		a. tar/cpio/dd commands b. Recovering single / multiple files.	
Week – 15	*	Software Package Management in Linux. a. Redhat Tools – rpm and yum b. Debian Tools – dpkg and apt-get	86 – 90
Week – 16		Lab Internal – 2	91
		*****Content beyond the university prescribed syllabi.	

WEEK – 1

1. Write a shell script that accepts a file name, starting and ending numbers as arguments and displays all the lines between the given line numbers.

AIM: To write a shell script that accepts a file name, starting and ending numbers as arguments and displays all the lines between the given line numbers.

DESCRIPTION:

Echo: Echo allows a user to repeat, or "echo," a string variable to standard output. More on using the echo command with shell scripts.

Read: Read is used to read lines of text from standard input and to assign values of each field in the input line to shell variables for further processing.

Head: The head command reads the first ten lines of a any given file name.

Tail: The tail command allows you to display last ten lines of any text file. Similar to the head command above, tail command also support options '**n**' number of lines and '**n**' number of characters

PROGRAM:

```
echo "Enter the file name"

read fname

echo "enter starting line number"

read sl

echo "enter ending line number"

read el

d=`expr $el - $sl`

if [ -f $fname ]

then
```

```
echo " the lines between $sl and $el of given file are "

head -$el $fname | tail -$d

else

echo "file doesnt exist"

fi
```

OUTPUT:

```
[root@localhost ~]# vi 1s.sh
[root@localhost ~]# ./1s.sh
bash: ./1s.sh: Permission denied
[root@localhost ~]# chmod 777 1s.sh
[root@localhost ~]# ./1s.sh
enter the filename
sales.dat
enter the starting line number
2
enter the ending line number
4
1       computers       9161
1       textbooks       21312           2       clothing        3
```

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

AIM: To write a shell script that deletes all lines containing the specified word in one or more files supplied as arguments to it.

DESCRIPTION:

Grep: Grep searches files for a given character string or pattern and can replace the string with another. This is one method of searching for files within Linux.

PROGRAM:

```
if [ $# -ne 0 ]
then
echo enter the word
read word
for fname in $*
do
if [ -f $fname ]
then
echo the given input filename is:$fname
grep -v "$word" $fname
else
echo its not a file
fi
done
else
echo "enter atleast one argument as input"
fi
```

OUTPUT:
[root@localhost ~]# sed -f del.sed del dell here using unix there are some difference between unix and windowsnt

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

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

DESCRIPTION:

Chmod: Chmod changes the access mode (permissions) of one or more files. Only the owner of a file or a privileged user may change the access mode.

PROGRAM:

```
echo "List of Files which have Read, Write and Execute Permissions in Current

Directory"

for file in *

do

if [ -r $file -a -w $file -a -x $file ]

then

echo $file

fi

done
```

OUTPUT:
student@ubuntu:~$sh prg3.sh enter the directory name dir1 ff has all permissions files not having permissions

WEEK – 2

4. Write a shell script that receives any number of file names as arguments checks if every argument supplied is a file as its arguments, counts and reports the occurrence of each word that is present in the first argument file on other argument files.

AIM: To write a shell script that receives any number of file names as arguments checks if every argument supplied is a file as its arguments, counts and reports the occurrence of each word that is present in the first argument file on other argument files.

DESCRIPTION:

WC: wc counts the number of words, lines and characters in text files and produces a count for multiple files if several files are selected.

PROGRAM:

```
for x in $*

do

if [ -f $x ]

then

echo "$x is a file"

echo "no of lines in the file are"

wc -l $x

elif [ -d $x ]

then

echo "$x is a directory"

else
```

```
echo "enter valid filename or directory name"

fi

done
```

OUTPUT:

```
guest-glcbIs@ubuntu:~$sh lprg4.sh dir1 d1
dir1 is a directory
d1 is a file
no of lines in the file are 2
```

5. Write a shell script that accepts a list of file names as its arguments, counts and reports the occurrence of each word that is present in the first argument file on other argument files.

AIM: To write a shell script that accepts a list of file names as its arguments, counts and reports the occurrence of each word that is present in the first argument file on other argument files.

DESCRIPTION:

Set: set is a built-in function of the Bourne shell (sh), C shell (csh), and Korn shell (ksh), which is used to define and determine the values of the system environment.

PROGRAM:

```
echo " Enter the number of files::"

read n

echo "Enter the "n" files ::"

read fn

set $fn

for i in `cat $1`
```

```
do
echo " word = $i"
echo "------------"
grep -c "$i" $*
echo "     "
done
```

OUTPUT:
$sh 9a.sh hegde.sh ravi.sh Raghu 2 Hary 1 Vinay 9

6. Write a shell script to list all of the directory files in a directory.

AIM: To write a shell script to list the entire directory files in a directory**.**

DESCRIPTION:

Ls: The ls command lists files and directories within the current working directory, allowing admins to see when configuration files were last edited.

PROGRAM:

```
echo "enter directory name"
read dir
if [ -d $dir ]
then
echo "list of files in the directory"
ls $dir
else
```

```
echo "enter proper directory name"
fi
```

OUTPUT:

```
guest-glcbIs@ubuntu:~$sh lprg6.sh
enter directory name
dir1
list of files in the directory
drwxrwxr-x 4 guest-glcbls guest-glcbls 140 2012-07-06 14:40 dir1
```

WEEK – 3

7. Write a shell script to find factorial of a given number.

AIM: To Write a shell script to find factorial of a given number.

DESCRIPTION:

The shell has similarities to the DOS command processor Command.com (actually Dos was design as a poor copy of UNIX shell), it's actually much more powerful, really a programming language in its own right.

A shell is always available on even the most basic UNIX installation. User have to go through the shell to get other programs to run. User can write programs using the shell. User uses the shell to administrate your UNIX system.

For example:

ls -al | more

is a short shell program to get a long listing of the present directory and route the output through the more command.

Shell responsibilities

1. Program Execution
2. Variable and Filename Substitution
3. I/O Redirection
4. Pipeline Hookup
5. Environment Control
6. Interpreted Programming Language

Creating a Script:

To create a shell script, first use a text editor to create a file containing the commands. For example, type the following commands and save them as first.sh

PROGRAM:

```
echo " Factorial"

echo "Enter a number: "

read f

fact=1

factorial=1

while [ $fact -le $f ]

do

factorial=`expr $factorial \* $fact`

fact=`expr $fact + 1`

done

echo "Factorial of $f = $factorial"
```

OUTPUT:

```
guest-glcbIs@ubuntu:~$sh lprg7.sh
enter a number
4
Factorial of 4 is 24
```

8. Write an awk script to count the number of lines in a file that do not contain vowels.

AIM: To write a script to count lines without vowels.

DESCRIPTION:

Awk: The awk command is a powerful method for processing or analyzing text files—in particular, data files that are organized by lines (rows) and columns. Simple awk commands can be run from the command line.

PROGRAM:

```
#!/bin/bash
echo "Enter file name"
read file
awk '$0!~/[aeiou]/{ count++ }
END{print "The number of lines that does not contain vowels are: ",count}' $file
```

OUTPUT:

```
[singh@00-13-02-56-15-7c programs]$ vi test1
engineering
data
and
lab
workshop
programming
rdx
bpb
hp
[singh@00-13-02-56-15-7c programs]$ sh raj11.sh
Enter file name
test1
The number of lines that does not contain vowels are:  3
```

9. Write a awk script to find the number of characters, words and lines in a file.

AIM: To write a awk script to find the number of characters, words and lines in a file.

DESCRIPTION:

Awk: Awk is one of the most powerful tools in Unix used for processing the rows and columns in a file. Awk has built in string functions and associative arrays. awk supports most of the operators, conditional blocks, and loops available in C language.

PROGRAM:

```
BEGIN {print "record.\t characters \t words"}
#BODY section{
len=length($0)
total_len =len
print(NR,":\t",len,":\t",NF,$0)
words =NF
}
END{
print("\n total")
print("characters :\t" total len)
print("lines :\t" NR) }
```

OUTPUT:

```
Student@ubuntu:~$ awk –f cnt.awk ff1
Record  words
1:          5:          1hello
Total
Characters:5
Lines:1
```

WEEK – 4

10. Write a C Program that makes a copy of a file using standard I/O and system calls.

AIM: To write a C Program that makes a copy of a file using standard I/O and system calls.

DESCRIPTION:

System calls are provided by UNIX to access and control files and devices.

open()	Open a file or device
read()	Read from an open file or device
write()	write to a file or device
close()	close the file or device

PROGRAM:

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

```
int

main (int argc, char **argv)

{

int argno;

for (argno = 1; argno < argc; argno  )

typefile (argv[argno]);

exit (0);

}
```

OUTPUT:

```
student@ubuntu:~$gcc –o prg10.out prg10.c
student@ubuntu:~$cat > ff
hello
hai
student@ubuntu:~$./prg10.out ff
hello
hai
```

11. Implement in C the following UNIX commands using system calls

A). cat B). mv

AIM: To create a C program by using system calls open(), read() and write() which works as cat and mv commands in shell script.

A) Cat

Description:

cat linux command concatenates files and print it on the standard output.

SYNTAX:

The Syntax is

cat [OPTIONS] [FILE]...

OPTIONS:

-A	Show all.
-b	Omits line numbers for blank space in the output.
-e	A $ character will be printed at the end of each line prior to a new line.
-E	Displays a $ (dollar sign) at the end of each line.
-n	Line numbers for all the output lines.
-s	If the output has multiple empty lines it replaces it with one empty line.
-T	Displays the tab characters in the output.
-v	Non-printing characters (with the exception of tabs, new-lines and form-feeds) are printed visibly.

Example:

To Create a new file: cat > file1.txt

This command creates a new file file1.txt. After typing into the file press control+d (^d) simultaneously to end the file.

1. To Append data into the file: cat >> file1.txt
   To append data into the same file use append operator >> to write into the file, else the file will be overwritten (i.e., all of its contents will be erased).

2. To display a file: cat file1.txt
   This command displays the data in the file.

3. To concatenate several files and display: cat file1.txt file2.txt

PROGRAM:

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

#include<sys/stat.h>

#include<stdio.h>

#include<fcntl.h>

main( int argc,char *argv[3] )

{

int fd,i;
```

```
char buf[2];
fd=open(argv[1],O_RDONLY,0777);
if(fd==-argc)
{
printf("file open error");
}
else
{
while((i=read(fd,buf,1))>0)
{
printf("%c",buf[0]);
}
close(fd);
}
}
```

OUTPUT:

```
student@ubuntu:~$gcc –o prgcat.out prgcat.c
student@ubuntu:~$cat > ff
hello
hai
student@ubuntu:~$./prgcat.out ff
hello
hai
```

B) mv

DESCRIPTION:

mv COMMAND:

mv command which is short for move. It is used to move/rename file from one directory to another. mv command is different from cp command as it completely removes the file from the source and moves to the directory specified, where cp command just copies the content from one file to another.

SYNTAX:

The Syntax is

mv [-f] [-i] oldname newname

OPTIONS:

-f	This will not prompt before overwriting (equivalent to --reply=yes). mv -f will move the file(s) without prompting even if it is writing over an existing target.
-i	Prompts before overwriting another file.

PROGRAM:

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

#include<sys/stat.h>

#include<stdio.h>

#include<fcntl.h>

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

int i,fd1,fd2;

char *file1,*file2,buf[2];

file1=argv[1];

file2=argv[2];

printf("file1=%s file2=%s",file1,file2);
```

```
fd1=open(file1,O_RDONLY,0777);

fd2=creat(file2,0777);

while(i=read(fd1,buf,1)>0)

write(fd2,buf,1);

remove(file1);

close(fd1);

close(fd2);

}
```

OUTPUT:

```
student@ubuntu:~$gcc –o mvp.out mvp.c
student@ubuntu:~$cat > ff
hello
hai
student@ubuntu:~$./mvp.out ff ff1
student@ubuntu:~$cat  ff
cat:ff:No such file or directory
student@ubuntu:~$cat ff1
hello
hai
```

WEEK – 5

12. Write a C program to list files in a directory

AIM: To write a C program to list files in a directory.

DESCRIPTION:

The **opendir** function opens a directory and establishes a directory stream.

```
#include <sys/types.h>
#include <sys/dir.h>
DIR *opendir( const char * name);
```

The **telldir** function returns a value that records the current position in a directory stream.

The **seekdir** function sets the directory entry pointer in the directory stream given by **dirp**.

The **closedir** function closes a directory stream and frees up the resources associated with it.

PROGRAM:

```
#include <sys/types.h>
#include <sys/dir.h>
#include <sys/param.h>
#include <stdio.h>
#define FALSE 0
#define TRUE 1
extern  int alphasort();
char pathname[MAXPATHLEN];
main()  {
int count,i;
struct dirent **files;
```

```
int file_select();
if (getwd(pathname) == NULL )
{
printf("Error getting pathn");
exit(0);
}
printf("Current Working Directory = %sn",pathname);
count = scandir(pathname, &files, file_select, alphasort);
if (count <= 0)
{
printf("No files in this directoryn");
exit(0);
}
printf("Number of files = %dn",count);
for (i=1;i<count 1;  i)
printf("%s  \n",files[i-1]->d_name);
}
int file_select(struct direct   *entry)
{
if ((strcmp(entry->d_name, ".") == 0) ||(strcmp(entry->d_name, "..") == 0))
return (FALSE);
else
return (TRUE);
}
```

OUTPUT:
Student@ubuntu:~$ gcc list.c Student@ubuntu:~$ ./a.out Current working directory=/home/student/ Number of files=5

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

AIM: To write a C program to emulate the UNIX ls-l command.

DECRIPTION:

The exec family of functions replaces the current process with another created according to the arguments given.

PROGRAM:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
int main()
{
int pid;            //process id
pid = fork();     //create another process
if ( pid < 0 )
{                           //fail
printf("\nFork failed\n");
exit (-1);
```

```
}
else if ( pid == 0 )
{                    //child
execlp ( “/bin/ls”, “ls”, “-l”, NULL );  //execute ls
}
else
{                          //parent
wait (NULL);               //wait for child
printf(“\nchild complete\n”);
exit (0);
}}
```

OUTPUT:

```
guest-glcbIs@ubuntu:~$gcc –o lsc.out lsc.c
guest-glcbIs@ubuntu:~$./lsc.out
total 100
-rwxrwx—x 1 guest-glcbls guest-glcbls 140 2012-07-06 14:55 f1
drwxrwxr-x 4 guest-glcbls guest-glcbls 140 2012-07-06 14:40 dir1
child complete
```

14. Write a C program to list for every file in a directory, it's inode number and file name.

AIM: To write a C program to list for every file in a directory, it's inode number and file name.

DESCRIPTION:

This program emulates the command 'ls -l'

List of all the files in the current directory with inode numbers

Inodes:

- A structure that is maintained in a separate area of the hard disk.
- File attributes are stored in the inode.
- Every file is associated with a table called the inode.
- The inode is accessed by the inode number.
- Inode contains the following attributes of a file: file type, file permissions , no. of links

UID of the owner, GID of the group owner, file size date and time of last modification, last access, change.

FILE ATTRIBUTES	
Attribute	**value meaning**
File type	type of the file
Access permission	file access permission for owner, group and others Hard link count no.of hard links of a file.
UID	file owner user ID.
GID	the file group ID.
File size	file size in bytes.
Inode number	system inode number of the file.
File system ID	file system ID where the file is stored.

PROGRAM:

```
#include<stdlib.h>

#include<stdio.h>

#include<string.h>

main(int argc, char *argv[])

{

char d[50];

if(argc==2)

{

bzero(d,sizeof(d));

strcat(d,"ls ");

strcat(d,"-i ");

strcat(d,argv[1]);

system(d);

}

else

printf("\nInvalid No. of inputs");

}
```

OUTPUT:

```
student@ubuntu:~$ mkdir dd

student@ubuntu:~$ cd dd

student@ubuntu:~/dd$ cat >f1

hello

^z

student@ubuntu:~/dd$ cd

student@ubuntu:~$gcc –o flist.out flist.c

student@ubuntu:~$./flist.out dd

hello

46490 f1
```

WEEK – 6

15. Write a C Program that demonstrates redirection of standard output to a file.

EX: ls>f1.

AIM: To write a C Program that demonstrates redirection of standard output to a file .EX:ls>f1.\

DESCRIPTION:

This program demonstrates redirection of standard output to a file.

./15redir [<text-file>] ["<string>"]

*** Arguments:** If no arguments are supplied the program creates/appends the text "This text was redirected from standard output" to a text file by name "redir.txt". If the first argument <text-file> is supplied then this same text is appended to the specified text file instead of the default file redir.txt". If the second argument is supplied then it must be a text string enclosed in double quotes "". If the second argument is supplied then it will be appended to the supplied <text-file> using output redirection.

PROGRAM:

```
#include<stdlib.h>

#include<stdio.h>

#include<string.h>

main(int argc, char *argv[])

{

char d[50];

if(argc==2)

{

bzero(d,sizeof(d));
```

```
strcat(d,"ls ");
strcat(d,"> ");
strcat(d,argv[1]);
system(d);
}
else
printf("\nInvalid No. of inputs");
}
```

OUTPUT:

```
student@ubuntu:~$ gcc –o std.out std.c
student@ubuntu:~$ls
downloads  documents  listing.c  listing.out  std.c  std.out
student@ubuntu:~$ cat > f1
^z
student@ubuntu:~$./std.out f1
student@ubuntu:~$cat f1
downloads
documents
listing.c
listing.out
std.c
std.out
```

16. Write a C program to create a child process and allow the parent to display "parent" and the child to display "child" on the screen.

AIM: To write a C program to create a child process and allow the parent to display "parent" and the child to display "child" on the screen.

DESCRIPTION:

This program demonstrates creation of child process.

Each of parent and child processes displays a message on screen

fork Function:

The only way a new process is created by the UNIX kernel is when an existing process calls the fork function.

#include<sys/types.h>

#include<unistd.h> pid_t fork(void);

Return: 0 is child; process ID of child in parent, -1 on error

The new process created by fork is called child process. This is called once, but return twice that is the return value in the child is 0, while the return value in the parent is the process ID of the new child. The reason the child's process ID is returned to the parent is because a process can have more than one child, so there is no function that allows a process to obtain the process IDs of its children. The reason fork return 0 to the child is because a process can have only a single parent, so that child can always call getppid to obtain the process ID of its parent.

Both the child and parent contain executing with the instruction that follows the call to fork. The child is copy of the parent. For example, the child gets a copy of the parent's data space, heap and stack. This is a copy for the child the parent and children don't share these portions of memory. Often the parent and child share the text segment, if it is read-only.

PROGRAM:

```
#include <stdio.h>

#include <sys/wait.h>  /* contains prototype for wait */

int main(void)

{

int pid;

int status;

printf("Hello World!\n");

pid = fork( );

if(pid == -1) /* check for error in fork */

{

perror("bad fork");

exit(1);

}

if (pid == 0)

printf("I am the child process.\n");

else

{

wait(&status); /* parent waits for child to finish */

printf("I am the parent process.\n");

}

}
```

OUTPUT:
student@ubutnu:$gcc –o child.out child.c student@ubutnu: ./child.out Hello World! I am the child process. I am the parent process

WEEK – 7

LAB INTERNAL – 1

WEEK – 8

17. Write a C program to create a Zombie process.

AIM: To write a C program to create a Zombie process.

DESCRIPTION:

This program creates a zombie process.

Zombie Processes:

When a child process terminates, an association with its parent survives until the parent in turn either terminates normally or calls wait.

This terminated child process is known as a zombie process.

PROGRAM:

```
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main ()
{
  int pid_t child_pid;
  child_pid = fork ();
  if (child_pid > 0) {
    sleep (60);
  }
  else {
    exit (0);
```

```
 }
 return 0;
}
```

OUTPUT:
guest-glcbIs@ubuntu:~$gcc zombie.c guest-glcbIs@ubuntu:~$./a.out Then command prompt will wait for some time (60 sec) and then again command prompt will appear later.

18. Write a C program that illustrates how an orphan is created.

AIM: To write a C program that illustrates how an orphan is created.

DESCRIPTION:

Orphan Process:

- When the parent dies first the child becomes Orphan.
- The kernel clears the process table slot for the parent.

PROGRAM:

```
#include <stdio.h>
main()
{
int pid ;
printf("I'am the original process with PID %d and PPID %d.\n",getpid(),getppid());
pid=fork();
if(pid!=0 )
```

```
{
printf("I'am the parent with PID %d and PPID %d.\n",getpid(),getppid());
printf("My child's PID is %d\n",pid) ;
}
else
{
sleep(4);
printf("I'm the child with PID %d and PPID %d.\n",getpid(), getppid()) ;
}
printf ("PID %d terminates.\n", getpid()) ;
}
```

OUTPUT:

```
guest-glcbIs@ubuntu:~$gcc  –o prg18.out prg18.c
guest-glcbIs@ubuntu:~$./prg18.out
I am the original process with PID2242 and PPID1677.
I am the parent with PID2242 and PPID1677
My child’s PID is 2243
PID2243 terminates.
$  I am the child with PID2243 and PPID1.
PID2243 terminates.
```

Note: Run this program in shell. To see the zombie status of the child process type 'ps -e -o pid, ppid, stat, cmd | grep Z' in another terminal window and see the child status

19. Write a program that illustrates how to execute two commands concurrently with a command pipe.

AIM: To write a program that illustrates how to execute two commands concurrently with a command pipe.

DESCRIPTION:

A First-in, first-out (FIFO) file is a pipe that has a name in the filesystem. It is also called as named pipes.

FIFO creation:

int mkfifo (const char *pathname, mode_t mode);

-makes a FIFO special file with name pathname.

(Mode specifies the FIFO's permissions, as common in UNIX-like file systems).

-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 a FIFO special file has been created, any process can open it for reading or writing, in the same way as an ordinary file.

PROGRAM:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
int main()
{
```

```
int pfds[2];
char buf[30];
if(pipe(pfds)==-1)
{
perror("pipe failed");
exit(1);
}
if(!fork())
{
close(1);
dup(pfds[1];
system (“ls –l”);
}
else
{
printf("parent reading from pipe \n");
while(read(pfds[0],buf,80))
printf("%s \n" ,buf);
}
}
```

OUTPUT:

```
[student@gcet ~]$ vi pipes2.c

[student@gcet ~]$ cc pipes2.c

[student@gcet ~]$ ./a.out

Parent reading from pipe

Total 24

-rwxrwxr-x l student student 5563Aug 3 10:39 a.out

-rw-rw-r—l

Student student 340 jul 27 10:45 pipe2.c

-rw-rw-r—l student student

Pipes2.c

-rw-rw-r—l student student 401 34127 10:27 pipe2.c

Student
```

WEEK – 9

20. Write a C program that illustrates communication between two unrelated processes using named pipe.

AIM: To write a C program that illustrates communication between two unrelated processes using named pipe.

DESCRIPTION:

This program illustrates how to execute two commands concurrently with a command pipe.

./20namedpipe_a ["<LHS-command>|<RHS-command>"]

Arguments: Either no arguments or 1 argument is allowed. If no arguments are supplied then a default left-hand-side command 'ls -l' is assumed, and a default right-hand-side command 'sort' is assumed. If an argument is supplied then it must include a pipe '|' and whole argument should be enclosed between double quotes. The correctness of LHS command and RHS command is totally at the discretion of the user

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<unistd.h>
int main()
{
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],"test",5);
printf("reading from file descriptor #%d\n ", pfds[0]);
read(pfds[0],buf,5);
printf("read\"%s\"\n" ,buf);
}
```

OUTPUT:

```
[student@gcet ~]$ vi pipes1.c
[student@gcet ~]$ cc pipes1.c
[student@gcet ~]$ ./a.out
writing to file descriptor #4
reading from file descriptor #3
read"test"
```

21. Write a C program to create a message queue with read and write permissions to write 3 messages to it with different priority numbers.

AIM: To create a message queue with read and write permissions to write 3 messages to it with different priority numbers.

DESCRIPTION:

Messagequeue: This is an easy way of passing message between two processes. It provides a way of sending a block of data from one process to another. The main advantage of using this is, each block of data is considered to have a type, and a receiving process receives the blocks of data having different type values independently.

Creation and accessing of a message queue:

You can create and access a message queue using the msgget() function.

#include<sys/msg.h>

int msgget(key_t key,int msgflg);

The first parameter is the key value, which specifies the particular message queue. The special constant IPC_PRIVATE will create a private queue. But on some Linux systems the message queue may not actually be private.

The second parameter is the flag value, which takes nine permission flags.

```
#include <stdio.h>
 #include <sys/ipc.h>
#include <fcntl.h>
#define MAX 255 struct mesg
{
long type;
char mtext[MAX];
```

```
}
 *mesg;
char buff[MAX];
main()
{
 int mid,fd,n,count=0;
 if((mid=msgget(1006,IPC_CREAT | 0666))<0)
{
 printf("\n Can't create Message Q");
exit(1);
}
printf("\n Queue id:%d", mid);
mesg=(struct mesg *)malloc(sizeof(struct mesg));
 mesg ->type=6;
 fd=open("fact",O_RDONLY);
while(read(fd,buff,25)>0)
{
strcpy(mesg ->mtext,buff);
 if(msgsnd(mid,mesg,strlen(mesg ->mtext),0)== -1) printf("\n Message Write Error");
}
if((mid=msgget(1006,0))<0)
{
 printf("\n Can't create Message Q");
exit(1);
```

```
}
while((n=msgrcv(mid,&mesg,MAX,6,IPC_NOWAIT))>0)
write(1,mesg.mtext,n);
 count++;
 if((n= = -1)&(count= =0))
 printf("\n No Message Queue on Queue:%d",mid);
 }
```

WEEK – 10

22. Write a C program to create a message queue with read and write permissions to write 3 messages to it with different priority numbers.

AIM: To write a C program to create a message queue with read and write permissions to write 3 messages to it with different priority numbers.

DESCRIPTION:

Adding a message:

The msgsnd() function allows to add a message to a message queue.

#include<sys/msg.h>

int msgsnd(int msqid,const void *msg_ptr ,size_t msg_sz,int msgflg);

The first parameter is the message queue identifier returned from an msgget function.

The second parameter is the pointer to the message to be sent. The third parameter is the size of the message pointed to by msg_ptr. The fourth parameter, is the flag value controls what happens if either the current message queue is full or within the limit. On success, the function returns 0 and a copy of the message data has been taken and placed on the message queue, on failure -1 is returned.

PROGRAM:

```
#include <stdio.h>
#include <sys/ipc.h>
#include <fcntl.h>
#define MAX 255
    struct mesg
    {
```

```
        long type;
        char mtext[MAX];
    }  *mesg;
    char buff[MAX];
main()
{
    int mid,fd,n,count=0;;
    if((mid=msgget(1006,IPC_CREAT | 0666))<0)
    {
        printf("\n Can't create Message Q");
        exit(1);
    }
    printf("\n Queue id:%d", mid);
    mesg=(struct mesg *)malloc(sizeof(struct mesg));
    mesg ->type=6;
    fd=open("fact",O_RDONLY);
    while(read(fd,buff,25)>0)
    {
        strcpy(mesg ->mtext,buff);
        if(msgsnd(mid,mesg,strlen(mesg ->mtext),0)== -1)
            printf("\n Message Write Error");
    }
    if((mid=msgget(1006,0))<0)
```

```
{
        printf("\n Can't create Message Q");
        exit(1);
}
while((n=msgrcv(mid,&mesg,MAX,6,IPC_NOWAIT))>0)
        write(1,mesg.mtext,n);
        count  ;
if((n= = -1)&(count= =0))
        printf("\n No Message Queue on Queue:%d",mid);
}
```

OUTPUT:

```
Student@ubuntu:~$ gcc msgq.c
Student@ubuntu:~$ cat > fact
Hello
Hai
Welcome
^z
Student@ubuntu:~$ ./msgq.out
Queue id :0
Mesgq created
Hello
Hai
Welcome
```

23. Write a C program that receives the messages (From the above message queue as specified in (21) and display them.

AIM: To write a C program that receives the messages (From the above message queue as specified in (21) and display them.

PROGRAM:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#define MSGSZ 128
/*
 * Declare the message structure.  */
typedef struct msgbuf {
    long    mtype;
    char    mtext[MSGSZ];
} message_buf;
main()
{
    int msqid;
    key_t key;
    message_buf  rbuf;
    /*
     * Get the message queue id for the
     * "name" 1234, which was created by
```

```
     * the server.
     */
    key = 1234;
    if ((msqid = msgget(key, 0666)) < 0) {
        perror("msgget");
        exit(1);
    }
    /*
     * Receive an answer of message type 1.
     */
    if (msgrcv(msqid, &rbuf, MSGSZ, 1, 0) < 0) {
        perror("msgrcv");
        exit(1);
    }
     /*
     * Print the answer.
     */
    printf("%s\n", rbuf.mtext);
    exit(0);
}
```

EXECUTION STEPS:

```
[student@gcet ~]$cc message_send.c

[student@gcet ~]$ mv a.out msgsend

[student@gcet ~]$  ./msgsend

msgget: Calling msgget(0x4d2,01666)

msgget: msgget succeeded: msqid = 0

msgget: msgget succeeded: msqid = 0

msgget: msgget succeeded: msqid = 0

Message: "Did you get this?" Sent

[student@gcet ~]$ cc message_rec.c

[student@gcet ~]$ mv a.out msgrec

[student@gcet ~]$./msgrec
```

OUTPUT:

```
[1] 2907

[student@gcet ~]$ Did you get this?
```

WEEK – 11

24. Write a C program that illustrates suspending and resuming processes using signals.

AIM: To write a C program that illustrates suspending and resuming processes using signals.

DESCRIPTION:

sleep Function

#include<unistd.h>

Unsigned int sleep(unsigned int seconds); Returns: 0 or number of unslept seconds.

This function causes the calling process to be suspended until either:

1. The amount of clock that is specified by seconds has elapsed or
2. A signal is caught by the process and the signal handler returns.

In case 1 the return value is 0 when sleep returns early, because of some signal being caught case 2, the return value is the number of unslept seconds.

PROGRAM:

```
#include <stdio.h>

#include <ospace/unix.h>

int child_function()

{

  while (true) // Loop forever.

    {

    Printf("Child loop\n");

    os_this_process::sleep( 1 );
```

```
    }
  return 0; // Will never execute.
}
int main()
{
  os_unix_toolkit initialize;
  os_process child ( child function ); // Spawn child.
  os_this_process::sleep( 4 );
  printf("child.suspend()\n");
  child.suspend();
  printf("Parent sleeps for 4 seconds\n");
  os_this_process::sleep (4);
  printf("child.resume()");
  child.resume ();
  os_this_process::sleep (4);
  printf("child.terminate()");
  child.terminate ();
  printf("Parent finished");
  return 0;
}
```

OUTPUT:

```
Child loop
Child loop
Child loop
Child loop
Child loop
child.suspend()
Parent sleeps for 4 seconds
child.resume()
Child loop
Child loop
Child loop
Child loop
child.terminate()
Child loop
Parent finished
```

25. Write client and server programs (using c) for interaction between server and client processes using Unix Domain sockets.

AIM: To write client and server programs (using c) for interaction between server and client processes using Unix Domain sockets.

DESCRIPTION:

A socket is a bidirectional communication device that can be used to communicate withanother process on the same machine or with a process running on other machines.Sockets are the only inter process communications we'll discuss in this chapter thatpermit communication between processes on different computers. Internet programs such as Telnet, rlogin, FTP, talk, and the World Wide Web use sockets.

PROGRAM:

client1.c:

```
#include <stdio.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <string.h>
int main(void)
{
 struct sockaddr_un address;
 int  socket_fd, nbytes;
 char buffer[256];
 socket_fd = socket(PF_UNIX, SOCK_STREAM, 0);
```

```
if(socket_fd < 0)
{
 printf("socket() failed\n");
 return 1;
}
/* start with a clean address structure */
memset(&address, 0, sizeof(struct sockaddr_un));
address.sun_family = AF_UNIX;
snprintf(address.sun_path, UNIX_PATH_MAX, "./demo_socket");
if(connect(socket_fd,  (struct sockaddr *) &address,  sizeof(struct sockaddr_un)) != 0)
{
 printf("connect() failed\n");
 return 1;
}
nbytes = snprintf(buffer, 256, "hello from a client");
write(socket_fd, buffer, nbytes);
nbytes = read(socket_fd, buffer, 256);
buffer[nbytes] = 0;
printf("MESSAGE FROM SERVER: %s\n", buffer);
close(socket_fd);
return 0;
}
```

server1.c:

```
#include <stdio.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
int connection_handler(int connection_fd)
{
 int nbytes;
 char buffer[256];
 nbytes = read(connection_fd, buffer, 256);
 buffer[nbytes] = 0;
 printf("MESSAGE FROM CLIENT: %s\n", buffer);
 nbytes = snprintf(buffer, 256, "hello from the server");
 write(connection_fd, buffer, nbytes);
 close(connection_fd);
 return 0;
}
int main(void)
{
 struct sockaddr_un address;
 int socket_fd, connection_fd;
```

```
socklen_t address_length;
pid_t child;
socket_fd = socket(PF_UNIX, SOCK_STREAM, 0);
if(socket_fd < 0)
{
 printf("socket() failed\n");
 return 1;
}
unlink("./demo_socket");
/* start with a clean address structure */
memset(&address, 0, sizeof(struct sockaddr_un));
address.sun_family = AF_UNIX;
snprintf(address.sun_path, UNIX_PATH_MAX, "./demo_socket");
if(bind(socket_fd,  (struct sockaddr *) &address,  sizeof(struct sockaddr_un)) != 0)
{
 printf("bind() failed\n");
 return 1;
}
if(listen(socket_fd, 5) != 0)
{
 printf("listen() failed\n");
 return 1;
}
```

```
while((connection_fd = accept(socket_fd,
(struct sockaddr *) &address,
&address_length)) > -1)
{
 child = fork();
 if(child == 0)
 {
  /* now inside newly created connection handling process */
  return connection_handler(connection_fd);
 }
 /* still inside server process */
 close(connection_fd);
}
close(socket_fd);
unlink("./demo_socket");
return 0;
}
```

OUTPUT:

```
Student@ubuntu:~$gcc –o server1.out server1.c

Student@ubuntu:~$gcc client1.out client1.c

Student@ubuntu:~$ ./client1.out

MESSAGE FROM SERVER hello from the server

Student@ubuntu:~$ ./server1.out

MESSAGE FROM CLIENT hello from the CLIENT
```

WEEK – 12

26. Write a client and server programs (using c) for interaction between server and client processes using Internet Domain sockets.

AIM: To write a client and server programs (using c) for interaction between server and client processes using Internet Domain sockets.

DESCRIPTION:

Internet-Domain Sockets

UNIX-domain sockets can be used only for communication between two processes on the same computer. Internet-domain sockets, on the other hand, may be used to connect processes on different machines connected by a network.

Sockets connecting processes through the Internet use the Internet namespace represented by PF_INET. The most common protocols are TCP/IP. The Internet Protocol (IP), a low-level protocol, moves packets through the Internet, splitting and rejoining the packets, if necessary. It guarantees only “best-effort” delivery, so packets may vanish or be reordered during transport. Every participating computer is specified using a unique IP number. The Transmission Control Protocol (TCP), layered on top of IP, provides reliable connection-ordered transport. It permits telephone-like connections to be established between computers and ensures that data is delivered reliably and in order.

PROGRAM:

Server program:

```
#include <stdio.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

void error(char *msg)
```

```
{
   perror(msg);
   exit(1);
}
int main(int argc, char *argv[])
{
    int sockfd, newsockfd, portno, clilen;
    char buffer[256];
    struct sockaddr_in serv_addr, cli_addr;
    int n;
   if (argc < 2)
   {
      fprintf(stderr,"ERROR, no port provided\n");
      exit(1);
    }
      sockfd = socket(AF_INET, SOCK_STREAM, 0);
   if (sockfd < 0)
      error("ERROR opening socket");
      bzero((char *) &serv_addr, sizeof(serv_addr));
      portno = atoi(argv[1]);
      serv_addr.sin_family = AF_INET;
      serv_addr.sin_addr.s_addr = INADDR_ANY;
      serv_addr.sin_port = htons(portno);
```

```
if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
   error("ERROR on binding");
   listen(sockfd,5);
   clilen = sizeof(cli_addr);
   newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
 if ( newsockfd < 0)
   error ("ERROR on accept");
   bzero ( buffer,256);
   n = read(newsockfd,buffer,255);
 if  (n < 0)
   error("ERROR reading from socket");
   printf("Here is the message: %s\n",buffer);
   n = write(newsockfd,"I got your message",18);
 if (n < 0) error("ERROR writing to socket");
   return 0;
}
```

Client Program:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
void error(char *msg)
```

```
{    perror(msg);
     exit(0);
}
int main(int argc, char *argv[])
{
  int sockfd, portno, n;
  struct sockaddr_in serv_addr;
  struct hostent *server;
  char buffer[256];
  if (argc < 3)
  {
     fprintf(stderr,"usage %s hostname port\n", argv[0]);
     exit(0);
  }
     portno = atoi(argv[2]);
     sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if (sockfd < 0)
     error("ERROR opening socket");
     server = gethostbyname(argv[1]);
  if (server == NULL)
  {
     fprintf(stderr,"ERROR, no such host\n");
     exit(0);
```

```
}
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,(char *)&serv_addr.sin_addr.s_addr,server->h_length);
    serv_addr.sin_port = htons(portno);
  if (connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr)) < 0)
    error("ERROR connecting");
    printf("Please enter the message: ");
    bzero(buffer,256);
    fgets(buffer,255,stdin);
    n = write(sockfd,buffer,strlen(buffer));
  if (n < 0)
    error("ERROR writing to socket");
    bzero(buffer,256);
    n = read(sockfd,buffer,255);
  if (n < 0)
   error("ERROR reading from socket");
   printf("%s\n",buffer);
   return 0;
}
```

OUTPUT:
Student@ubuntu:~$gcc –o server2.out server2.c Student@ubuntu:~$gcc –o client2.out client2.c Please enter the message Hello world Hello world I got ur message Student@ubuntu:~$ ./server2.out Here is the message hello world

27. Write a C program that illustrates inters process communication using shared memory.

AIM: To Write a C program that illustrates inter process communication using shared memory.

DESCRIPTION:

This program illustrates the inter-process communication using shared memory. This header file defines some global constants and other structures which are used in both processes.

PROGRAM:

```
#include <stdio.h>

#include <sys/shm.h>

#include <sys/stat.h>

int main ()

{
```

```
int segment_id;

char* shared_memory;

struct shmid_ds shmbuffer;

int segment_size;

const int shared_segment_size = 0x6400;

/* Allocate a shared memory segment. */

segment_id = shmget (IPC_PRIVATE, shared_segment_size,

IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);

/* Attach the shared memory segment. */

shared_memory = (char*) shmat (segment_id, 0, 0);

printf (“shared memory attached at address %p\n”, shared_memory);

/* Determine the segment’s size. */

shmctl (segment_id, IPC_STAT, &shmbuffer);

segment_size = shmbuffer.shm_segsz;

printf (“segment size: %d\n”, segment_size);

/* Write a string to the shared memory segment. */

printf (shared_memory, “Hello, world.”);

/* Detach the shared memory segment. */

shmdt (shared_memory);

/* Reattach the shared memory segment, at a different address. */

shared_memory = (char*) shmat (segment_id, (void*) 0x5000000, 0);
```

```
printf (“shared memory reattached at address %p\n”, shared_memory);

/* Print out the string from shared memory. */

printf (“%s\n”, shared_memory);

/* Detach the shared memory segment. */

shmdt (shared_memory);

/* deallocate the shared memory segment. */

shmctl (segment_id, IPC_RMID, 0);

return 0;}
```

WEEK – 13

1. The Ext3 file system

The ext3 file system is essentially an enhanced version of the ext2 file system. These improvements provide the following advantages:

1. Availability
2. Data Integrity
3. Speed
4. Easy Transition
5. Default Inode Sizes Changed
6. New Mount Option: data_err
7. More Efficient Storage Use

Creating an Ext3 File System:

The steps for creating an ext3 file system are as follows:

1. Format the partition with the ext3 file system using mkfs.
2. Label the file system using e2label.

Converting to an Ext3 file system

The tune2fs command converts an ext2 file system to ext3.

```
# tune2fs -j block_device
```

block_device contains the ext2 file system to be converted.

A valid block device can be one of two types of entries:

A mapped device

A logical volume in a volume group, for example, /dev/mapper/VolGroup00-LogVol02.

A static device

A traditional storage volume, for example, /dev/sdbX, where sdb is a storage device name and X is the partition number.

Reverting to an Ext2 file system

1. Unmount the partition by logging in as root and typing:

```
# umount /dev/mapper/VolGroup00-LogVol02
```

2. Change the file system type to ext2 by typing the following command:

```
# tune2fs -O ^has_journal /dev/mapper/VolGroup00-LogVol02
```

3. Check the partition for errors by typing the following command:

```
# e2fsck -y /dev/mapper/VolGroup00-LogVol02
```

4. Then mount the partition again as ext2 file system by typing:

```
# mount -t ext2 /dev/mapper/VolGroup00-LogVol02 /mount/point
```

In the above command, replace */mount/point* with the mount point of the partition.

2. The Ext4 file system.

The ext4 file system is a scalable extension of the ext3 file system, which was the default file system of Red Hat Enterprise Linux 5. Ext4 is the default file system of Red Hat Enterprise Linux 6, and can support files and file systems up to 16 terabytes in size.

Allocation Features

The ext4 file system features the following allocation schemes:

1. Persistent pre-allocation
2. Delayed allocation
3. Multi-block allocation
4. Stripe-aware allocation

Other Ext4 Features

The ext4 file system also supports the following:

- *Extended attributes* (xattr) — This allows the system to associate several additional name and value pairs per file.
- *Quota journaling* — This avoids the need for lengthy quota consistency checks after a crash.
- *Subsecond timestamps* — This gives timestamps to the subsecond.

Creating an Ext4 File System

To create an ext4 file system, use the mkfs.ext4 command. In general, the default options are optimal for most usage scenarios:

```
# mkfs.ext4 /dev/device
```

An ext4 file system can be mounted with no extra options. For example:

```
# mount /dev/device /mount/point
```

An ext4 file system may be grown while mounted using the resize2fs command:

```
# resize2fs /mount/device node
```

3. Advanced Directory Navigations

Change to the home directory from anywhere

$typing 'cd /home/<your-home-directory-name>'

$ pwd

/usr/include/netipx

$ cd

$ pwd

/home/himanshu

Switch between directories using cd

Suppose your current working directory is this:

$ pwd
/home/himanshu/practice

and you want to switch to the directory **/usr/bin/X11,** and then switch back to the directory mentioned above. So what will you do? The most straight forward way is :

$ cd /usr/bin/X11
$ cd /home/himanshu/practice/

Although it seems a good way out, it really becomes tiring when the path to the directories is very long and complicated. In those cases, you can use the 'cd -' command.

While using 'cd -' command, the first step will remain the same, i.e., you have to do a cd <path> to the directory to you want to change to, but for coming back to the previous directory, just do a 'cd -', and that's it.

$ cd /usr/bin/X11
$ cd -
/home/himanshu/practice
$ pwd
/home/himanshu/practice

And if you want to again go back to the last directory, which in this case is /usr/bin/X11, run the 'cd -' command again. So you can see that using 'cd -' you can switch between directories easily. The only limitation is that it works with the last switched directories only.

Switch between directories using pushd and popd

If you closely analyse the 'cd -' trick, you'll find that it helps switching between only the last two directories, but what if there is a situation in which you switch multiple directories, and then want to switch back to the first one. For example, if you switch from directory A to directory B, and then to directory C and directory D. Now, you want to change back to Directory A.

As a general solution, you can type 'cd' followed by the path to directory A. But then again, if the path is long or complicated, the process can be time-consuming, especially when you have to switch between them frequently.

In these kind of situations, you can use the 'pushd' and 'popd' commands. The 'pushd' command saves the path to a directory in memory, and the 'popd' command removes it, and switches back to it too.

For example:

$ **pushd .**
/usr/include/netipx /usr/include/netipx
$ cd /etc/hp/
$ cd /home/himanshu/practice/
$ cd /media/
$ **popd**
/usr/include/netipx
$ pwd
/usr/include/netipx

So you can see that I used 'pushd' command to save the path to current working directory (represented by .), and then changed multiple directories. To come back to the saved directory, I just executed the 'popd' command.

NOTE- You can also use 'pushd' command to switch back to the saved directory, but that doesn't remove it from the memory, like 'popd' does.

GETTING FAMILIAR WITH VI/VIM EDITORS ADVANCE CONCEPTS.

Starting the vi Editor

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

S.No.	Command & Description
1	**vi filename** Creates a new file if it already does not exist, otherwise opens an existing file.
2	**vi -R filename** Opens an existing file in the read-only mode.
3	**view filename** Opens an existing file in the read-only mode.

Following is an example to create a new file **testfile** if it already does not exist in the current working directory –

```
$vi testfile
```

The above command will generate the following output −

```
|
~~
~
~
"testfile" [New File]
```

Operation Modes

While working with the vi editor, we usually come across the following two modes −

- **Command mode** − This mode enables you to perform administrative tasks such as saving the files, executing the commands, moving the cursor, cutting (yanking) and pasting the lines or words, as well as finding and replacing. In this mode, whatever you type is interpreted as a command.
- **Insert mode** − This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and placed in the file.
- vi always starts in the **command mode**. To enter text, you must be in the insert mode for which simply type **i**. To come out of the insert mode, press the **Esc** key, which will take you back to the command mode.
- **Hint** − If you are not sure which mode you are in, press the Esc key twice; this will take you to the command mode. You open a file using the vi editor. Start by typing some characters and then come to the command mode to understand the difference.

Getting Out of vi

The command to quit out of vi is **:q**. Once in the command mode, type colon, and 'q', followed by return. If your file has been modified in any way, the editor will warn you of this, and not let you quit. To ignore this message, the command to quit out of vi without saving is **:q!**. This lets you exit vi without saving any of the changes.

The command to save the contents of the editor is **:w**. You can combine the above command with the quit command, or use **:wq** and return.

The easiest way to **save your changes and exit vi** is with the ZZ command. When you are in the command mode, type **ZZ**. The **ZZ** command works the same way as the **:wq** command.

If you want to specify/state any particular name for the file, you can do so by specifying it after the **:w**. For example, if you wanted to save the file you were working on as another filename called **filename2**, you would type **:w filename2** and return.

Moving within a File

To move around within a file without affecting your text, you must be in the command mode (press Esc twice). The following table lists out a few commands you can use to move around one character at a time −

S.No.	Command & Description
1	**K** Moves the cursor up one line
2	**J** Moves the cursor down one line
3	**H** Moves the cursor to the left one character position
4	**L** Moves the cursor to the right one character position

The following points need to be considered to move within a file −

- vi is case-sensitive. You need to pay attention to capitalization when using the commands.
- Most commands in vi can be prefaced by the number of times you want the action to occur. For example, **2j** moves the cursor two lines down the cursor location.

There are many other ways to move within a file in vi. Remember that you must be in the command mode (**press Esc twice**). The following table lists out a few commands to move around the file −

Given below is the list of commands to move around the file.

Control Commands

The following commands can be used with the Control Key to performs functions as given in the table below −

Given below is the list of control commands.

Editing Files

To edit the file, you need to be in the insert mode. There are many ways to enter the insert mode from the command mode −

S.No.	Command & Description
1	**i** Inserts text before the current cursor location
2	**I** Inserts text at the beginning of the current line
3	**a** Inserts text after the current cursor location
4	**A** Inserts text at the end of the current line
5	**o** Creates a new line for text entry below the cursor location
6	**O** Creates a new line for text entry above the cursor location

Deleting Characters

Here is a list of important commands, which can be used to delete characters and lines in an open file –

S.No.	Command & Description
1	**x** Deletes the character under the cursor location
2	**X** Deletes the character before the cursor location
3	**dw** Deletes from the current cursor location to the next word
4	**d^** Deletes from the current cursor position to the beginning of the line
5	**d$** Deletes from the current cursor position to the end of the line
6	**D** Deletes from the cursor position to the end of the current line
7	**dd** Deletes the line the cursor is on

Change Commands

You also have the capability to change characters, words, or lines in vi without deleting them. Here are the relevant commands −

S.No.	Command & Description
1	**cc** Removes the contents of the line, leaving you in insert mode.
2	**cw** Changes the word the cursor is on from the cursor to the lowercase **w** end of the word.
3	**r** Replaces the character under the cursor. vi returns to the command mode after the replacement is entered.
4	**R** Overwrites multiple characters beginning with the character currently under the cursor. You must use **Esc** to stop the overwriting.
5	**s** Replaces the current character with the character you type. Afterward, you are left in the insert mode.
6	**S** Deletes the line the cursor is on and replaces it with the new text. After the new text is entered, vi remains in the insert mode.

Copy and Paste Commands

You can copy lines or words from one place and then you can paste them at another place using the following commands −

S.No.	Command & Description
1	**yy** Copies the current line.
2	**yw** Copies the current word from the character the lowercase w cursor is on, until the end of the word.
3	**p** Puts the copied text after the cursor.
4	**P** Puts the yanked text before the cursor.

Advanced Commands

There are some advanced commands that simplify day-to-day editing and allow for more efficient use of vi −

Given below is the list advanced commands.

Word and Character Searching

The vi editor has two kinds of searches: **string** and **character**. For a string search, the **/** and **?** commands are used. When you start these commands, the command just typed will be shown on the last line of the screen, where you type the particular string to look for.

These two commands differ only in the direction where the search takes place −

- The **/** command searches forwards (downwards) in the file.

- The **?** command searches backwards (upwards) in the file.

The **n** and **N** commands repeat the previous search command in the same or the opposite direction, respectively. Some characters have special meanings. These characters must be preceded by a backslash (\) to be included as part of the search expression.

S.No.	Character &Description
1	^ Searches at the beginning of the line (Use at the beginning of a search expression).
2	. Matches a single character.
3	* Matches zero or more of the previous character.
4	$ End of the line (Use at the end of the search expression).
5	[Starts a set of matching or non-matching expressions.
6	< This is put in an expression escaped with the backslash to find the ending or the beginning of a word.
7	> This helps see the '<' character description above.

Set Commands

You can change the look and feel of your vi screen using the following: set commands. Once you are in the command mode, type: set followed by any of the following commands.

S.No.	Command & Description
1	**:set ic** Ignores the case when searching
2	**:set ai** Sets autoindent
3	**:set noai** Unsets autoindent
4	**:set nu** Displays lines with line numbers on the left side
5	**:set sw** Sets the width of a software tabstop. For example, you would set a shift width of 4 with this command — **:set sw = 4**
6	**:set ws** If *wrapscan* is set, and the word is not found at the bottom of the file, it will try searching for it at the beginning
7	**:set wm** If this option has a value greater than zero, the editor will automatically "word wrap". For example, to set the wrap margin to two characters, you would type this: **:set wm = 2**

8	**:set ro** Changes file type to "read only"
9	**:set term** Prints terminal type
10	**:set bf** Discards control characters from input

Opening multiple files with VI/VIM editor

To open multiple files, command would be same as is for a single file; we just add the file name for second file as well.

$ vi file1 file2 file 3

Now to browse to next file, we can use

$:n

or we can also use

$:e filename

Run external commands inside the editor

$:! command

An example for this would be

$:! df -H

Searching for a pattern

To search for a word or pattern in the text file, we use following two commands in command mode,

- command '/' searches the pattern in forward direction
- command '?' searched the pattern in backward direction

Both of these commands are used for same purpose, only difference being the direction they search in. An example would be,

$:/ search pattern (If at beginning of the file)

$ search pattern (If at the end of the file)

Searching & replacing a pattern

We might be required to search & replace a word or a pattern from our text files. So rather than finding the occurrence of word from whole text file & replace it, we can issue a command from the command mode to replace the word automatically. Syntax for using search & replacement is,

$:s/pattern_to_be_found/New_pattern/g

Suppose we want to find word “alpha” & replace it with word “beta”, the command would be

$:s/alpha/beta/g

If we want to only replace the first occurrence of word “alpha”, then the command would be

$:s/alpha/beta/

Using Set commands

We can also customize the behaviour, the and feel of the vi/vim editor by using the set command. Here is a list of some options that can be use set command to modify the behaviour of vi/vim editor,

$:set ic ignores cases while searching

$:set smartcase enforce case sensitive search

$:set nu display line number at the begining of the line

$:set hlsearch highlights the matching words

$: set ro change the file type to read only

$: set term prints the terminal type

$: set ai sets auto-indent

$:set noai unsets the auto-indent

Some other commands to modify vi editors are,

$:colorscheme its used to change the color scheme for the editor. (for VIM editor only)

$:syntax on will turn on the color syntax for .xml, .html files etc. (for VIM editor only)

WEEK-14

1. **Linux/Unix Backup Tool**
   - **Backup through tar / cpio / dd commands**
   - **Recovering single / multiple files**

Using cpio:

1. Using cpio to create a file archive on a tape device:

   # find . -print |cpio -ocBv /dev/rmt0

2. Using cpio to list the entries in a file archive on a tape device:

   # cpio -itcvB < /dev/rmt0

3. Using cpio to retrieve a file from a tape device:

   # cpio -icvdBum file.name < /dev/rmt0

   We can also use cpio to copy directory structure.

   For example copy Directory structure from current path to /export/home/tariq

   # find . -print|cpio -pmdv /export/home/tariq

Using tar:

1. Using tar to create a file archive on a tape device:

   # tar -cvf /dev/rmt0 file.name

   or

   # tar -cvf /dev/rmt0 .

2. Or for multiple directory hierarchies

   # tar -crvf my.tar `find /tmp/junk -type f` `find /var/tmp -type f`

3. Using tar to list the entries in a file archive on a tape device:

   # tar -tvf /dev/rmt0

4. Using tar to retrieve a file from a tape device:

   # tar -xvf /dev/rmt0 file.name

5. Copy files from current directory to directory /to/dst/dir/, keeping ownership and rights

#tar cvf - . | (cd /to/dst/dir/; tar xvf -)

To make a dump of root filesystem on tape device /dev/nrsa0.

Note that this is a non-rewinding device. See example below.

/sbin/dump -0ua -f /dev/nrsa0 /

To interactively restore a backup

/sbin/restore -i -f /dev/nrsa0 //Everything will be restored in current directory.

Using mt command with dump and restore

mt (magnetic tape manipulating program) is a very useful command specialy if you are using dump and restore combination.

Following are some useful options of mt command.

- ✓ # mt status Print status information about the tape unit.
- ✓ # mt rewind Rewind the tape.
- ✓ # mt erase Erase the tape.
- ✓ # mt retension Re-tension the tape (one full wind forth and back.
- ✓ # mt fsf 1 Forward space count by one file. One can be any number.

-f option can be used with mt to specify the different device.

mt -f /dev/rmt/1n fsf 3

WEEK – 15

Software Package Management in Linux.

a. **Redhat Tools – rpm and yum**
b. **Debian Tools – dpkg and apt-get**

1. Installing a package from a compiled (*.deb or *.rpm) file

The downside of this installation method is that no dependency resolution is provided. You will most likely choose to install a package from a compiled file when such package is not available in the distribution's repositories and therefore cannot be downloaded and installed through a high-level tool. Since low-level tools do not perform dependency resolution, they will exit with an error if we try to install a package with unmet dependencies.

```
# dpkg -i file.deb            [Debian and derivative]

# rpm -i file.rpm             [CentOS / openSUSE]
```

Note: Do not attempt to install on CentOS a *.rpm file that was built for openSUSE, or vice-versa!

2. Upgrading a package from a compiled file

Again, you will only upgrade an installed package manually when it is not available in the central repositories.

```
# dpkg -i file.deb            [Debian and derivative]

# rpm -U file.rpm             [CentOS / openSUSE]
```

3. Listing installed packages

When you first get your hands on an already working system, chances are you'll want to know what packages are installed.

```
# dpkg -l              [Debian and derivative]

# rpm -qa              [CentOS / openSUSE]
```

If you want to know whether a specific package is installed, you can pipe the output of the above commands to grep.

```
# dpkg -l | grep mysql-common
```

Another way to determine if a package is installed.

```
# dpkg --status package_name          [Debian and derivative]

# rpm -q package_name                 [CentOS / openSUSE]
```

For example, let's find out whether package sysdig is installed on our system.

```
# rpm -qa | grep sysdig
```

2. Finding out which package installed a file.

```
3.  # dpkg --search file_name

4.  # rpm -qf file_name
```

For example, which package installed pw_dict.hwm?

```
# rpm -qf /usr/share/cracklib/pw_dict.hwm
```

Common Usage of High-Level Tools

The most frequent tasks that you will do with high level tools are as follows.

1. Searching for a package

aptitude update will update the list of available packages, and aptitude search will perform the actual search for package_name.

```
# aptitude update && aptitude search package_name
```

In the search all option, yum will search for package_name not only in package names, but also in package descriptions.

```
# yum search package_name

# yum search all package_name

# yum whatprovides "*/package_name"
```

Let's supposed we need a file whose name is sysdig. To know that package we will have to install, let's run.

```
# yum whatprovides "*/sysdig"
```

what provides tells yum to search the package the will provide a file that matches the above regular expression.

```
# zypper refresh && zypper search package_name                [On openSUSE]
```

2. Installing a package from a repository

While installing a package, you may be prompted to confirm the installation after the package manager has resolved all dependencies. Note that running update or refresh (according to the package manager being used) is not strictly necessary, but keeping installed packages up to date is a good sysadmin practice for security and dependency reasons.

```
# aptitude update && aptitude install package_name          [Debian and
derivatives]

# yum update && yum install package_name                    [CentOS]

# zypper refresh && zypper install package_name             [openSUSE]
```

3. Removing a package

The option remove will uninstall the package but leaving configuration files intact, whereas purge will erase every trace of the program from your system.
aptitude remove / purge package_name
yum erase package_name

```
---Notice the minus sign in front of the package that will be uninstalled,
openSUSE ---

# zypper remove -package_name
```

4. Displaying information about a package

The following command will display information about the birthday package.

```
# aptitude show birthday

# yum info birthday

# zypper info birthday
```

WEEK – 16

LAB INTERNAL – 2