COURSE CONTENT

LINUX INTERNALS LABORATORY								
V Semester: CSE(DS)								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
ACDD13	**Core**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		0	0	2	1	40	60	100
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: 45**				**Total Classes: 45**		
Prerequisite: Programming for Problem Solving Laboratory								

I. COURSE OVERVIEW:

This course aims to provide a well-rounded introduction to Linux, operating systems, and software development in a Linux environment. Linux is the blend of innovative concepts required by its unique environment involving kernel concepts, basic commands, shell scripting, file processing, socket programming, processes and Inter process communication (IPC). This course equip individuals with the knowledge and skills needed to work effectively in a Linux-based environment, software development, system administration and security awareness.

II. COURSES OBJECTIVES:

The students will try to learn

I. The fundamental concepts of operating system including bourne shell (bash) with the Linux command line environment.
II. The shell programming using arithmetic operations, control structures and functions in shell scripts in vi editor.
III. The process management and inter-process communication used for exchanging data between multiple threads in one or more processes.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1 Demonstrate text processing utilities, file handling utilities, security by file permissions, process utilities, disk utilities and networking commands with different options available for solving problems.
CO 2 Make use of bourne shell constructs, decision structures and loops in designing programs for complex problems.
CO 3 Interpret to write, compile, debug and run C language program in Linux shell environment for implementing kernel level concepts.
CO 4 Identify basic methods and techniques used in solving simple programming tasks in the area of execution environment, processes signals and threads.
CO 5 Experiment with IPC mechanisms such as pipes, named pipes, shared memory, message queues, semaphores and sockets for inter-process communication.
CO 6 Choose the appropriate protocol such as TCP or UDP for effective communication in client-server applications.

IV. COURSE CONTENT:

No.	Topic Name	Page No.
1	Basic Exercises on Linux: 1. Help and man pages 2. Useful Commands/Utilities 3. Pipe and Input/output Redirection	4-6
2	Basic Exercises on Files in Linux: 1. File Handling Utilities 2. File/Directory Ownerships and Permissions 3. Disk utilities	6-10
3	Exercises on Process, Text, network and backup utilities-I 1. System and Application Software Directories 2. Processes utilities 3. Text Processing Utilities head:	10-14
4	Exercises on Process, Text, network and backup utilities-II 1. Hard Links and Symbolic Links 2. Network Utilities 3. Tape Archive (tar) and ZIP Compression (gzip, bzip2)	14-16
5	Exercises on Shell Programming-I 1. Displaying files 2. moving files 3. displaying logged in users	16-18
6	Exercises on Shell Programming-I 1. Wishing user based on time 2. Searching for specified word 3. Displaying between lines	18-20
7	Exercises on Shell Programming (Input, Decision and Loop)-I 1. Add2Integer (Input) 2. SumProductMinMax3 (Arithmetic & Min/Max) 3. Income Tax Calculator (Decision) 4. Pension Contribution Calculator with Sentinel (Decision & Loop)	20-24
8	Exercises on Shell Programming (Input, Decision and Loop)-II 1. Sales Tax Calculator (Decision & Loop) 2. Reverse Int (Loop with Modulus/Divide) 3. Amicable Numbers 4. Capricorn Number	24-26
9	Exercises on Simulating commands - I 1. Simulating CAT Command 2. Simulating CP Command 3. Simulating RM Command 4. Simulating LS Command	26-28
10	Exercises on Simulating commands - II 1. Simulating Head Command 2. Simulating Tail Command 3. Simulation of MV Command 4. Simulation of NL Command	28-31

11	Exercises on Signal Handling 1. Signal handler function with SIGINT 2. Signal handler function with SIGDFL 3. Signal handler function with SIGKILL	**31-33**
12	Exercises on Inter Process Communication (IPC) and Message Queues 1. One-Way Communication Using Pipe 2. One-Way Communication Using FIFO Function 3. Storing Messages In Message Queues (Sender) 4. Retrieving Messages From Message Queues (Receiver)	**33-36**
13	Exercises on shared memory	**36-37**
14	Exercises on Socket Programming 1. echo Client Server Program using TCP elementary functions 2. Receiver (Reading Messages from Queue)	**37-40**

V. COURSE CONTENT:

EXERCISES FOR LINUX INTERNALS LABORATORY

Note: Students are encouraged to bring their own laptops for laboratory practice sessions

Getting Started Exercises

1. Basic Exercises on Linux

1.1 Help and man pages

You can issue help command:

```
$ help                          // Display the help menu for the bash shell
$ help <command-name>  // Display the help menu for the command
```

Most of the commands also support an help option, but may exist in various style:

```
$ <command-name> -h        // Unix-style: dash followed by a single character
$ <command-name> -?
$ <command-name> -help     // X-style: dash followed by a keyword
$ <command-name> --help    // GNU-style: double-dash followed by a keyword
```

Unix systems provide so called man pages (or manual pages) for all the commands and utilities. To display the man page for a particular command, use man command:

```
$ man <command-name>          // Display manual page for the command
                              // You can use Up/Down/PgUp/PgDown keys to scroll the texts
$ info <command-name>
$ man <command-name> | less   // Display in page-mode
```

To search for commands:

```
$ man -k <keyword>      // Search for commands relevant to keyword
$ apropos <keyword>     // Same as above
```

1.2 Useful Commands/Utilities

These are the commands/utilities that a good Unix programmer is expected to know. Check the man pages ("man *command-name*" or google) to get the detailed description.

- File related:
  - **pwd**: Print current working directory. In bash shell, the current working directory is also shown in the command prompt.
  - **cd *pathname***: Change current working directory. The *pathname* could be either absolute or relative (to the current working directory). Special notations "." and ".." refer to the current and parent directories, respectively.
  - **ls**: List files (in short-format). "ls -l" lists file in long-format; "ls -a" lists also the hidden files.
  - **cat**: Concatenate files and print its content.
  - **less**, **more**: View file in pages.
  - **touch *filename***: Create the file if it does not exist; otherwise, update the last-modified timestamp.
  - **export *name=value***: Set a variable and export to global environment.
  - **top**: Print resource usage and top processes.
  - **hostname**: Print hostname.
  - **uptime**: Print how long the system has been running.
  - **date**: Print date/time.
- Utilities:
  - **which *program-name***: Print the location of the program-name.

    - **whereis** ***program-name***: List all files related to the program-name.
    - **whatis** ***program-name***: Print one-line description of program-name.
    - **locate** ***filename***: Search for files in local system.
    - **man** ***command-name***: Display manual pages for the command.
- Editors:
    - **vi/vim**, **nano**, **emacs**: Console-based (text-based) editors.
    - **gedit**: graphical text editor.
- Programming:
    - **make**: Install programs.
    - **gcc**, **g++**: GNU C/C++ compiler.
- More: diff, gzip, tar, ping, ssh, history, su, sudo, adduser, addgroup, etc.

More on cd (change directory) command

Read "Change Directory (cd) command" for basic usage.
You can use "cd *path*" to change the current working directory. The new path could be an absolute path, beginning with root "\" or home "~"; or relative to the current working directory (PWD).
In cd command, you can use "/" to denote the root directory, "~" to denote home directory of the current login user; ".." (double-dot) to refer to the parent directory; "." (single-dot) to refer to the current directory; and "-" (dash) to refer to the previous working directory (OLDPWD).
By default, in "cd *relative-path*", the new path is relative to the current working directory. Nonetheless, you can set the environment variable CDPATH to change the base. If CDPATH is not set, it is defaulted to current working directory. CDPATH could contain multiple directories separated by ":" (colon). For example,

```
$ cd                // home directory
$ pwd
/home/peter
$ mkdir local       // create a directory local (/home/peter/local)

$ export CDPATH=/usr    // set base for relative cd to /usr

$ cd local          // relative to CDPATH
/usr/local

$ export CDPATH=.:/usr  // set base for relative cd to current directory and /usr
$ cd                // home directory
$ pwd
/home/peter
$ cd local          // found local relative to current directory
/home/peter/local
```

1.3 Pipe and Input/Output Redirection

By default, the output of a command goes to the screen (called STDOUT), and the input of a command comes from the keyboard (called STDIN). You can use a *redirection operator* to redirect input and output from/to a file or another command:

- > (output redirection): Writes the output to a file (or a device such as printer), instead of the screen (STDOUT).
- >> (output append redirection): Appends the output to a file, instead of the screen.
- < (input redirection): Reads the input from a file or a device, instead of the keyboard (STDIN).
- | (pipe): Pipes the output of one command as the input of another command.
- tee: sends output to standard output and to file(s). Named after T-pipe, which splits water into two directions.

An output redirector '>' involves a program and a sink (destination). An input redirector '<' involves a

program and a source. A pipe '|' involves two programs.

Try:,

```
// Redirect the output of the ls command to a file, instead of screen
$ ls -l > listing.txt

// Pipe the output of ls command as the input of program less for view page-by-page
$ ls -l | less
// Pipe the output of ls command as the input of program wc to count the lines
$ ls -l | wc -l
// Pipe the output of ls command as the input of program grep to filter lines containing "xxxx"
$ ls -l | grep xxxx

// Pipe the output of ls command as the input of tee, which sends the output to standout and file "listing.txt"
$ ls -l | tee listing.txt
// Pipe the output of ls command as the input of grep.
// grep then sends its output to standout and files "listing1.txt" and "listing2.txt"
$ ls -l | grep xxxx | tee listing1.txt listing2.txt
```

2. Basic Exercises on Files in Linux

2.1 File Handling Utilities:

cp: ***The cp command copies a file or a group of files. It creates an exact image of the file on disk with a different name.***

```
Ex: $cp t1 t2
This will copy the contents of t1 into t2. If t2 does not exist, it will be created. However, if t2 already exists, then its contents are overwritten.
Ex: $cp -i t1 t2
cp: overwrite t2 (Yes/No)?
This will copy the contents of t1 into t2. If t2 does not exist, it will be created. However, if t2 already exists, then it warns the user before overwritten contents to a file. If "y" at this prompt overwrites the file, any other response leaves it uncopied.
```

mv:

The mv command renames files. It has two functions:

1. It renames a file (or directory).
2. It moves a group of files to a different directory.

Syn: $mv <sourcefilename> <renamefile>/<directoryname>. For example,

```
$ Ex: $mv s1 t1

Then s1 is renamed as t1.

Ex: $mv file1 file2 newdir

On execution of this command 'file1' and 'file2' are no longer present at their original location, but are moved to the directory 'newdir'.
```

rm: (Removing files): The rm command removes the given file or files supplied to it.
Syn: $rm options <filename(s)>

```
Ex: $rm –i file1
Where –i is a switch, removes file1 interactively; i.e. you are asked for confirmation before deleting the file
```

mkdir: The mkdir command is used to create a new directory

Syn:

$mkdir options <directoryname>

Ex: $mkdir book

The above command creates a directory named 'book'

Among the options available with mkdir is -p, which allows us to create multiple generations of directories specified in the given path too.

directory. Syn: $rmdir options <directoryname>

The rmdir is used to remove directories.

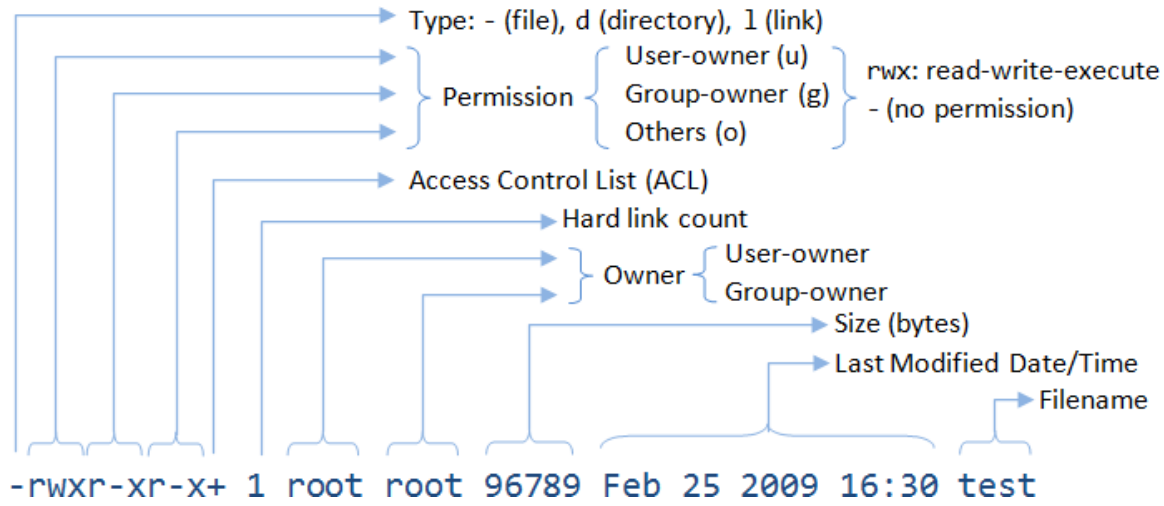
With '-p' option, it removes not only the specified directory but also its parent directories. However rmdir removes only the empty directories.

Ex: $ rmdir -p works/dir1/unix/book

It removes the directory book.

2.2 File/Directory Ownerships and Permissions

You can issue command "ls -l" to list files/directory in long format, which shows all the file attributes, e.g.,



For each entry:

- The first character indicates the type, with - for file and d for directory. The other type codes are l for link (symlink or hard link), b for block device, c for character device, p for named pipe, and s for socket.
- A file/directory has a user-owner and a group-owner, as indicated. The user-owner may or may not belong to the group.
- A file/directory has 3 permission settings, read/list (r), write (w) and execute/access (x), for user-owner (u), group-owner (g) and others (or the world) (o), respectively. The permissions are represented with 9 characters, in 3 groups of "rwx" for user-owner, group- owner and others, as shown in the above listing. "-" indicates absence of permission. For examples,

- -rwxr-x--- peter devel test.php
- // File,
- // filename is test.php,
- // user-owner is peter, group-owner is devel,

```
    // user-owner has read, write and execute permissions,
    // group-owner has read and execute permissions,
    // others (the world) have no permission
-rwx ------root root myconfig
    // File,
    // filename is myconfig,
    // user-owner is root, group-owner is root,
    // user-owner has read, write and execute permissions,
    // group-owner and others have no permission
drwxr-xr-x peter devel www
    // Directory,
    // directory name is www,
    // user-owner is peter, group-owner is devel,
    // user-owner has list, write and access permissions,
    // group-owner has list and access permissions,
// others (the world) have list and access permissions
```

File Permissions

A file is indicated by type of "-". For files:

- "r" (read) permits reading the file content.
- "w" (write) permits writing into the file.
- "x" (execute) indicates that the file is executable, i.e., a program file (or binary file).

Directory Permissions

A directory is indicated by type of "d". For directories, "r" shall be interpreted as list and "x" as access, as follows:

- "r" (list) permits listing of directory's contents (filenames and sub-directory names only) via listing command such as ls. Without "r" permission, you cannot issue "ls" command.
- "w" (write) permits writing into the directory, i.e., creating new files or sub-directories inside the directory.
- "x" (access) permits access into this directory (i.e., "cd" into the directory).

A directory holds two pieces of information for each file/sub-directory it contains: the filename/subdirectory-name and its inode number. I-node stores the attributes of the file/directory, including the disk block location. You can list the name and inode number via command "ls -i". For directories, "r" permission is needed to get the name; "x" permission is needed to get its inode number given the name, which is needed to enter the directory. For example, to issue "cat /home/peter/test/f1.txt", you need "x" permission for directories /, home, peter, test (so as to enter the directories); and "r" permission for file "f1.txt". No "r" permission is needed for the directories, as the names are known. "r" permission is needed for a directory for issuing ls command.
For production system, "x" is needed to enter (access) the directory. Very few programs need to list the directory contents. For development system, both "x" and "r" are needed to enter the directory and issue the ls command.

Change Mode (chmod)

chmod (change file mode) to change the file mode (i.e., permission), in the form of "ugoa±rwx" with "+" to add permission and "-" to remove; or "ugoa=rwx" to set the permission, where "u" for user-owner, "g" for group-owner, "o" for others, and "a" for all (i.e., u+g+o). For example,

```
// Grant executable mode for owner(u), group(g) for all *.sh files
// -c lists all the changes
$ chmod -c ug+x *.sh
// Remove write permission from group(g), others(o) for all *.txt files
$ chmod -c go-w *.txt
// All (u+g+o) read and execute
$ chmod -c a=rx *
```

You can also use three octal numbers (each for u, g and o) to represent the file permission. Each octal number carries 3 bits (xxx), corresponding to the read-write-execute permissions, where 4 (100B) for read,

2 (010B) for write and 1 (001B) for execute. E.g., "700" is equivalent to "rwx------"; "664" is equivalent to "rw-rw-r--"; "775"
is equivalent to "rwxrwxr-x". For example,

Change Owner/Group (chown, chgrp)

We can use command chown (change file owner/group) to change the owner of files, e.g.,

```
// Change owner for a file
$ chown peter test.txt
// Change owner of the current directory
$ chown peter .
// Also change the group to programmers for a directory
$ chown peter:programmers /myproject

// Change Group
$ chgrp programmers .
```

You can use option -R to recursively changing the files in the subdirectories.
In most systems, only superuser can run chown, not even the file owner. This is because Unix systems prevent users from "giving away" files (you can only chown to yourself, which does not require any chown command!)
The file owner can run the chgrp if he belongs to the target group (again, you cannot "give away" files).

umask

umask (filemode creation mask) controls the permissions for new files and directory created. Recall that permissions can be represented by 3 octal numbers, each representing rwx for u (user), g (group) and o (others) respectively, e.g., 660 for (rw- rw-); 775 for (rwx rwx r-x).
umask is also represented in 3 octal numbers, with 1 to *disable* a certain permission. For example, suppose the umask is 022 (--- -w- -w-), it will disables the w for group and others, when a new file/directory is created. In other words, a new file/directory could have (rwx r-x r-x), or lesser permissions; it will never have w for group and others.
Each program has its own umask. For "Terminal" application, you can set the umask in one of the login scripts (such as "~/.profile") with the following line:

```
umask 022
```

You can also issue the command umask to display/change the current umask setting.
(For apache, the umask can be set in /etc/apache2/envvars.)

setuid and setgid permission for executable-file

When you execute a program (having "x" file permission) with setuid (Set User ID upon execution) or setgid (Set Group ID upon execution), it takes on the file owner's (or group's) privilege. For example, if a setuid program is owned by root, and executable by all, it will be run in root privilege, even when invoked by a less privilege user. setuid and setgid can be dangerous! Use them with great care!
You can use chmod command to enable setuid/setgid:

2.3 Disk utilities

A disk utility is a utility program that allows a user to perform various functions on a computer disk, such as disk partitioning and logical volume management, as well as multiple smaller tasks such as changing drive letters and other mount points, renaming volumes, disk checking, and disk formatting, which are otherwise handled separately by multiple other built-in commands.

- **du**: Print disk file space usage.
- **df**: Report file system disk space usage.
- **ulimit:** It stands for 'user limit' and contains a value, which signifies the largest file that can be created by the user in the file system.
- **find:** You The find command is recursively examines a directory tree to look for file matching some criteria, and then takes some action on the selected files or locating files.

Syn: $find pathlist selection_criteria action. For example,

```
Ex: $ find -name "*.c"
  // Find all filename ending with .c, in current directory and subdirectories

$ find -iname "*.c"
  // option -i for case insensitive

$ find /usr -name "*.c"
  // Search /usr and its subdirectories
```

mount/unmount: A super user may extend the file system by using the 'mount' utility.
Syntax : mount –o options [device name directory]
unmount device name
The file system built on the floppy disk can be linked into the existing file system on the hard disk using the 'mount' command. Once mounted we can create files and directories in the new file system and treat it as a normal directory existing in a file system.

3. Exercises on Process, Text, network and backup utilities-I

Everything in Unix is a *file* - from data files, executable programs, to input and output devices. Files are organized in *directories* (aka *folders*). The directories are organized in a hierarchical *tree* structure, starting from the *root* directory, denoted by "/". There is only one root directory for the entire Unix's file system. A directory may contain sub-directories and files.
Unix is a *multi-user* system. Some files are used solely by the system; some are shared by all users; while some belong to a particular user.

`Disk Drives and Root Directory`
Windows systems has a concept of drive (e.g., C drive and D drive). Each drive begins with a root directory (e.g., `c:\`, `d:\`), resulting in multiple root directories.
Unix has no concept of drive and has a single root for the entire file system. The drives are mounted under the file system at a specific directory.

3.1 System and Application Software Directories

A Unix system has these directories for *system and application software* available to all users:

- /lib, /bin, /sbin: System libraries, binaries and superuser's binaries. Binaries are executable programs (having *executable* file attribute). Libraries are supporting codes for programs. There are two types of libraries, static libraries (.a for archives in Unix, .lib in Windows) and dynamic libraries (.so for shared objects in Unix, .dll for dynamic link libraries in Windows). Static library codes are linked into the program; whereas shared library codes are loaded during runtime and can be shared by many programs.
- /usr: Application Software (contrast to System Software in /bin, /lib, and /sbin). It contains sub-directories such as / usr / bin, / usr / lib, / usr / include, / usr/ share / man (man pages), / usr/ share/ doc (documentation) and etc.
- /usr/local: Locally installed application software which are not part of the distribution. It contains sub-directories such as /usr/lcoal/bin, /usr/local/lib, /usr/local/include, and etc.
- /opt: optional application software package.
- /etc: System-wide configuration files, such as fstab (file system table for disks, CD drives, and storage devices), passwd (list of users), and sudoers (list of users with superuser access). (It is called "et cetera" in early days to mean additional things to bin and lib.)
  - /var: Variable (changing) files, such as /var/logs (for log files), /var/spool (for printer spool files), /var/mail (for mail), /var/tmp, and etc.
  - /root: superuser's home directory.
  - /dev: file representation of devices. A special /dev/null (or the null device) is a special file that discards all data written to it but reports that the write operation succeeded. It provides no data to any process that reads from it, yielding EOF immediately.
  - /mnt: file system mount point for (fixed) hard disks, CD drive, etc.
  - /media: for removable media, such as external drive and USB drive.
  - /sys: real-time information on devices used by the kernel.

- /boot: boot loader files and Linux kernel.
- /proc: process information.
- /srv: services.
- /tmp: System's temporary directory.

3.2 Processes utilities

Unix is a multi-process, multi-user operating system. It supports many processes concurrently. You can use ps to list all the processes.

```
$ ps          // Print processes of current user
$ ps -e       // Print all processes
$ ps -ef      // Print all processes in full-listing
$ ps aux      // Same as above (in old BSD options)

// Search for processes
$ ps aux | grep mysqld   // Print MySQL server process
$ ps -ef | grep mysqld   // same as above
$ ps -ef | grep tomcat
$ ps -ef | grep $USER   // of current user, same as ps -f
```

[TODO] pid, Sample full-listing of ps

To terminal a process, you can issue kill command with the process ID, or job ID.

```
$ kill pid      // Kill a particular possess with the given processID
$ kill -9 pid   // Force kill
```

3.3 Text Processing Utilities

head: Displaying the beginning

Ex: $head -10 newfile
The first 10 lines of newfile are displayed.

tail: Displaying the end of a file.

The *tail* command is the counter part of the head command, displays the end of the file. By default *head* and *tail* commands display first and last 10 lines of a file respectively.

Ex: $tail -3 emp
It displays the last 3 lines in emp file.
The disadvantage of head and tail is that they cannot display a range of lines.

sort: This command can be used for sorting the contents of a file. Apart form sorting files, sort can merge multiple sorted files and store the result in the specified output file. While sorting the sort command bases its comparisons on the first character in each line in the file. If the first character of two lines is same then the second character in each line is compared and so on. It sorts the spaces the tabs first, then the punctuation marks followed by numbers, uppercase letters and lowercase letters in that order.

This simplest form of sort command would be
$sort file1
This would sort the contents of file1 and display the sorted output on the screen.

nl: line numbering:This is for numbering the lines.

Example:
$nl file1
1 chairman
2 special officer
3 principal
4 head
5 lecturer

uniq: locating repeated lines:

Ex:$uniq mp1
Uniq simply fetches one copy of each record and writes it to the standard output.

grep - Print lines matching the pattern

You can use grep to find lines matching a search pattern. The syntax is:

$ grep [*options*] *pattern* [*file*...]
`for Ex:` $ grep search-word filename
$ grep search-word file1 file2 file3
$ grep -r search-word directoryname `// -r` (or -R) for recursive

cut:: Like sort, cut is also a filter. It cuts or picks up a given number of character or fields from the specified file cut is slitting a file vertically cut identifies both columns and fields.

Ex: a) $cut –c 5-10 16-20 file1

Displays the columns from 5 to 10 and 16 to 20 of file1

b) $cut –c -3 5-10 16-20 30- file1

The expression 30- indicates column number 30 to the end of the line. Similarly, -3 is the same as 1-3.

Files often don't contain fixed length records, in which case, it is better to cut fields rather than columns. Two options need to be used here –d (delimiter) for the field delimiter and –f (field) for specifying the field list.

paste: pasting files:
It is a special type of 'concatenation' in that it pastes files vertically rather than horizontally.

Ex:$ paste file1 file2

Then the fields of file2 are concatenated to fields of file1. By default the delimiter is tab. The delimiter of our choice is set by '-d'(delimiter) option.

$cut –d "|" –f 1,4 –shortlist | paste –d "|" –cutlist

Join: A command that extracts common lines from two sorted files.

Syntax : join [options] filename1 filename2
One line of o/p is created for each line in the two files that match, based on option.
This command joins the common lines found in filename1 and filename2; if filename1 is not specified, *join* reads from the standard input.

wc: It counts the number of lines, words and characters in the specified file or files. It comes with the options *-l, -w* and *–c* which allow the user to obtain the number of lines, words or characters individually or in any desired combination.

Ex: $wc –lc file1 file2

file1 20 571

file2 30 804

The file file1 constitutes 20 lines and 571 characters similarly for the file2.

The wc command is capable of accepting input directly from the keyboard. By entering wc without any arguments, it waits for the user to type in the input. On terminating input (using ctrl d), the appropriate counts are displayed for the input.

Note: Filters are: *cat, pg, more, head, tail, grep, sort, wc, nl, uniq, cut, paste* etc.

tee:

Ex: who | tee logfile | sort

Here, the output of who becomes the standard input of tee. tee now sends one copy of the input to sort through one pipeline, whereas the other copy is stored in a file called logfile.

$who | tee file1 file2 | sort

Here, the output of who is stored in file1 and file2 and one copy is sent to sort.

If we want to store the output of who in file1 and file2, display the same output on the screen and store the sorted output in file3, then we write

$who | tee file1 file2 /dev/tty3a | sort >file3

If we wish to append the output of tee to a file, -a option is used with it as shown below:

$cat file1 file2 | tee –a file3 | more

In this pipeline the output of cat(contents of the files file1 and file2) are appended to the existing contents of file3. Another copy of output of cat is sent to more for displaying on the screen.

comm: finding what is common: *comm* requires two sorted files, and compares each line of the first file with it's corresponding line in the second.

Ex : $cat file1	$cat file2	$cat file3
Raghu.V	Varun.T	Raghu.V
Shiva.P	Tharun.Q	Shiva.P
Roja.R	Raghu.V	Gopi.N
Gopi.S	Roja.R	Sekhar.S

$comm file1 file3

Raghu.V

Shiva.P

Gopi.N

Roja.R

Gopi.S Sekhar.S

comm can also produce selective output, using options -1, -2 or -3. To drop a particular column, simply use it's column number with the '-' sign. We can also combine options and display only those lines that are common.

$comm -3 file1 file2 $comm -12 file1 file2

cmp: comparing two files:

Whether two files are identical or not is determined through *cmp, diff and comm*. commands.

Ex: $cmp chap1 chap2

chap1 chap2 differ :char 9,line1

The two files are compared byte by byte and the location of the first mismatch(in the 9th character of first line) is echoed to the screen. cmp, when invoked without options, doesnot bother about possible subsequent mismatches.

The –l(list) option gives a detailed list of the byte number and the differing bytes in octal for each character that differs in both files:

$cmp –l file1 file3

17 162 147

30 12 56

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

diff: converting one file to other:

diff is used to display file differences. It tells us which lines in one file have to be changed to make the two files identical.

Ex: $diff file1 file3

3,4c3,4 #change line 3 of file1

< roja.r #}replace these

< gopi.s #}lines

- - - - - #with

> gopi.n #}these

> sekhar.s #}two lines

diff uses certain special symbols and instructions to indicate the changes that are required to make two files identical.

Maintaining several versions of a file: diff has an extremely useful option for the system administrator the –e option. If we have ten versions of a file differing only nominally, we now need to keep only one in full. For the others, we can keep only the differences, which help us conserve disk space.

tr (translate):

The *tr* (translate) command is a simple filter designed to replace one or more characters in given files with one or more characters. The syntax is:

Ex: tr [-cds] [in-string] [out-string]

The '-s' option substitutes all of the specified characters with another specified character and displays the results.

4. Exercises on Process, Text, network and backup Utilities-II

4.1 Hard Links and Symbolic Links

A link is a special file that references another file or directory. A link serves as an alias, which can be used to access the linked file/directory. For example, In Ubuntu, the link vi (in /usr/bin) references /etc/alternatives/vi.

Symlink: A symbolic link (or symlink or soft link) maintains a reference (not a direct pointer) to a file or a directory. A symlink is a file that stores the path to another file/directory. If the referenced file is removed, the symlink will be referencing an non-existent file.

To create a symbolic link, use command ln with option -s:

```
$ ln -s file-or-dir-name symlink-name
```

In "ls -l", symlinks are identified via the "symlink -> referenced file/dir".
Hard Link: A hard link is an *additional* pointer to the file's inode (physical location). You can view the inode of files via "ls -i". A file can be accessed via any hard link. It is available as long as there is at least one hard link left. The "hard link count" for a file is reflected in command "ls -l" (as illustrated in the earlier example).
To create a hard link, use command ln:

```
$ ln filename hard-link-name
```

Symlinks are more commonly-used. Symlinks can span file systems; while hard links work only in the same file system. Hard links usually work for file (hard link to directory could lead to inconsistency); while symlinks work for both file and directory. Hard links are also much hard to maintain, as there is little clue on where are the files, other than the link count.

You can remove a link (hard or soft) via rm command, just like any file. A hard-linked file is only deleted from the file system when there is no link to it.

Windows does not support symlink until Windows 7?!

4.2 Network Utilities

ftp: Copying files between non-unix hosts. It is the file transfer protocol.

```
Ex: $ftp source1        #open ftp connection to source1
Name :
Password :
```

Distributing processing:
Some hosts supply limited password less accounts with user ID's like 'user' so those explorers can roam the network without causing any harm.
Three utilities for distributing access are:
rlogin : Which allows you to log into a remote Unix host.
rsh : Which allows you to execute a command on a remote Unix host, and
telnet : Which allows you to execute commands on any remote host that has a telnet server.
Of these, 'telnet' is the most flexible, since there are other systems in addition to Unix that support telnet servers.
rlogin : Remote logins:To logon to the remote host, we use rlogin.
Syntax : rlogin –ec [-l useID] hostName
'rlogin' attempts to log you into the remote host *hostName.* If you don't supply a user ID by using the '-l' option, your local user ID is used during the login process.
If the remote host is not set as an equivalent of your local host in your "$Home/.rhost" file, your are asked for your password on the remote host.
Once you are connected, your local shell goes to sleep the remote shell starts to execute. When you are finished with the remote login shell, terminate it in the normal fashion(ctrl D) and your local shell will then awaken.
There are a few special "escape commands" each is preceded by the escape character, which is a tilde (~) by default, you may change this escape character by following the '-e' option with the preferred escape character.

```
Ex : $rlogin user2        #remote login
     Last login: Tue    Dec    20  17:23:51  from Varun
     user2 % date
     Wed    Dec  21  18:50:47   CDT   2005
     user2 % ^D       #terminate the remote login shell
     connection closed.
     $-                #back home again at Varun
```

Here, we have logged into remote host 'user2' from local host 'Varun'.

Remote connection: *'telnet'* allows you to communicate with any remote host on the Internet that has a '*telnet*' server.

Syntax : telnet [host[port]]

telnet establishes a two way connection with a remote port. If you supply a host name, but not a port specifier, you are automatically connected to a *telnet* server on the specified host, which typically allows you to log into the remote machine. If you don't even supply a host name, *'telnet'* goes directly into command mode (like *ftp*).

arp : Address Resolution Protocol:

The arp command is used to manipulate system arp cache.

$arp options hostname

Options:

-d - deletes hostname

-a - displays entry of given host

-n - shows numerical address

In a networked system when a packet arrives at router machine then the IP address to Ethernet address mapping is needed. This is achieved by arp protocol. Theses mappings are stored in arp cache such that next time another packet arrives the same ip address then its Ethernet or physical address is calculated

by carrying out a lookup operation on the arp cache. With arp command we can modify, view, delete the entries of this cache.

4.3 Tape Archive (tar) and ZIP Compression (gzip, bzip2)

The relevant file formats and utilities are:

- .gz: A file compressed via gzip utility; which can be uncompress via gunzip utility.
- .tar: A tar (tar archive) file, or tarball, is a collection of many files into a single file, used for distribution or backup. It is created via tar utility with option c (create); and can be extracted viatar with option x (extract).
- .tar.gz: A compressed tar file with gzip - the most popular format for software distribution and backup.

The Tape Archive (tar) Utility

You can use the tar utility to create, list, and extract .tar.gz files, as follows:

```
// "Create" a compressed archive of the given files and directories
$ tar czvf <compressed-filename>.tar.gz <directory1> <directory2> <file1> <file2> ...
  // Option "c" to create; "z" to compress; "v" for verbose; "f" for archive filename
```

Notes: to process .tar files without gzip compression (instead of .tar.gz), use the above commands minus the 'z' option.

gzip/gunzip Compression

gzip/gunzip (GNU zip) is an older compression utility. The resultant file type is .gz. gzip is still the most popular form for software distribution over the Internet. The man pages are also kept in gzip format. For example,

bzip2/bunzip2 Compression

bzip2/bunzip2 is a newer compression utility, which is more efficient than gzip, but not as popular. The

resultant file type is .bz2. For example,

```
$ bzip2 -v listing.txt        // Compress to listing.txt.bz2 (-v for verbose)
$ bunzip2 listing.txt.bz2     // Uncompress
```

5. Exercises on Shell Programming-I

5.1 Displaying files

Write a shell script called displaying type extension file that prompts user to enter. The script shall read the type extension files and display all file names to output stream. For example,

```
Input:
Enter directory name: cfiles
Output:
f1.txt f2.txt add.c fibo.c
```

Hints

```
// For keyboard input
/**
 * 1. Prompt directory name
 * 2. Change to given directory name
 * 3. Using ls command to display .txt and .c files
 */
    echo directory name

        read dir   // Scan the keyboard for input
        cd $dir   // check the given file is directory or not if directory then change to new directory
        // displaying files
        ls *.txt *.c // ls command to list all files
```

Try

Write a shell script called displaying type extension file with detailed information (filename, permissions, no of links, owner name, group name, file created. The script shall read the type extension files and display all file names to output stream.

5.2 moving files

Write a shell script called moving files from one directory to another directory that prompts user to enter source directory and destination directory. The script shall read the directory files and move all files from source directory to destination directory. For example,

```
Input:
enter the  Existing directory
olddir
"enter new directory"
newlydir
Output:
f1.text  f2.c  g1.txt  g2.c  g3.txt  g4.c  mvf.sh  newlydirf1.txt f2.txt add.c fibo.c
```

Hints

```
// For keyboard input
/**
 * 1. Prompt source directory name
 * 2. Prompt destination directory name
 * 3. Test given directory name directory and existing or not
 * 4. Using mv command move files from source to destination directory
 */
    mv ~/$dir/* ~/$ndir
    cd $ndir // change directory to new directory
    ls// ls command to list all files
```

Try

Write a shell script called moving all c extension files from one directory to another directory that prompts user to enter source directory, type extension files and destination directory. The script shall read the directory files and move all c type extension files from source directory to destination directory.

5.3 displaying logged in users

Write a shell script to display all the users who are currently logged in after a specified time. The script shall read the date and time, users who are logged in to linux server then displays users who logged in after specified time. For example,

```
Input:
enter the  Existing directory
olddir
"enter new directory"
newlydir
Output:
f1.text  f2.c  g1.txt  g2.c  g3.txt  g4.c  mvf.sh  newlydirf1.txt f2.txt add.c fibo.c
```

Hints

```
// For keyboard input
/**
 * 1. Prompt login time
 * 2. Use who command to list who are logged in linux server
 * 3. Extract time and user name
 * 4. Test the time with given time if condition is true then display time and username
 */
  echo "enter time to list specified users who login after specified time"
  read time1
  for i in `who|tr -s " " "|"|cut -d "|"-f1`
  do
  t=`w $i|tr -s " " ":"|tail -1|cut -d ":" -f4`
  for s in $t
  do
  if [ $time1 -le $s ]
  then
  echo $i $t
  fi
  done
  done
    cd $ndir // change directory to new directory
    ls// ls command to list all files
```

6. Exercises on Shell Programming-II

6.1 Wishing user based on time

Write a Shell Program to wish the user based on the login time. The script shall read the time of user logged in time and wish the user based on time. For example,

Input:
read the time of particular user
Output:
Good afternoon

try:

```
// For keyboard input
/**
 * 1. Prompt source directory name
 * 2. Prompt destination directory name
 * 3. Test given directory name directory and existing or not
 * 4. Using mv command move files from source to destination directory
 */
    h=`date +%H`
   if [ $h -lt 12 ] wish good morning
    elif [ $h -lt 18 ] wish Good afternoon
    else wish Good evening
```

6.2 Searching for specified word

Write a shell script that deletes all lines containing a specified word in one or more files supplied as arguments to it. The script shall read the word and filenames and display all lines other than of matching word. For example,

Input:

sh we2.sh we2.sh
enter the word
echo
Output:
The lines displayed other than given word

Hints

```
// For keyboard input
/**
 * 1. Prompt the word
 * 2. Prompt the file name
 * 3. Read all files of given directory
 * 4. Display all lines other than given word
 */
grep -v "$word" $fname
else
echo "file doesnot exist"
//test given file is existing or not if existing display between lines
```

Try

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

6.3 Displaying between lines

Write a shell script to read starting line and ending line of a file and display the lines in between them. For example,

```
Input:
enter file name
we1.sh
enter starting line number
2
enter ending line number
5
Output:
The lines displayed between line 2 and line 5 are
```

Hints

```
// For keyboard input
/**
 * 1. Prompt starting line
 * 2. Prompt ending line
 * 2. Find difference between two lines
 * 3. Using head and tail command of given file and display all between lines
 */
    d=`expr $el - $sl`
    head -$el $fname|tail -$d// test given file is existing or not if existing display between lines
```

Try

Write a shell script called displaying last 10 lines of given input file. The script shall read the line number and display all last lines.

7. Exercises on Shell Programming (Input, Decision and Loop)-I

7.1 Add2Integer (Input)

Write a shell script called Add2Integers that prompts user to enter two integers. The script shall read the two integers as int; compute their sum; and print the result. For example,

```
Enter first integer: 8
Enter second integer: 9
The sum is: 17
```

Hints

```
// For keyboard input
/**
 * 1. Prompt user for 2 integers
 * 2. Read inputs as "int"
 * 3. Compute their sum in "int"
 * 4. Print the result
 */
    // Put up prompting messages and read inputs as "int"
    echo enter a value and b value
    read a b   // Scan the keyboard for input
    // Compute sum
    sum=`expr $a + $b` or sum=$((a + b))

    // Display result
    echo "The sum is: $sum"   // Print with newline
```

Try

Write a shell script called Swap2Integers that prompts user for two integers. The script shall read the inputs as int, save in two variables called number1 and number2; swap the contents of the two variables; and print the results.

7.2 SumProductMinMax3 (Arithmetic & Min/Max)

Write a shell script called SumProductMinMax3 that prompts user for three integers. The script shall read the inputs as int; compute the sum, product, minimum and maximum of the three integers; and print the results. For example,

```
Enter 1st integer: 8
Enter 2nd integer: 2
Enter 3rd integer: 9
The sum is: 19
The product is: 144
The min is: 2
The max is: 9
```

Hints

```
    // Prompt and read inputs as "int"
    echo enter 3 numbers to find sum product min and max
    read number1 number2 number3   // Scan the keyboard
    // Compute sum and product use expr keyword
    sum = ......
    product = ......

    // Compute min
    // The "coding pattern" for computing min is:
    // 1. Set min to the first item
```

```
// 2. Compare current min with the second item and update min if second item is smaller
// 3. Repeat for the next item
min = $number1        // Assume min is the 1st item
if [ $number2 -lt  $min ]  // Check if the 2nd item is smaller than current min
then
   min = $number2;     // Update min if so
else if [ $number3 -lt $min ]
then
// Continue for the next item
   min = $number3;
fi
fi
// Compute max - similar to min
......

// Print results
......
```

Try

1. Write a shell script called SumProductMinMax5 that prompts user for five integers. The script shall read the inputs as int; compute the sum, product, minimum and maximum of the five integers; and print the results. Use five int variables: number1, number2, ..., number5 to store the inputs.

7.3 IncomeTaxCalculator (Decision)

The progressive income tax rate is mandated as follows:

Taxable Income	Rate (%)
First $20,000	0
Next $20,000	10
Next $20,000	20
The remaining	30

For example, suppose that the taxable income is $85000, the income tax payable is $20000*0% + $20000*10% + $20000*20% + $25000*30%.

Write a shell script called **IncomeTaxCalculator** that reads the taxable income (in int). The script shall calculate the income tax payable (in double); and print the result rounded to 2 decimal places. For examples,

```
Enter the taxable income: $41234
The income tax payable is: $2246.80

Enter the taxable income: $67891
The income tax payable is: $8367.30

Enter the taxable income: $85432
The income tax payable is: $13629.60

Enter the taxable income: $12345
The income tax payable is: $0.00
```

1. $41,234 → Tax: $2,246.80

Effective tax rate ≈ 2246.80 / 41234 ≈ **5.45%**

2. $67,891 → Tax: $8,367.30

Effective tax rate ≈ 8367.30 / 67891 ≈ **12.33%**

3. $85,432 → Tax: $13,629.60

Effective tax rate ≈ 13629.60 / 85432 ≈ **15.95%**

This strongly suggests **a tiered structure**, such as:

- 0% up to a threshold
- Then 10%
- Then 20%
- Then 30%...
- The program takes **taxable income** as input from the user.
- Based on the input amount, it calculates the **income tax payable**.
- The tax calculation follows a **progressive tax system**, where higher income is taxed at higher rates.
- For example, lower portions of income may be taxed at 10%, while higher portions may be taxed at 20% or more.
- In the first case, $41,234 results in a tax of $2,246.80 (about 5.45%).
- In the second case, $67,891 results in a tax of $8,367.30 (about 12.33%).
- In the third case, $85,432 results in a tax of $13,629.60 (about 15.95%).
- This shows that the **tax rate increases** as the income increases.
- The program uses if-else conditions or tax brackets to apply different rates on income ranges.
- The final output shows the input income and the computed tax in a formatted message.

Hints

```
// Declare constants first (variables may use these constants)
// The keyword "final" marked these as constant (i.e., cannot be changed).
// Use uppercase words joined with underscore to name constants
TAX_RATE_ABOVE_20K = 0.1
TAX_RATE_ABOVE_40K = 0.2
TAX_RATE_ABOVE_60K = 0.3
// Compute tax payable in "double" using a nested-if to handle 4 cases
if [ $taxableIncome -le 20000 ]         // [0, 20000]
then
   taxPayable = ......
else if [ $taxableIncome -le 40000)   // [20001, 40000]
then
   taxPayable = ......
else if [ $taxableIncome -le 60000 ]  // [40001, 60000]
then
   taxPayable = ......
else                              // [60001, ]
   taxPayable = ......
fi
// Alternatively, you could use the following nested-if conditions
// but the above follows the table data
//if [ $taxableIncome -gt 60000 ]          // [60001, ]
//then   ......
// else if [ $taxableIncome -gt 40000 ]   // [40001, 60000]
// then  ......
// else if [ $taxableIncome -gt 20000 ]   // [20001, 40000]
// then  ......
// else                              // [0, 20000]
//   ......
//
// Print results
Echo "The income tax payable is:
```

Try

Suppose that a 10% tax rebate is announced for the income tax payable, capped at $1,000, modify your program to handle the tax rebate. For example, suppose that the tax payable is $12,000, the rebate is $1,000, as 10% of $12,000 exceed the cap.

7.4 PensionContributionCalculatorWithSentinel (Decision & Loop)

Write a shellscript called PensionContributionCalculatorWithSentinel which shall repeat the calculations until user enter -1 for the salary. For examples,

```
Enter the monthly salary (or -1 to end): $5123
Enter the age: 21
The employee's contribution is: $1024.60
The employer's contribution is: $870.91
```

```
The total contribution is: $1895.51

Enter the monthly salary (or -1 to end): $5123
Enter the age: 64
The employee's contribution is: $384.22
The employer's contribution is: $461.07
The total contribution is: $845.30

Enter the monthly salary (or -1 to end): $-1
bye!
```

Hints

```
// Read the first input to "seed" the while loop
SENTINEL=-1
echo "Enter the monthly salary (or -1 to end): $"
read salary
while [ $salary -ne $SENTINEL]
do
  // Read the remaining
  echo "Enter the age:"
  read age
done
  ......
  ......

  // Read the next input and repeat
  Echo "Enter the monthly salary (or -1 to end): $")
  read salary
```

8. Exercises on Shell Programming (Input, Decision and Loop)-II

8.1 SalesTaxCalculator (Decision & Loop)

A sales tax of 7% is levied on all goods and services consumed. It is also mandatory that all the price tags should include the sales tax. For example, if an item has a price tag of $107, the actual price is $100 and $7 goes to the sales tax.

Write a shell script using a loop to continuously input the tax-inclusive price (in double); compute the actual price and the sales tax (in double); and print the results rounded to 2 decimal places. The script shall terminate in response to input of -1; and print the total price, total actual price, and total sales tax. For examples,

```
Enter the tax-inclusive price in dollars (or -1 to end): 107
Actual Price is: $100.00, Sales Tax is: $7.00
Enter the tax-inclusive price in dollars (or -1 to end): 214
Actual Price is: $200.00, Sales Tax is: $14.00
Enter the tax-inclusive price in dollars (or -1 to end): 321
Actual Price is: $300.00, Sales Tax is: $21.00
Enter the tax-inclusive price in dollars (or -1 to end): -1
Total Price is: $642.00
Total Actual Price is: $600.00
Total Sales Tax is: $42.00
```

Hints

```
// Declare constants
SALES_TAX_RATE = 0.07;
SENTINEL = -1;        // Terminating value for input
```

```
// Declare variables
totalPrice = 0.0, totalActualPrice = 0.0, totalSalesTax = 0.0  // to accumulate
......
// Read the first input to "seed" the while loop
echo "Enter the tax-inclusive price in dollars (or -1 to end): "
read price

while [ price -ne SENTINEL ]
do
  // Compute the tax
  ......
  // Accumulate into the totals
  ......
  // Print results
  ......
  // Read the next input and repeat
  echo "Enter the tax-inclusive price in dollars (or -1 to end):"
  echo $price
done
// print totals
......
```

8.2 ReverseInt (Loop with Modulus/Divide)

Write a shell script that prompts user for a positive integer. The script shall read the input as int; and print the "reverse" of the input integer. For examples,

```
Enter a positive integer: 12345
The reverse is: 54321
```

Hints

Use the following *coding pattern* which uses a while-loop with repeated modulus/divide operations to extract and drop the last digit of a positive integer.

```
// Extract and drop the "last" digit repeatably using a while-loop with modulus/divide operations
while [ $inNumber -gt 0 ]
  inDigit=`expr $inNumber % 10` // extract the "last" digit
  // Print this digit (which is extracted in reverse order)
  ......
  inNumber=`expr $isNumber / 10`          // drop "last" digit and repeat
}
......
```

Try

Write a shell script that prompts user for a positive integer. The script shall read the input as int; compute and print the sum of all its digits.

8.3 Amicable Numbers

Two different numbers are said to be so Amicable numbers if each sum of divisors is equal to the other number. Amicalble Numbers are: (220, 284), (1184, 1210), (2620, 2924), (5020, 5564), (6232, 6368). For example,

```
Enter 1st number: 228
Enter 2nd number: 220
The numbers are Amicable Numbers.
```

Hints

```
220 and 284 are Amicable Numbers.
Divisors of 220 = 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110
1+2+4+5+10+11+20+22+44+55+110 = 284
Divisors of 284 = 1, 2, 4, 71, 142
1+2+4+71+142 = 220
```

Try

Write a shell script called **TimeTable** that prompts user for the size (a positive integer in int); and prints the multiplication table as shown:

```
Enter the size: 10
 *|  1   2   3   4   5   6   7   8   9  10
 ---------------------------------------------
 1|  1   2   3   4   5   6   7   8   9  10
 2|  2   4   6   8  10  12  14  16  18  20
 3|  3   6   9  12  15  18  21  24  27  30
 4|  4   8  12  16  20  24  28  32  36  40
 5|  5  10  15  20  25  30  35  40  45  50
 6|  6  12  18  24  30  36  42  48  54  60
 7|  7  14  21  28  35  42  49  56  63  70
 8|  8  16  24  32  40  48  56  64  72  80
 9|  9  18  27  36  45  54  63  72  81  90
10| 10  20  30  40  50  60  70  80  90 100
```

8.4 Capricorn Number

A number is called Capricorn or Kaprekar number whose square is divided into two parts in any conditions and parts are added, the additions of parts is equal to the number, is called Capricorn or Kaprekar number. For example,

```
Enter a number : 45
45 is a Capricorn/Kaprekar number
Enter a number : 297
297 is a Capricorn/Kaprekar number
Enter a number : 44
44 is not a Capricorn/Kaprekar number
```

Hints

```
Number = 45
(45)2  = 2025

All parts for 2025:
202    + 5    =    207 (not 45)
20     + 25   =    45
2+  025    =    27 (not 45)
```

From the above we can see one combination is equal to number so that 45 is Capricorn or Kaprekar number.

Try

Write a shell script to generate and show all Kaprekar numbers less than 1000.

9. Exercises on Simulating commands - I

9.1 Simulating CAT Command

Write a program called simulating cat command that reads the file name and displays the file content to output stream.

Input:
Enter the filename: file1
Output:
displays the content of given file1 to output stream

Hints

```
#include<unistd.h>
// open function used to open existing file
open(argv[1],O_RDONLY);// file name read from command line arguments
while((n=read(fd,&buf,1))>0)// while loop to read content from given file character by
{                    character
 write(1,&buf,1);// Use write() to display characters to output stream


      .....
}      // repeat the while loop upto end of file
```

Try

Rewrite the above program to append new content to existing file.

9.2 Simulating CP Command

Write a program called simulating cp command that create duplicate file of exiting file, reads the file name and writes the file content to new file, repeats the procedure up to end of file.

Input:
Reads existing file
Output:
creates duplicate file and copy content from source file to new file

Hints

```
#include<unistd.h>
// open function used to open existing file
fd1=open(argv[1],O_RDWR);// read existing file
fd2=open(argv[2],O_RDWR);//create or open new file
while((n=read(fd,&buf,1))>0)// while loop to read content from given file character by
{                    character
 write(fd2,buf,sizeof(buf));// Use write() to write characters to new file


      .....
}      // repeat the while loop up to end of file
```

Try

Rewrite the above program to simulate cp -R command to recursive copy (including hidden files.

9.3 Simulating RM Command

Write a program called simulating rm command that remove exiting file or group of files of given directory, reads the file name or group of filenames and directory name then removes that related files from given directory using rm command.

Input:
Reads existing file
Output:
removes source file

Hints

```
#include<unistd.h>
//system function executes linux commands
system(cd dir) //move to specific directory
remove("abc.txt");// remove function deletes existing file
```

Try

Rewrite the above program to simulate rm -i command for interactive remove existing file or group of files.

9.4 Simulating LS Command

Write a program called simulating ls command using low level system calls, that reads particular directory and displays the list files, subdirectories and executable files to output stream.

```
Input:
Enter the filename: file1
Output:
# displays the content of given file1 to output stream
```

Hints

```
#include<unistd.h>
// opendir function used to open existing directory
DIR *dp;
struct dirent *p;
dp=opendir("dir1");// opens dir is directory name
while((p=readdir(dp))!=NULL) // while loop to read files from given directory
    {
 printf("%d\t",p->d_ino); // displays inode number of file
 printf("%s\n",p->d_name);// displays name of the file
      .....
}      // repeat the while loop upto end of directory file
 close(dir); //closes open directory
```

Try

Rewrite the above program to implement ls -l command to display long listing files.

10. Exercises on Simulating commands - II

10.1 Simulating head command

Write a program called simulating head command using system calls, that reads existing file name and displays the first 10 lines content of file to output stream.

```
Input:
gcc headcmd.c -o headcmd
./headcmd
Output:

# displays the content of given file1 first 10 lines to output stream
```

Hints

```
#include<unistd.h>
 // reads file name from command line arguments
 open(argv[1],O_RDONLY;");// open function opens file for read only purpose
 while(lseek(read_fd,offset, SEEK_SET) < statFd.st_size)
  // lseek function to move cursor to particular location of given file
  {
    if(read(read_fd, &lu, 1) != -1)
    {// reads line by line
          . . . .
       printf("%c",lu);// displays content line by line
          ......
     }      // repeat the while loop upto condition fail
      close(fd); //closes open file
   }
```

Try

Rewrite the above program to implement head -20 command to display top 20 lines.

10.2 Simulating tail command

Write a program called simulating tail command using system calls, that reads existing file name and displays the last 10 lines content of file to output stream.

Input:
gcc tailcmd.c -o tailcmd./headcmd
./tailc

```
#include<unistd.h>
 // reads file name from command line arguments
/* Allocate space for tail buffer */
 tail = calloc(count, sizeof(char *));
 for (i = 0; i < count; i++) {
   tail[i] = calloc(MAX_LINE_LEN, sizeof(char));
 }
/* Fill circular tail buffer until EOF */
while (fgets(tail[tailX], MAX_LINE_LEN, stdin) != NULL) {
   tailX = (tailX + 1) % count;
   if (tailX == headX) {
     headX = (headX + 1) % count;
   }
 }
/* Display tail */
i = tailX;
do {
```

md

file

Outp

ut:Hi

nts

```
  printf("%s", tail[i]);
  i = (i + 1) % count;
  } while (i != tailX);
return 0;
   }
```

Try

Rewrite the above program to implement tail command to display last lines with option, option is number of lines take as input.

10.3 Simulation of mv command

Write a program called **simulation of mv command**, which prompts user for source file and destination file to rename file. The output shall look like:

```
Input:
$gcc mvcmd.c -o mvcmd
$/mvcmd xx movefile
Output:
#renames source file to destination file.
```

Hints

```
#include<unistd.h>
// Define variables
int main()
{
   ........
open(argv[1],O_RDONLY);// opens existing file
creat(argv[2],S_IWUSR);// creates new file
rename(fd1,fd2); // rename source file with new file
unlink(argv[1]);//removes existing file
   ........
}
```

Try

Rewrite the above program to implement mv command to move files from source directory to new directory.

10.4 Simulation of nl command

Write a program called simulation of nl command, which prompts the user for a file name and read line by line and display line number before each line. For example,

```
Input: $ gcc nlprg.c -o nlprg
$ ./nlprg sample1
  1 aaaaaaaaaa
  2 sssssssssss
  3 ddddddddddd
  4 fffffffffff
  5 ggggggggggg
```

```
#include<unistd.h>
// Define variables
 int main()
{
 while(lseek(read_fd,offset, SEEK_SET) < statFd.st_size)
 {
     if(read(read_fd, &lu, 1) != -1)
     {
       printf("%c",lu);
       offset++;
       if(lu=='\n')
         {
         printf("  %d",++counter);          //counter++;
         }
     }
            . . . . .
   }
}
```

Try

Rewrite the above program to implement nl command to display special symbol before each line.

11. Exercises on Signal Handling

11.1 Signal handler function with SIGINT

Write a program called signal handler to catch SIGINT, SIGINT signal which interrupts the process by pressing ctrl + c. For example,

```
Input
$ gcc signalpg.c -o sp
Output:
$ ./sp
received SIGINT
```

Hints

```
// Declare variables
#include<unistd.h>
void sig_handler(int signo) // function definition
 {
   if (signo == SIGINT)
   printf("received SIGINT\n");
 }
 int main(void)
 {       ......
   if (signal(SIGINT, sig_handler) == SIG_ERR) // function calling
   printf("\ncan't catch SIGINT\n");
   // A long long wait so that we can easily issue a signal to this process
   while(1)
   sleep(1 );             ......
 }
```

11.2 Signal handler function with SIGDFL

Write a program called signal handler to catch SIGINT, SIGINT signal which interrupts the process by pressing ctrl + c, SIG_DFL signal do default action i.e. terminates the current process. For example,

```
Input
$ gcc signalpg.c -o sp
Output:
$ ./sp
#Press ctrl + c, Process terminates automatically.
```

Hints

```
// Declare variables
#include<unistd.h>
void sig_handler(int signo) // function definition
  {
    if (signo == SIGINT)
    printf("received SIGINT\n");
  }
 int main(void)
  {        .....
     if (signal(SIGINT, SIG_DFL) == SIG_ERR) //default action
     printf("\ncan't catch SIGINT\n");
     // A long long wait so that we can easily issue a signal to this process
     while(1)
     sleep(1 );              .....
   }
```

11.3 Signal handler function with SIGKILL

Write a program called signal handler to catch SIGINT, SIGINT signal which interrupts the process by pressing ctrl + c, SIG_DFL signal do default action i.e. terminates the current process. For example,

```
Input
$ gcc signalpg.c -o sp
Output:
$ ./sp
#Press ctrl + DEL, Process ignores SIGKILL signal.
```

Hints

```
#include<unistd.h>
void sig_handler(int signo) // function definition
  {
    if (signo == SIGINT)
    printf("received SIGINT\n");
   }
 int main(void)
  {        .....
    if (signal(SIGKILL, SIG_IGN) == SIG_ERR) //ignoring signal
    printf("\ncan't catch SIGINT\n");
    // A long long wait so that we can easily issue a signal to this process
    while(1)
```

```
    sleep(1 );
    ......
  }
```

Try

Rewrite the above program to generate SIGABRT signal and call function to display message process aborted.

12. Exercises on Inter Process Communication (IPC)

12.1 One way communication using pipe

Write a program to implement one way communication using pipes. The program shall execute pipe function, stores data at one end reads the data from other end, fork function creates new process and exchange data between related process. For examples,

```
Input:
$ gcc pipepg.c -o pp
$./pp hello
Ouput:world
```

Hints

Use the following *coding pattern* which uses a pipe, fork function to exchange data between to related process.

```
# one way communication using pipes
int main(void)
 {
# declare variables
if (pipe(fd) < 0) //execute pipe function to create pipe
printf("pipe error");
if ((pid = fork()) < 0) // creates new process is called child process
{
printf("fork error");
}
else if (pid > 0) // parent process writes the data to pipe one end
 { /* parent */
    . . . .
    write(fd[1], "hello world\n", 12);
  } else
 { /* child */
 . . . .
 read(fd[0], line, MAXLINE); // child process reading data from pipe
 write(STDOUT_FILENO, line, n); // displaying data to output stream
}
```

Try

Rewrite the above program to implement two way communication using pipes. The program shall execute two pipe functions, one fork function exchange data between related process between parent and child process.

12.2 One way communication using fifo function

Write a program to implement one way communication using fifo function. The program shall execute mkfifo function, creates fifo file, stores data in fifo file by one process and other process has to read by other process, it exchanges data between un-related process. For example,

```
Input:
$ gcc producer.c -o producer          #first window
$gcc consumer.c  -o consumer          # second window
Output:
$ ./producer                        #first window
$./consumer                          # second window
Producer:
Producer sent: hai                    #first window
Consumer read: hai                     # second window
Producer sent: good morning           #first window
Consumer read: good morning            # second window
Producer sent: welcome                #first window
Consumer read: welcome                 # second window
$./pp hello
Ouput:world
```

Hints

Use the following *coding pattern* which uses a mkfifo function to create fifo file and exchange data between to un-related process.

```
# one way communication using fifos
//FIFO or Unnamed(Child Process)
#include<unistd.h>
int main()
{
 //declare variables
 open(FIFO_NAME, O_RDONLY);//Open existing fifo file
 while(1) //repeat while loop upto reading end of file
{
 n=read(fd, r, MAXSIZE);
 if(n > 0)
 printf("\nConsumer read: %s", r);
}
}

//FIFO or Unnamed(Parent Process)
#include<unistd.h>
int main()
{
// declare variables
fifoid=mkfifo(FIFO_NAME, 0755); // create fifo file
while(1)
{
 // write content to fifo file
 read(0, w, MAXSIZE);
 n=write(fd, w, MAXSIZE);
 if(n > 0)
 printf("\nProducer sent: %s", w);
```

```
}
}
```

Try

Rewrite the above program to implement two way communication using fifos.

12.3 storing messages in Message queues (sender)

Write a program (sender.c) to create a message queue with read and write permissions to write 3 messages to it with sequence order. For example,

Input:

```
$gcc msend.c -o msend
Output:
$./msend
Enter the message to send: hi
Enter the message to send: hello, how are you
Enter the message to send: bye
```

Hints

Create message queue or open an existing using msgget() which returns a common key in order to gain access to the queue and communicating between different process using msgsnd() and msgrcv(). Each message is given an identification or type so that processes can select the appropriate message.

```
#include<unistd.h>
#include<string.h>
int main()
{
//declaring variables
struct
{
 long mtype;
 char mtext[15]; //declaring message structure
 }message;
 msgget((key_t)10,IPC_CREAT|0666);  //create message queue with key 10
 msgsnd(qid,&message,len+1,0); //store messages in existing message queue
}
```

Rewrite the above program (sender.c) to create a message queue with read and write permissions to write 3 messages to it with different priority numbers.

12.4 Retrieving messages from message queues (receiver)

Write a C program (receiver.c) that receives the messages (from the above message queue as specified and displays them to output stream. For example

```
Input:
$ $gcc mrecv.c -o mrecv
Output
$./mrecv
```

```
Message received from sender is: hi
Message received from sender is: hello, how are you
Message received from sender is: bye
```

Hints

A message queue is a linked list of messages stored within the kernel and identified by a message queue identifier. A new queue is created or an existing queue opened by msgget(). New messages are added to the end of a queue by msgsnd().Messages are fetched from a queue by msgrcv(). Fetch the messages in a first-in, first-out order. All processes can exchange information through access to a common system message queue. Process must share a common key in order to gain access to the queue in the first place.

```
#include<sys/ipc.h>
#include<sys/msg.h>
main()
{
//declaring variables
struct
{
long mtype;
char mtext[15]; //declaring message structure
}buff;
msgget((key_t)10,IPC_CREAT|0666);
        //accessing existing message queue with key 10
if(msgrcv(qid,&buff,15,i,0)==-1)//retrieve messages from message queue
printf("Message received from sender is %s\n",buff.mtext);
   //displaying messages to output stream
}
```

Try

Rewrite the above program (receiver.c) to access existing message queue with read and write permissions to retrieve and display 3 messages to output stream with different priority numbers.

13. Exercises on Shared Memory

13.1 Sharing memory segment between processes

To write a program to implement inter process communication using shared memory. The program creates shared memory segment and stores data, multiple users can access data at the same time. For example:

```
Input:cc shmem.c
Output:./a.out
Write Data :hai
child: Data read from memory: hai
parent: Data written in memory: hai
```

Hints

Shared memory is the fastest method of interprocess communication (IPC) under Linux and other Unix- like systems. The system provides a shared memory segment which the calling process can map to its address space. After that, it behaves just like any other part of the process's address space due to which as soon as the first process writes data in the shared memory segment, it becomes available to the second process..

```
#include <sys/ipc.h>
#include <sys/shm.h>
int main()
{
```

```
    // ftok to generate unique key
    key_t key = ftok("shmfile1",10); //generates key
    int child=fork();
 if(!child)
 {
    // shmget returns an identifier in shmid
    int shmid = shmget(key,1024,0666|IPC_CREAT);
    // shmat to attach to shared memory
    char *str = (char*) shmat(shmid,(void*)0,0);
    printf("Write Data : ");
    gets(str);
    printf("parent: Data written in memory: %s\n",str);
 }
    else
    {    // ftok to generate unique key
    key_t key = ftok("shmfile1",10);

    // shmget returns an identifier in shmid
    int shmid = shmget(key,1024,0666|IPC_CREAT);
    // shmat to attach to shared memory
    char *str = (char*) shmat(shmid,(void*)0,0);
    printf("child :Data read from memory: %s\n",str);
    //detach from shared memory
    shmdt(str);
    // destroy the shared memory
    shmctl(shmid,IPC_RMID,NULL);
  }
 }
```

Try

Rewrite the above program to implement inter process communication using shared memory without fork function. The program creates shared memory segment and stores data (sender process), multiple users can access data (receiver process) at the same time. For example:

14. Exercises on Socket Programming

14.1 echo Client Server Program using TCP elementary functions

1. **#include <sys/ipc.h>**

- Includes the header for **IPC (Inter-Process Communication)** key functions.
- Provides access to ftok() which generates unique keys to identify IPC objects like message queues.

2. **#include <sys/msg.h>**

- Contains definitions for working with **System V message queues**.
- Provides functions such as msgget(), msgsnd(), msgrcv(), and msgctl() to manage message queues.

3. **#include <stdio.h>**

- Standard Input/Output library.
- Used here for input/output functions like printf() and fgets().

sender.c – **Sending Messages via System V IPC**

```
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
// Define message structure
struct msg_buffer {
  long msg_type;
  char msg_text[100]; };
```

```
int main() {
  key_t key = ftok("progfile", 65);            // Generate a unique key
  int msgid = msgget(key, 0666 | IPC_CREAT);     // Create/get message queue
    struct msg_buffer message;
  message.msg_type = 1;
  printf("Enter message: ");
  fgets(message.msg_text, sizeof(message.msg_text), stdin);  // Read message
  msgsnd(msgid, &message, sizeof(message), 0);   // Send message
  printf("Message sent\n");
  return 0;
}
```

ftok("progfile", 65)	Generates a key based on the file "progfile" and project id 65.
msgget(...)	Creates a message queue or connects to it using the generated key.
struct msg_buffer	Structure representing the message (mandatory format: long type followed by data).
msgsnd(...)	Sends the message to the message queue.
fgets(...)	Reads user input safely, including spaces.

📌 Note: The file progfile **must exist** in the working directory.

► Compile the Program:

```
$ gcc sender.c -o sender
```

► Run the Sender:

```
$ ./sender
Enter message: Hello from sender!
Message sent
```

A message queue is created (or reused if it already exists).
The user enters a message (e.g., "Hello from sender!").
The message is sent to the queue using msgsnd.
A corresponding receiver.c can be used to fetch the message using msgrcv.

System-Level Insight

To see the message queue:
$ ipcs -q
To remove the queue after use:
$ ipcrm -q <msgid>
Optional: Sample receiver.c Code (for testing)

```
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
struct msg_buffer {
  long msg_type;
  char msg_text[100];
};
int main() {
  key_t key = ftok("progfile", 65);
  int msgid = msgget(key, 0666 | IPC_CREAT);
  struct msg_buffer message;
  msgrcv(msgid, &message, sizeof(message), 1, 0);
  printf("Received message: %s", message.msg_text);
  msgctl(msgid, IPC_RMID, NULL);  // remove the queue
```

return 0; }

14.2. Receiver (Reading Messages from Queue)

#include <sys/ipc.h>

This header file defines the structures and functions related to **Inter-Process Communication (IPC)** keys.

It provides functions like ftok() which generates a unique key for IPC objects (message queues, shared memory, semaphores).

#include <sys/msg.h>

This header contains the definitions and function prototypes needed for working with **System V message queues**.

It provides functions such as msgget(), msgsnd(), msgrcv(), and msgctl() to create, send, receive, and control message queues.

#include <stdio.h>

The standard input/output library.

Provides functions like printf() and fgets() for console input/output operations.

receiver.c – **Receiving Messages via System V IPC**

Source Code Recap

```
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
struct msg_buffer {
long msg_type;
char msg_text[100];};
int main() {
  key_t key = ftok("progfile", 65);                 // Generate the same key
  int msgid = msgget(key, 0666 | IPC_CREAT);        // Connect to the message queue
  struct msg_buffer message;
  msgrcv(msgid, &message, sizeof(message), 1, 0);   // Receive message of type 1
  printf("Received message: %s", message.msg_text);  // Display the message
  msgctl(msgid, IPC_RMID, NULL);                    // Delete the message queue
  return 0;
}
```

ftok("progfile", 65)	Generates a key from the same file and ID used in sender.c.
msgget(...)	Connects to the existing message queue.
msgrcv(...)	Receives a message of type 1 from the queue.
printf(...)	Prints the received message.
msgctl(..., IPC_RMID, ...)	Deletes the message queue after receiving (cleanup step).

Run Scenario with Simulated Terminal Output

Terminal 1: Compile and Run Sender

```
$ gcc sender.c -o sender      Enter message: Hello from sender!
$ ./sender                    Message sent
```

Terminal 2: Compile and Run Receiver

```
$ gcc receiver.c -o receiver
$ ./receiver
Received message: Hello from sender!
```

Message received successfully!

The message queue is deleted right after receiving, to avoid clutter or stale queues.

Behind the Scenes

- sender.c puts the message in a queue.
- receiver.c retrieves it using msgrcv(...).
- msgctl(..., IPC_RMID, ...) cleans up the queue.
- The key generated with ftok() ensures both processes refer to the **same queue**.

Optional Commands for Manual Inspection

- View message queues:

$ ipcs -q

- Remove manually (if needed):

$ ipcrm -q <msgid>

- Uses System V message queues.
- msgsnd() and msgrcv() send and receive messages using a common key.
- msgctl() removes the queue after receiving.

Try:

1. Write a sender program that stores a message in a System V message queue.
2. Write a receiver program that retrieves and displays the message from the message queue.

V. TEXT BOOKS:

1 Sumitabha Das, "*Your Unix The Ultimate Guide*", Tata McGraw-Hill, New Delhi, India, 2007.
2 B. A. Forouzan and R. F. Gilberg, "*Unix and Shell Programming*", Cengage Learning.

VI. REFERENCE BOOKS:

1 Robert Love, "*Linux System Programming*", O'Reilly, SPD.
2 Stephen G. Kochan, Patrick Wood, "*Unix Shell Programming*", Sams publications, 3rd Edition, 2007.
3 T. Chan, "*Unix System Programming using C++*", Prentice Hall India, 1999.

VII.. ELECTRONICS RESOURCES:

1 http://spoken-tutorial.org/tutorialsearch/?search_foss=Linux&search_language=English
2 https://www.redhat.com/en/files/resources/en-rhel-whats-new-in-rhel-712030417.pdf
3 http:// www.tutorialspoint.com/unix/
4 http://cse09-iiith.virtual-labs.ac.in

VIII. MATERIALS ONLINE

1 Course Template
2 Lab Manual