 2
    local updation value = 1
val of shared var updated by thread1 = 1
final value of shared = 1
```

**5. WAP to demonstrate the use of message queue – IPC**

**AIM:** A program to demonstrate the use of message queue - IPC

**DESCRIPTION:**

Program for IPC using Message queues are almost similar to named pipes with the exception that they do

not require the opening and closing of pipes. But, they face one similar problem like named pipes; blocking


on full pipes. Message queues send blocks of data from one process to another. Each block of data is considered to have a type. There is an upper limit on the maximum size of each block and also a limit on

the maximum total size of all blocks on all queues in the system.

**CODE:**

```
#include<string.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#define maxtext 512
struct my_msg
{long int msgType;
char text[maxtext]; };
int main()
{ int running=1,msgid;
struct my_msg data;
char buff[50];
msgid=msgget((key_t)14534,0666|IPC_CREAT);
if(msgid== -1)
```

```
{ printf("error in queue creation\n");
exit(0);}
while(running)
{printf("enter some text\n");
fgets(buff,50,stdin);
data.msgType = 1;
strcpy(data.text,buff);
if(msgsnd(msgid,(void*)&data,maxtext,0)==-1)
{ printf("msg not sent\n");}
if(strncmp(buff,"end",3)==0)
{running=0;} } return 0;}
```

**OUTPUT:**

```
enter some text
sky is blue
enter some text
hello world
enter some text
karthikeya swami
enter some text
end
```

**10. WAP to demonstrate the use of socket – IPC**

**AIM:** A program to demonstrate the use of socket – IPC

**DESCRIPTION:**

A socket is a bi-directional data transfer mechanism. They are used to transfer data between two processes.

The two processes can be running on the same system as Unix-domain or loopback sockets, or on different

systems as network sockets. To be more precise, it's a way to talk to other computers using standard Unix

file descriptors. A Unix Socket is used in a client-server application framework. A server is a process that

performs some functions on request from a client. Most of the application-level protocols like FTP, SMTP,

and POP3 make use of sockets to establish connection between client and server and then to exchange data.

**CODE: 1. SOCKET:**

```
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define MAX 80
#define PORT 8080
#define SA struct sockaddr // Function designed for chat between client and server.
void func(int connfd)
{char buff[MAX];
int n;
for (;;) { // infinite loop for chat
bzero(buff, MAX); // read the message from client and copy it in buffer
read(connfd, buff, sizeof(buff));
printf("From client: %s\t To client : ", buff); // print buffer which contains the client contents
bzero(buff, MAX);
n = 0;
while ((buff[n++] = getchar()) != '\n');
write(connfd, buff, sizeof(buff));
if (strncmp("exit", buff, 4) == 0) {
printf("Server Exit...\n");
break;
}}}
// Driver function
int main()
{ int sockfd, connfd, len;
struct sockaddr_in servaddr, cli;
// socket create and verification
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd == -1) {
printf("socket creation failed...\n");
exit(0);}
else
printf("Socket successfully created..\n");
bzero(&servaddr, sizeof(servaddr));
// assign IP, PORT
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(PORT);
if ((bind(sockfd, (SA*)&servaddr, sizeof(servaddr))) != 0) {
printf("socket bind failed...\n");
exit(0); }
else
printf("Socket successfully binded..\n");
```

```
if ((listen(sockfd, 5)) != 0) {
    printf("Listen failed...\n");
    exit(0);}
else
    printf("Server listening..\n");
len = sizeof(cli);
connfd = accept(sockfd, (SA*)&cli, &len);
if (connfd < 0) {
    printf("server accept failed...\n");
    exit(0);}
else
    printf("server accept the client...\n");
func(connfd);
close(sockfd);}

2. CLIENT:
#include <arpa/inet.h> // inet_addr()
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <strings.h> // bzero()
#include <sys/socket.h>
#include <unistd.h> // read(), write(), close()
#define MAX 80
#define PORT 8080
#define SA struct sockaddr
void func(int sockfd)
{
    char buff[MAX];
    int n;
    for (;;) {
        bzero(buff, sizeof(buff));
        printf("Enter the string : ");
        n = 0;
        while ((buff[n++] = getchar()) != '\n')
            ;
        write(sockfd, buff, sizeof(buff));
        bzero(buff, sizeof(buff));
        read(sockfd, buff, sizeof(buff));
        printf("From Server : %s", buff);
        if ((strcmp(buff, "exit", 4)) == 0) {
            printf("Client Exit...\n");
            break;
        }
    }
}

int main()
{ int sockfd, connfd;
  struct sockaddr_in servaddr, cli;
```

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd == -1) {
    printf("socket creation failed...\n");
    exit(0); }
else
    printf("Socket successfully created..\n");
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
servaddr.sin_port = htons(PORT);
if (connect(sockfd, (SA*)&servaddr, sizeof(servaddr))
    != 0) {
    printf("connection with the server failed...\n");
    exit(0); }
else
    printf("connected to the server..\n");
func(sockfd);
close(sockfd); }
```

**OUTPUT:**

```
Socket successfully created..
connected to the server..
Enter the string : end
```



Laboratory Record

of \_\_\_\_\_

Sheet No. \_\_\_\_\_

Experiment No. \_\_\_\_\_

Date \_\_\_\_\_

--

Roll No.

**C B I T**

Laboratory Record

of \_\_\_\_\_

Sheet No. \_\_\_\_\_

Experiment No. \_\_\_\_\_

Date \_\_\_\_\_

--

Roll No.

**C B I T**

Laboratory Record

of \_\_\_\_\_

Sheet No. \_\_\_\_\_

Experiment No. \_\_\_\_\_

Date \_\_\_\_\_

--

Roll No.

**C B I T**

Laboratory Record

of \_\_\_\_\_

Sheet No. \_\_\_\_\_

Experiment No. \_\_\_\_\_

Date \_\_\_\_\_

--

Roll No.

**C B I T**

Laboratory Record

of \_\_\_\_\_

Sheet No. \_\_\_\_\_

Experiment No. \_\_\_\_\_

Date \_\_\_\_\_

--

Roll No.

**C B I T**

Laboratory Record

of \_\_\_\_\_

Sheet No. \_\_\_\_\_

Experiment No. \_\_\_\_\_

Date \_\_\_\_\_

--

Roll No.

**C B I T**



Laboratory Record

of \_\_\_\_\_

Sheet No. \_\_\_\_\_

Experiment No. \_\_\_\_\_

Date \_\_\_\_\_

--

Roll No.

**C B I T**