SRI VENKATESWARA COLLEGE OF ENGINEERING AND TECHNOLOGY
(Autonomous)
Unix & Shell Programming Lab Manual

Subject Name & Code	**:**	**UNIX & Shell Programming Lab (12ACS06)**
Year & Programme	**:**	**II Year, 2013-14**
Branch / Semester	**:**	**CSE / I Semester**

Week1

Session-1
a)Log into the system
Sol : Login
b)Use vi editor to create a file called myfile.txt which contains some text.
Sol : Vi mytable
c)correct typing errors during creation.
Sol: Practice vi editor commands
d)Save the file
Sol: :wq + Enter
e)logout of the system
Sol: logout

Note… Make Use of following commands:

To Get Into and Out Of vi

To Start `vi`

To use `vi` on a file, type in `vi filename`. If the file named `filename` exists, then the first page (or screen) of the file will be displayed; if the file does not exist, then an empty file and screen are created into which you may enter text.

*	**`vi filename`**	*edit `filename` starting at line 1*
	`vi -r filename`	*recover `filename` that was being edited when system crashed*

To Exit `vi`

Usually the new or modified file is saved when you leave `vi`. However, it is also possible to quit `vi` without saving the file.
Note: The cursor moves to bottom of screen whenever a colon (`:`) is typed. This type of command is completed by hitting the `<Return>` (or `<Enter>`) key.

*	**:x<Return>**	*quit* `vi`*, writing out modified file to file named in original invocation*
	:wq<Return>	*quit* `vi`*, writing out modified file to file named in original invocation*
	:q<Return>	*quit (or exit)* `vi`
*	**:q!<Return>**	*quit* `vi` *even though latest changes have not been saved for this* `vi` *call*

Session-2
a)Log into the system
b)open the file created in session 1
c)Add some text
d)Change some text
e)Delete some text
f)Save the Changes
Sol : Practice the commands in Vi editor
g)Logout of the system

Note… Make Use of following commands

Inserting or Adding Text

The following commands allow you to insert and add text. Each of these commands puts the `vi` editor into insert mode; thus, the `<Esc>` key must be pressed to terminate the entry of text and to put the `vi` editor back into command mode.

*	**i**	*insert text before cursor, until* `<Esc>` *hit*
	I	*insert text at beginning of current line, until* `<Esc>` *hit*
*	**a**	*append text after cursor, until* `<Esc>` *hit*
	A	*append text to end of current line, until* `<Esc>` *hit*
*	**o**	*open and put text in a new line below current line, until* `<Esc>` *hit*
*	**O**	*open and put text in a new line above current line, until* `<Esc>` *hit*

Changing Text

The following commands allow you to modify text.

*	**r**	*replace single character under cursor (no* `<Esc>` *needed)*

	R	*replace characters, starting with current cursor position, until* `<Esc>` *hit*
	cw	*change the current word with new text,* *starting with the character under cursor, until* `<Esc>` *hit*
	cNw	*change* `N` *words beginning with character under cursor, until* `<Esc>` *hit;* *e.g.,* `c5w` *changes 5 words*
	C	*change (replace) the characters in the current line, until* `<Esc>` *hit*
	cc	*change (replace) the entire current line, stopping when* `<Esc>` *is hit*
	Ncc *or* cNc	*change (replace) the next N lines, starting with the current line,* *stopping when* `<Esc>` *is hit*

Deleting Text

The following commands allow you to delete text.

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

Week2

a)Log into the system
b)Use the cat command to create a file containing the following data. Call it mytable use tabs to separate the fields.

1425	Ravi	15.65
4320	Ramu	26.27
6830	Sita	36.15
1450	Raju	21.86

Sol: cat > mytable

1425	Ravi	15.65
4320	Ramu	26.27
6830	Sita	36.15
1450	Raju	21.86

c)Use the cat command to display the file, mytable.
Sol: $cat mytable

1425	Ravi	15.65
4320	Ramu	26.27
6830	Sita	36.15
1450	Raju	21.86

d) Use the vi command to correct any errors in the file, mytable.
Sol: Verify the file with Vi editor Commannds
e) Use the sort command to sort the file mytable according to the first field. Call the sorted file my table
(same name)
Sol: $sort +1 mytable > mytable
f) Print the file mytable
Sol: cat mytable

1425	Ravi	15.65
1450	Raju	21.86
4320	Ramu	26.27
6830	Sita	36.15

g) Use the cut and paste commands to swap fields 2 and 3 of mytable. Call it my table (same name)
Sol: $cut -f1 > mytab1
$ cut –f 2 > mytab 2
$cut –f 3 > my tab3
$paste mytab3 mytab2 > mytab4
$paste mytab1 mytab4 > mytable
h)Print the new file, mytable
Sol: $ cat mytable

1425	15.65	Ravi
1450	21.86	Raju
4320	26.27	Ramu
6830	36.15	Sita

i)Logout of the system.

Note… Make Use of following commands

Cat:----

cat	*to display a text file or to con**cat**enate files*	
	cat file1	displays contents of `file1` on the screen (or window) without any screen breaks.
	cat file1 file2	displays contents of `file1` followed by `file2` on the screen (or window) without any screen breaks.
	cat file1 file2 > file3	creates `file3` containing `file1` followed by `file2`

Sort :----
The "`sort`" command sorts information piped into it. There are several options that let you sort information in a variety of ways.

```
ps -ef | sort
```

The most important options in Sort :

The following list describes the options and their arguments that may be used to control how **sort** functions.

- **-** Forces **sort** to read from the standard input. *Useful for reading from pipes and files simultaneously.*
- **-c** Verifies that the input is sorted according to the other options specified on the command line. If the input is sorted correctly then no output is provided. If the input is not sorted then **sort** informs you of the situation. The message resembles this.
    - `sort: disorder: This line not in sorted order.`
-
- `-m` Merges the sorted input. **sort** assumes the input is already sorted. **sort** normally merges input as it sorts. This option informs **sort** that the input is already sorted, thus **sort** runs much faster.
- **-o** *output* Sends the output to file *output* instead of the standard output. The *output* file may be the same name as one of the input files.
- **-u** Suppress all but one occurrence of matching keys. Normally, the entire line is the key. If field or character keys are specified, then the suppressing is done based on the keys.
- **-y *kmem*** Use *kmem* kilobytes of main memory to initially start the sorting. If more memory is needed, **sort** automatically requests it from the operating system.

The amount of memory allocated for the sort impacts the speed of the sort significantly. ***If no kmem is specified, sort starts with the default amount of memory (usually 32K).*** The maximum (usually 1 Megabyte) amount of memory may be allocated if needed. If 0 is specified for *kmem*, the minimum (usually 16K) amount of memory is allocated.

- **-z** *recsz* Specifies the record size used to store each line. Normally the *recsz* is set to the longest line read during the sort phase. If the -c or -m options are specified, the sort phase is not performed and thus the record size defaults to a system size. If this default size is not large enough, **sort** may abort during the merge phase. To alleviate this problem you can specify a *recsz* that will allow the merge phase to run without aborting.

Week3

Session1:

a)Login to the system

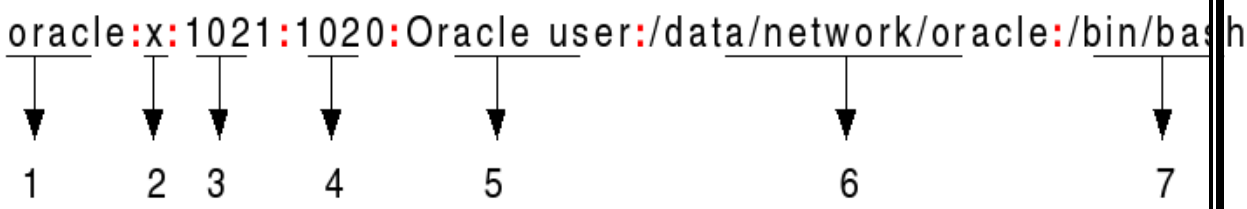
b)Use the appropriate command to determine your login shell

Sol: $echo $SHELL

sh

c)Use the /etc/passwd file to verify the result of step b.

Sol: $cat /etc/passwd



d)Use the who command and redirect the result to a file called myfile1. Use the more command to see the contents of myfile1.

Sol : $who > myfile1 | more

User1 pts/0 Apr 23 10:43

User2 pts/1 May 6 18:19

e)Use the date and who commands in sequence (in one line) such that the output of date will display on the screen and the output of who will be redirected to a file called myfile2. Use the more command to check the contents of myfile2.

Sol: $ date ; who > myfile2

Fri Aug 9 16:47:32 IST 2008

Cat myfile2 :

User3 pts/2 Apr 25 10:43

User4 pts/3 May 8 18:19

Note… Make Use of following commands:

Who :---

The "who" command lets you display the users that are currently logged into your Unix computer system.

who
This is the basic who command with no command-line arguments. It shows the names of users that are currently logged in, and may also show the terminal they're logged in on, and the time they logged in.

who | more

In this example the output of the who command is piped into the more command. This is useful when there are a lot of users logged into your computer system, and part of the output of the who command scrolls off the screen. See the more command for more examples.

```
who -a
```

The -a argument lists all available output of the who command for each user.

Piping:---

To connect the output of the one command directly to the input of the other command. This is exactly what pipes do. The symbol for a pipe is the vertical bar |

For example, typing

```
% who | sort
```

will give the same result as above, but quicker and cleaner.

To find out how many users are logged on, type

```
% who | wc -l
```

Session 2:

Input File : file1.dat :
Unix is Multiuser OS
Unix was developed by Brian Kernighan and KenThomson

a)Write a sed command that deletes the first character in each line in a file.
Sol: **sed** 's/^./ /' file1.dat
nix is Multiuser OS
nix was developed by Brian Kernighan and KenThomson

b)Write a sed command that deletes the last character in each line in a file.
Sol: sed '$s/.$//' file1.dat
Unix is Multiuser O
Unix was developed by Brian Kernighan and KenThomso

c)Write a sed command that swaps the first and second words in each line in a file.

```
sed -e 's/\([^ ]\+\)    *\([^ ]\+\)/\2 \1/'
```

sed 's/\([a-z]*\) \([a-z]*\)/\2 \1/' (Modified & working)

(Substrings enclosed with "\(" and "\)" can be referenced with "\n" (n is a digit from 1 to 9))

Note : Make use of following Link to know more about sed

Ref : http://www.grymoire.com/Unix/Sed.html#uh-0

Week4

a)Pipe your /etc/passwd file to awk, and print out the home directory of each user.
Sol: cat /etc/passwd | awk ' { print $7}'
b)Develop an interactive grep script that asks for a word and a file name and then tells how many lines
contain that word.
Sol:

```
echo  "Enter a word"
read word
echo "Enter the filename"
read file
nol=grep  -c  $word  $file
echo " $nol  times  $word present in  the $file"
```

c)Part using awk
Sol:

```
echo  "Enter a word"
read word
echo "Enter the filename"
read file
nol=awk '/$word/  { print NR }'  Infile
echo " $nol  times  $word present in  the $file"
```

Note… Make Use of following commands:

Grep: ---grep is one of many standard UNIX utilities. It searches files for specified words or patterns. First clear the screen, then type

```
% grep science science.txt
```

As you can see, **grep** has printed out each line containg the word **science**.

Or has it ????

Try typing

```
% grep Science science.txt
```

The **grep** command is case sensitive; it distinguishes between Science and science.

To ignore upper/lower case distinctions, use the -i option, i.e. type

```
% grep -i science science.txt
```

To search for a phrase or pattern, you must enclose it in single quotes (the apostrophe symbol). For example to search for spinning top, type

```
% grep -i 'spinning top' science.txt
```

Some of the other options of grep are:

-v display those lines that do NOT match
-n precede each matching line with the line number
-c print only the total count of matched lines

Try some of them and see the different results. Don't forget, you can use more than one option at a time. For example, the number of lines without the words science or Science is

```
% grep -ivc science science.txt
```

Note: Make use of Following Link to know about Awk

Ref : http://www.grymoire.com/Unix/Awk.html

Week5

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

```
echo " enter file"
read str
if test -f $str
then echo "file exists n it is an ordinary file"
elif test -d $str
then echo "directory file"
else
echo "not exists"
fi
if test -c $str
then echo "character device files"
fi
```

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

```
# get filename
echo -n "Enter File Name : "
read fileName

# make sure file exits for reading
if [ ! -f $fileName ]
then
echo "Filename $fileName does not exists"
exit 1
fi

# convert uppercase to lowercase using tr command
tr '[A-Z]' '[a-z]' < $fileName
```

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

```
echo “enter the login of the user”
read name
logindetails=`who|grep –w “$name” | grep “tty”
if [ $? –ne 0 ]
then
echo “$name has not logged in yet”
exit
fi
```

```
loginhours=`echo “$logindetails” | cut –c 26,27`
loginminuts=`echo “$logindetails” | cut –c 29-30`
hoursnow=’date | cut –c 12,13`
minnow =`date | cut –c 15,16`
hour=`expr  $loginhours - $hoursnow`
min=`expr $loginminuts - $minnow`
echo  “ $name is working since $hour Hrs - $min Minuts”
```

Week6

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

```
If [ $#  -ne  3 ]
then
echo  “chech the arguments once”
lastline=’wc –l < $1’
if [ $2 –lt $lastline –a $3  -le $lastline ]
then
nline=’expr $3 -$2 + 1’
echo “’tail +$2 $1 | head -$nline’”
else
echo “invalid range specification”
fi
fi
```

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

```
if [ $# -lt 1]
then
echo  “ Chech the arguments once”
exit
fi
echo “Enter a word”
read word
for file in $*
do
grep –iv “$word” $file | tee 1> /dev/null
done
echo   “ lines containing given word are deleted”
```

Week7

a)Write a shell script that computes the gross salary of a employee according to the following rules:

i)If basic salary is < 1500 then HRA =10% of the basic and DA =90% of the basic.

ii)If basic salary is >=1500 then HRA =Rs500 and DA=98% of the basic

The basic salary is entered interactively through the key board.

Sol:

```
echo enter basic salary
read sal
a=0.1
b=0.8
echo $a
echo "hra is"
hra=`echo 0.1 \* $sal|bc`
echo da is
da=`echo 0.8\*$sal|bc`
gsal=’expr $hra + $da + $sal’
echo $gsal
```

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

Sol:

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

Week8

a)Write an interactive file-handling shell program. Let it offer the user the choice of copying, removing, renaming, or linking files. Once the user has made a choice, have the program ask the user for the necessary information, such as the file name, new name and so on.
b)Write shell script that takes a login name as command – line argument and reports when that person logs in
Sol:

```
#Shell script that takes loginname as command line arg and reports when that person logs in.
if [ $# -lt 1 ]
then
echo improper usage
echo correct usage is: $0 username
exit
fi
logname=$1
while true
do
who|grep "$logname">/dev/null
if [ $? = 0 ]
then
echo $logname has logged in
echo "$logname">>sh01log.txt
date >>sh01log.txt
echo "Hi" > mesg.txt
echo "$logname" >> mesg.txt
echo "Have a Good Day" >> mesg.txt
mail "$logname" < mesg.txt
exit
else
sleep 60 fi  done
```

c)Write a shell script which receives two file names as arguments. It should check whether the two file contents are same or not. If they are same then second file should be deleted.
Sol:

```
echo  “enter first file name”
read  file1
echo “ enter second file name”
read file2
cmp file1 file2  >  file3
if  [   -z  $file1 ]          rm file2
fi
echo  “duplicate file deleted successfully”
```

Week9

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

Sol:

ls –l | grep "^.rwx" | cut –f 9

b)Develop an interactive script that ask for a word and a file name and then tells how many times that word occurred in the file.

c)Write a shell script to perform the following string operations:

i)To extract a sub-string from a given string.

ii)To find the length of a given string.

Note: Make use of Following Link to know about Shell Programming

Ref : http://www.freeos.com/guides/lsst/ch02.html

Week10

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

i)File type

ii)Number of links

iii)Read, write and execute permissions

iv)Time of last access

Sol:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<fcntl.h>
void main()
{       int fd;
struct stat buf;
fd=open("f5.txt",O_RDONLY|O_CREAT,600);
if(fd!=-1)
{       if(fstat(fd,&buf)==0)
{       printf("mode of fileis %u",buf.st_mode);
printf("\n size of the file is %u",buf.st_size);
printf("\n device name %u",buf.st_dev);
printf("\n inode of file  is %u",buf.st_ino);
printf("\n no. of links are %u",buf.st_nlink);
printf("\n  owner oof a file  is %u",buf.st_uid);
printf("\n no.of blocks is %u",buf.st_blocks);
printf("\n group owner  is %u",buf.st_gid);
printf("\n blocks size of the file is %u",buf.st_blksize);
printf("\n time of last modifiedis %u",buf.st_ctime);

}
else
printf("error in fstat() syscall");
}
else
printf("error in open() sys call");
}
```

Note Make Use of following Description on stat system call

Stat : --

NAME

stat - get file status

SYNOPSIS

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

int stat(const char *restrict path, struct stat *restrict buf);
```

DESCRIPTION

The *stat*() function shall obtain information about the named file and write it to the area pointed to by the *buf* argument. The *path* argument points to a pathname naming a file. Read, write, or execute permission of the named file is not required. An implementation that provides additional or alternate file access control mechanisms may, under implementation-defined conditions, cause *stat*() to fail. In particular, the system may deny the existence of the file specified by *path*.

If the named file is a symbolic link, the *stat*() function shall continue pathname resolution using the contents of the symbolic link, and shall return information pertaining to the resulting file if the file exists.

The *buf* argument is a pointer to a **stat** structure, as defined in the *<sys/stat.h>* header, into which information is placed concerning the file.

The *stat*() function shall update any time-related fields (as described in the Base Definitions volume of IEEE Std 1003.1-2001, Section 4.7, File Times Update), before writing into the **stat** structure.

Unless otherwise specified, the structure members *st_mode*, *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atime*, *st_ctime*, and *st_mtime* shall have meaningful values for all file types defined in this volume of IEEE Std 1003.1-2001. The value of the member *st_nlink* shall be set to the number of links to the file.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, -1 shall be returned and *errno* set to indicate the error

<sys/stat.h> : ---

The **stat** structure shall contain at least the following members:

```
dev_t     st_dev     Device ID of device containing file.
ino_t     st_ino     File serial number.
mode_t    st_mode    Mode of file (see below).
nlink_t   st_nlink   Number of hard links to the file.
uid_t     st_uid     User ID of file.
gid_t     st_gid     Group ID of file.
dev_t     st_rdev    Device ID (if file is character or block
special).
off_t     st_size    For regular files, the file size in bytes.
For symbolic links, the length in bytes of the
pathname contained in the symbolic link.
```

For a shared memory object, the length in bytes.
For a typed memory object, the length in bytes.
For other file types, the use of this field is unspecified.

time_t	st_atime	Time of last access.
time_t	st_mtime	Time of last data modification.
time_t	st_ctime	Time of last status change.
blksize_t	st_blksize	A file system-specific preferred I/O block size for this object. In some file system types, this may vary from file to file.
blkcnt_t	st_blocks	Number of blocks allocated for this object.

Week11

Write C programs that simulate the following unix commands:
a)mv
Sol:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
void main()
{
int fd1,fd2;
char buf[60];
char *p="/f2";
fd1=open("f2",O_RDWR);
fd2=open("f6",O_RDWR);
read(fd1,buf,sizeof(buf));
write(fd2,buf,sizeof(buf));
remove(p);
}
```

b)cp
Sol:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
void main()
{
int fd1,fd2;
char buf[60];
fd1=open("f2",O_RDWR);
fd2=open("f6",O_RDWR);
read(fd1,buf,sizeof(buf));
write(fd2,buf,sizeof(buf));
close(fd1);
close(fd2);
}
```

Note Make Use of following Description on File related system calls

System calls for File Processing :----

FreeBSD (4.4) has six file-related system calls. The following table briefly describe the function of each.

System calls	Function
open	open an existing file or create a new file
read	Read data from a file
write	Write data to a file
lseek	Move the read/write pointer to the specified location
close	Close an open file
unlink	Delete a file
chmod	Change the file protection attributes
stat	Read file information from inodes

Files to be included for file-related system calls.

```
#include    <unistd.h>
#include    <fcntl.h>
#include    <sys/types.h>
#include    <sys/uio.h>
#include    <sys/stat.h>
```

Open files

The ***open*** system call can be used to open an existing file or to create a new file if it does not exist already. The syntax of *open* has two forms:

*int open(const char *path, int flags); and*
*int open(const char *path, int flags, mode_t modes);*

The first form is normally used to open an existing file, and the second form to open a file and to create a file if it does not exist already. Both forms returns an integer called the *file descriptor*. The file descriptor will be used for reading from and writing to the file. If the file cannot be opened or created, it returns -1. The first parameter *path* in both forms sPecifies the file name to be opened or created. The second parameter (*flags*) specifies how the file may be used. The following list some commonly used flag values.

Flag	Description
O_RDONLY	open for reading only
O_WRONLY	open for writing only
O_RDWR	open for reading and writing
O_NONBLOCK	do not block on open
O_APPEND	append on each write
O_CREAT	create file if it does not exist
O_TRUNC	truncate size to 0
O_EXCL	error if create and file exists
O_SHLOCK	atomically obtain a shared lock
O_EXLOCK	atomically obtain an exclusive lock
O_DIRECT	eliminate or reduce cache effects
O_FSYNC	synchronous writes
O_NOFOLLOW	do not follow symlinks

The flag (*O_CREAT*) may be used to create the file if it does not exist. When this flag is used, the third parameter (*modes*) must be used to specify the file access permissions for the new file. Commonly used modes (or access permissions) include

Constant Name	Octal Value	Description
S_IRWXU	0000700	/* RWX mask for owner */
S_IRUSR	0000400	/* R for owner */
S_IWUSR	0000200	/* W for owner */
S_IXUSR	0000100	/* X for owner */
S_IRWXO	0000007	/* RWX mask for other */
S_IROTH	0000004	/* R for other */
S_IWOTH	0000002	/* W for other */
S_IXOTH	0000001	/* X for other */

R: read, W: write, and X: executable

For example, to open file "tmp.txt" in the current working directory for reading and writing:

fd = open("tmp.txt", O_RDWR);

To open "sample.txt" in the current working directory for appending or create it, if it does not exist, with read, write and execute permissions for owner only:

fd = open("tmp.txt", O_WRONLY|O_APPEND|O_CREAT, S_IRWXU);

A file may be opened or created outside the current working directory. In this case, an absolute path and relative path may prefix the file name. For example, to create a file in /tmp directory:

open("/tmp/tmp.txt", O_RDWR);

Read from files

The system call for reading from a file is ***read***. Its syntax is

*ssize_t read(int fd, void *buf, size_t nbytes);*

The first parameter *fd* is the file descriptor of the file you want to read from, it is normally returned from *open*. The second parameter *buf* is a pointer pointing the memory location where the input data should be stored. The last parameter *nbytes* specifies the maximum number of bytes you want to read. The system call returns the number of bytes it actually read, and normally this number is either smaller or equal to *nbytes*. The following segment of code reads up to 1024 bytes from file tmp.txt:

```
int actual_count = 0;
int fd = open("tmp.txt", O_RDONLY);
void *buf = (char*) malloc(1024);

actual_count = read(fd, buf, 1024);
```

Each file has a pointer, normally called read/write offset, indicating where next *read* will start from. This pointer is incremented by the number of bytes actually read by the *read* call. For the above example, if the offset was zero before the *read* and it actually read 1024 bytes, the offset will be 1024 when the *read* returns. This offset may be changed by the system call *lseek*, which will be covered shortly.

Write to files

The system call ***write*** is to write data to a file. Its syntax is

*ssize_t write(int fd, const void *buf, size_t nbytes);*

It writes *nbytes* of data to the file referenced by file descriptor *fd* from the buffer pointed by *buf*. The *write* starts at the position pointed by the offset of the file. Upon returning from *write*, the offset is advanced by the number of bytes which were successfully written. The function returns the number of bytes that were actually written, or it returns the value -1 if failed.

Week12

Write a C program that simulates ls Command

Sol:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<dirent.h>
void main()
{
DIR *dp;
struct dirent *dirp;
dp=opendir(".");
while(dirp=(readdir(dp)!=NULL))
{
if(dirp->d_ino==0)
continue;
else
printf("%s \n",dirp->d_name);
}
}
```

Note Make Use of following Description on directory related system calls

Opendir():----

NAME

opendir - open a directory

SYNOPSIS

```
#include <dirent.h>

DIR *opendir(const char *dirname);
```

DESCRIPTION

The *opendir*() function shall open a directory stream corresponding to the directory named by the *dirname* argument. The directory stream is positioned at the first entry. If the type **DIR** is implemented using a file descriptor, applications shall only be able to open up to a total of {OPEN_MAX} files and directories.

RETURN VALUE

Upon successful completion, *opendir*() shall return a pointer to an object of type **DIR**. Otherwise, a null pointer shall be returned and *errno* set to indicate the error.

Ref : http://www.opengroup.org/onlinepubs/009695399/functions/opendir.html

readdir():---

NAME

readdir, readdir_r - read a directory

SYNOPSIS

```
#include <dirent.h>

struct dirent *readdir(DIR *dirp);

int readdir_r(DIR *restrict dirp, struct dirent *restrict entry,
       struct dirent **restrict result);
```

DESCRIPTION

The type **DIR**, which is defined in the *<dirent.h>* header, represents a *directory stream*, which is an ordered sequence of all the directory entries in a particular directory. Directory entries represent files; files may be removed from a directory or added to a directory asynchronously to the operation of *readdir*().

The *readdir*() function shall return a pointer to a structure representing the directory entry at the current position in the directory stream specified by the argument *dirp*, and position the directory stream at the next entry. It shall return a null pointer upon reaching the end of the directory stream. The structure **dirent** defined in the *<dirent.h>* header describes a directory entry.

The *readdir*() function shall not return directory entries containing empty names. If entries for dot or dot-dot exist, one entry shall be returned for dot and one entry shall be returned for dot-dot; otherwise, they shall not be returned.

The pointer returned by *readdir*() points to data which may be overwritten by another call to *readdir*() on the same directory stream. This data is not overwritten by another call to *readdir*() on a different directory stream.

If a file is removed from or added to the directory after the most recent call to *opendir*() or *rewinddir*(), whether a subsequent call to *readdir*() returns an entry for that file is unspecified.

The *readdir*() function may buffer several directory entries per actual read operation; *readdir*() shall mark for update the *st_atime* field of the directory each time the directory is actually read.

After a call to *fork*(), either the parent or child (but not both) may continue processing the directory stream using *readdir*(), *rewinddir*(),or *seekdir*().If both the parent and child processes use these functions, the result is undefined.

If the entry names a symbolic link, the value of the *d_ino* member is unspecified.

The *readdir*() function need not be reentrant. A function that is not required to be reentrant is not required to be thread-safe.

The *readdir_r*() function shall initialize the **dirent** structure referenced by *entry* to represent the directory entry at the current position in the directory stream referred to by *dirp*, store a pointer to this structure at the location referenced by *result*, and position the directory stream at the next entry.

The storage pointed to by *entry* shall be large enough for a **dirent** with an array of **char** *d_name* members containing at least {NAME_MAX}+1 elements.

Upon successful return, the pointer returned at **result* shall have the same value as the argument *entry*. Upon reaching the end of the directory stream, this pointer shall have the value NULL.

The *readdir_r*() function shall not return directory entries containing empty names.

If a file is removed from or added to the directory after the most recent call to *opendir*() or *rewinddir*(), whether a subsequent call to *readdir_r*() returns an entry for that file is unspecified.

The *readdir_r*() function may buffer several directory entries per actual read operation; the *readdir_r*() function shall mark for update the *st_atime* field of the directory each time the directory is actually read. ⌫

Applications wishing to check for error situations should set *errno* to 0 before calling *readdir*(). If *errno* is set to non-zero on return, an error occurred.

RETURN VALUE

Upon successful completion, *readdir*() shall return a pointer to an object of type **struct dirent**. When an error is encountered, a null pointer shall be returned and *errno* shall be set to indicate the error. When the end of the directory is encountered, a null pointer shall be returned and *errno* is not changed.

If successful, the *readdir_r*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

<dirent.h> Structure :----

The <*dirent.h*> header shall define the following type:

DIR A type representing a directory stream.

It shall also define the structure **dirent** which shall include the following members:

```
ino_t  d_ino       File serial number.
char   d_name[]    Name of entry.
```

The type **ino_t** shall be defined as described in *<sys/types.h>*.

The character array *d_name* is of unspecified size, but the number of bytes preceding the terminating null byte shall not exceed {NAME_MAX}.

The following shall be declared as functions and may also be defined as macros. Function prototypes shall be provided.

```
int            closedir(DIR *);
DIR           *opendir(const char *);
struct dirent *readdir(DIR *);

int            readdir_r(DIR *restrict, struct dirent *restrict,
struct dirent **restrict);
void           rewinddir(DIR *);
[void          seekdir(DIR *, long);
long           telldir(DIR *);
```

Ref: http://www.opengroup.org/onlinepubs/009695399/functions/readdir.html