UNIT-1

Review of unix utilities and shell programming

O.S:-

An operating system is the software that manage the computer hardware and provides a convinient and safe environment for running programs. It acts as an Interface between program and the hardware resources. that these programs access like memory, printer and harddisk

Features of operating system:-

1) The operating system allocates memory for the program and loads the program to the allocated memory.
2) It also loads the cpu registers with control information related to the program. The
3) The instructions provided in program

are executed by C.P.U.

4) If the program needs to access the hardware, it makes a call to the O.S rather than attempt to do the Job itself

5) After the program has completed the execution the O.S cleans up the memory and registers and makes them available for the next program.

Introduction to unix :-

The O.S is developed for 2 purpose. They are

1) Single user System.
2) Multi user System.

1) Single user

The system which was designed for use for by one person at a time is known as single user system.

The system which can ~~is~~ be used by more than one person is known as the multiuser system.

These types of system would be required when no. of applications have to run simultaneously like printers, disks are to be shared by no. of users.

Unix is most popular o.s on multiuser systems. It is origionated as the single user system at Bell laboratories in 1969 by ken Thomson along with the help from Dennis Ritchie and others wrote a small general purpose o.s which includes utilities to manage files & processes.

In 1973, Thomson and Ritchie rewrote the unix o.s in C around 1974 it was licensed to universities for educational purpose and a few years later unix was made commercially available

unix is termed as unics. The full form of unics is uniplexed information and computing system.

Features and Benifits of UNIX:-

The unix o.s is available on wide range of machines, from microprocessor to mainframes and on machines of different manufactures. The popularity and success of this is due to the following reasons.

1. portability:- It is written in high level language making it easier to read, understand, change and move

to other machines.

2. Machine Independent:-
The system hides the machine architecture from users, making it easier to write applications that can be run on any machine.

3. Multiuser:- Unix is a multi user system designed to support a group of users simultaneously. The system allows for sharing of processes, power and other hardware resources while at the same time excellent security features.

4. Heirarchical file system:-
It uses the hierarchical structure to store information. It allows for easy maintainance and efficient implementation

5. Pipes and filters:- Unix has facilities called pipes and filters which permits the user to create complex programs from the simpler programs

6. Background processing:- Unix has a facility which the user can start a task and then proceed to work on other task in the background and the remaining in the foreground. "Background processinges helps the user in effective utilization of time"

7. Software development tools:-
UNIX offers an excellent variety of tools for software development for all phases from program editing to maintainance of software.

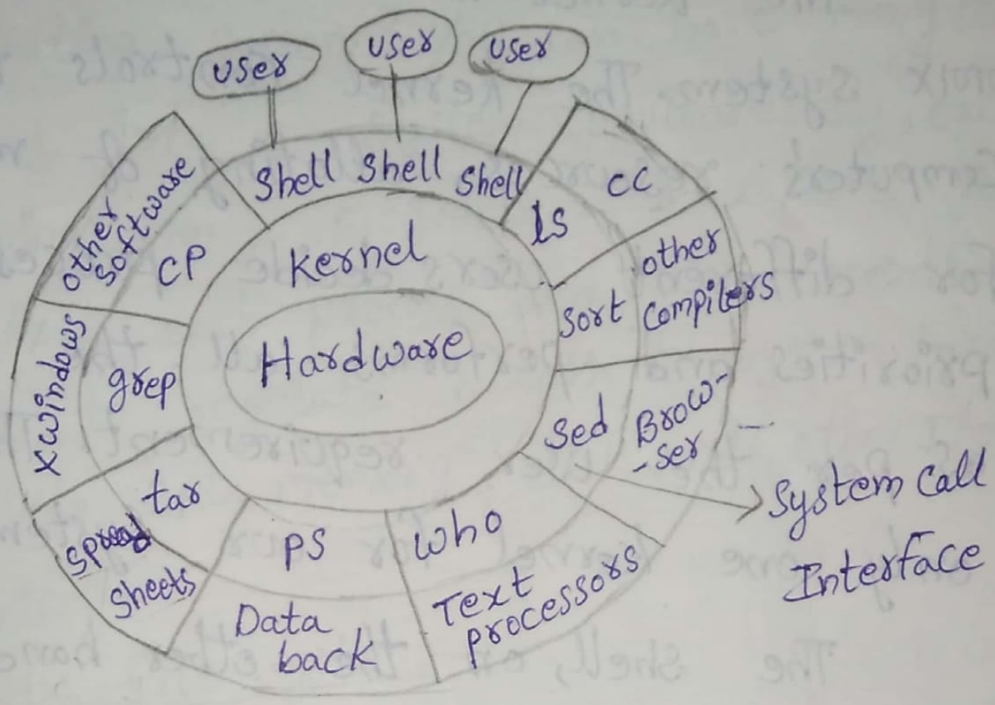
8. Maturity:- UNIX supports a wide variety of languages - C, COBOL, PASCAL, ADA, BASIC, LISP AND PROLOG.

* Different flavours of UNIX:-
1. BSD (Berkeley Software distribution)
2. SUN SOLARIS (Sun MicroSystems)
3. NOVEL NETWARE (Novell)
4. IBM SIX (International business machines)
5. HP-UX (Hewllet packard)

6. TRU 64 (Hewllet packard)

7. LINUX

★ UNIX STRUCTURE (OR) ARCHITECTURE:-



UNIX is a layered O.S. The inner-most layer is the hardware that provides the services for the O.S. In UNIX operating System is treated as the kernel. This kernel interacts directly with the hardware and provides the services to the User programs. The users don't need to know about the hardware Just they need to know how

to work and interact with the kernel and it's up to the kernel to provide the desired service.

The kernel is the heart of the unix system. The kernel controls the computer's resources, alloting of memory for different users, decide process priorities and performs all the tasks as per the user requirement. There is only one kernel for our system.

The shell, on the other hand is technically a unix command. The shell acts as an interpreter b/w user and kernel. So many people work on unix and LINUX environments. but they don't know what exactly shell does. The computer doesn't have the capability of converting the commands in to machine understandable format this conversion is done by the shell. It takes the commands from the user, disapears it,

and by exchanging information and sends that command to the kernel. It has a programming capability of its own.

"Unix is a multiuser, multi-tasking operating system". You can have many users logged into a system simultaneously, each running many programs. It's a kernel job to keep each process and user seperate and to regulate access to system hardware include CPU, memory, disk and other I/O devices.

19/6/17

Unix File System:-

The file is a container for storing information. Unix file doesnot contain the eof (end-of-file) mark. A file's size is not stored in the file, nor even its name. Unix treats directories and devices as files as well. All physical devices like the hard disk, memory, CD-ROM, printer and ~~divided~~ modem are treated as files. These files are divided into three categories:-

1. ordinary files (regular)
2. Directory files
3. Device files.

1. ordinary files:-

An ordinary file or regular file is the most common file type. All program files belongs to this type. ordinary file Can be divided in to two types:

a) Text file
b) Binary file

(ASCII) → (0 - 255) for all printable & non

a) textfile:-

Text file contains only printable characters and you can often view the contents and make sense out of them. A Textfile contains lines of characters ~~while~~ where every line is terminated with the newline character also known as linefeed (LF)

b) Binary file:- (ex:- enter key (↵))

A Binary files contains both printable and unprintable characters that cover

the entire ASCII range (0 to 255). Most unix commands are binary files are the object code and executables that produced by compiling C programs are also binary files.

2. Directory files:- A directory contains no data, but keeps some details of the files and sub-directories that it contains. A directory is a store house of information about the file and the sub directories in that directory. Each entry in the directory has two components

* The filename
* A unique identification number for the file or directory (called the inode number)

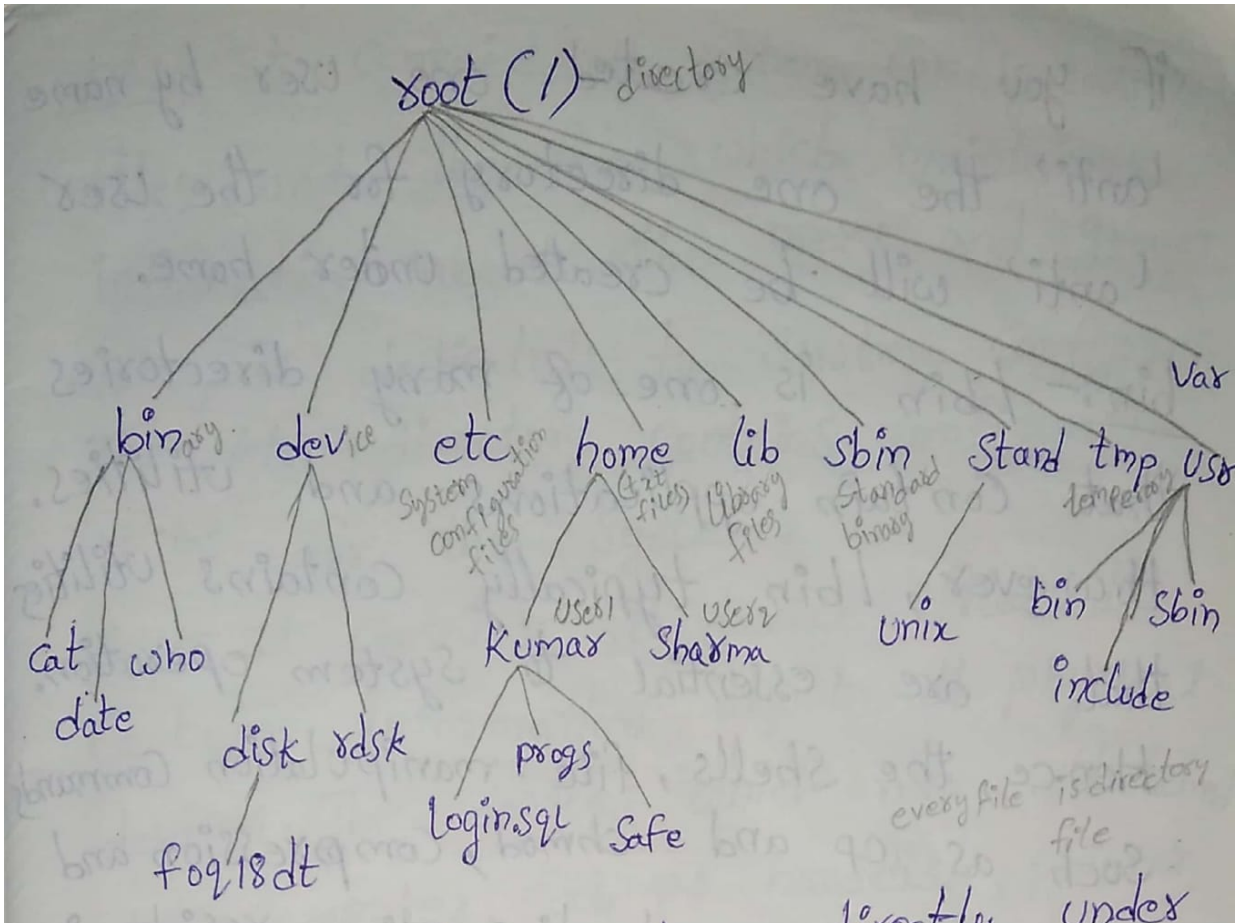
3) Device file:- Most of the system files in UNIX are special files. The device file is special in the sense that any output directed to it will be reflected

on to respective I/O device associated with the filename. Special files are typically associated with input-output devices. Such files are found in the standard UNIX directories such as /device and /etc., users cannot alter these special files.

The structure of unix file system (or) The parent-child Relationship:-

All files in Unix are related to one another. The file system in UNIX is a collection of all these related files (ordinary, directory and device files) organized in a hierarichical (an inverted tree) structure.

The following diagram displays the structure of the UNIX file system:



The directories shown directly under root are normally found on every UNIX System. in addition to others.

Root:- Root is actually a directory file and it has number of sub directories under it. These Sub-directories in turn, have more Sub-directories and other files under them. for instance bin and user are the Sub-directories under root.

home:- Default directory allocated for the users when the administrator doesn't specify any directory then by default home directory is allocated. Consider

if you have created one user by name 'anti' the one directory for the user 'anti' will be created under home.

bin:- /bin is one of many directories that contain applications and utilities. However, /bin typically contains utilities that are essential to system operation. Hence, the shells, file manipulation commands such as cp and chmod, compression and de-compression and diagnostics reside in /bin.

Sbin:- It contains all system administ-rator executable files. Commands generally the normal user don't have the privilege for using it. Sbin also contains utilities crucial to system operation and maintenance. However, the programs found in /Sbin can be executed only by super users.

etc:- It contains all system configuration files and the files which maintain information about the users and groups. /etc is dedicated to system configuration. The /etc directory contains configuration files for the system.

dev:- It contains all device files. It contains information about all physical devices. The devices which are attached to computer, such as harddisk, printer, cdrom, etc., whenever we add or remove any of the device on UNIX system. the dev directory will be updated.

lib:- It contains all library and object files. All files here are vital and the ~~object~~ corruption or removal of even one file can render a system useless. These are used to create new executa-bles from source code.

tmp:- System and users create temporary files which will be removed when the server reebots. /tmp or "temperory",

is the System-wide. scratch pad. files in /tmp are Considered disposable. Indeed system administrator probably deletes all files older than a certain expiry every evening.

usr:- Default directory provided by Unix operating System to create users home directories and Contains manual pages. Manual pages are used for help. /user is the umbrella for a great number of files

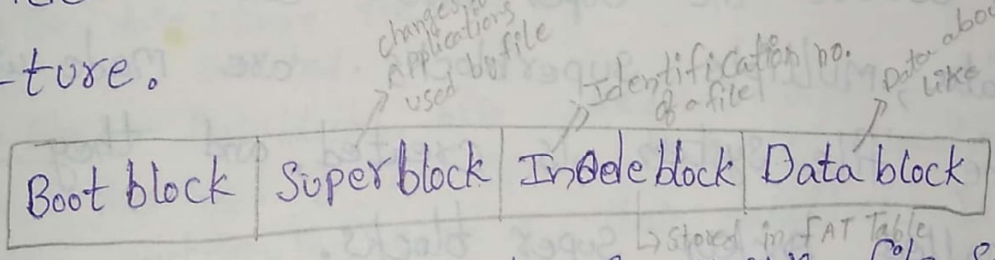
var:- var short for "variable" is the repository for files that typically grow in size over time mailboxes, log files, printer queues and databases can be found in /var. It's Common place also for websites to be kept in /var because a website tends to a mass data-preternaturally over time.

20/6/17

UNIX filesystem Architecture:-

50 In UNIX, a hard disk partition is considered as any array of disk block logically, where as disk block can be a physical sector or multiples of physical Sectors on the disk. It contain four important areas namely, boot block, Super block, Inode block, and data block.

The following diagram gives an idea about UNIX file System architecture.



Logical components of UNIX file System architecture.

1. Boot Block:- This stores bootable objects that are necessary for booting the System. And through space is reserved for boot block in all the cylinder groups, only the first cylinder group has bootable information.

2) Super Block:- This stores all the information about the file system like:

* Size and status of file system.
* Label, which includes the file system name and volume name.
* Size of the file system logical block.
* Date and time of the last update.
* Cyllinder group size.
* Number of data blocks in a cyllinder group.
* Summary data blocks.
* file system state.
* Pathname of the last mount point.

Because the super block contains the critical data, multiple superblocks are made when the file system is created, and they are called backup super blocks.

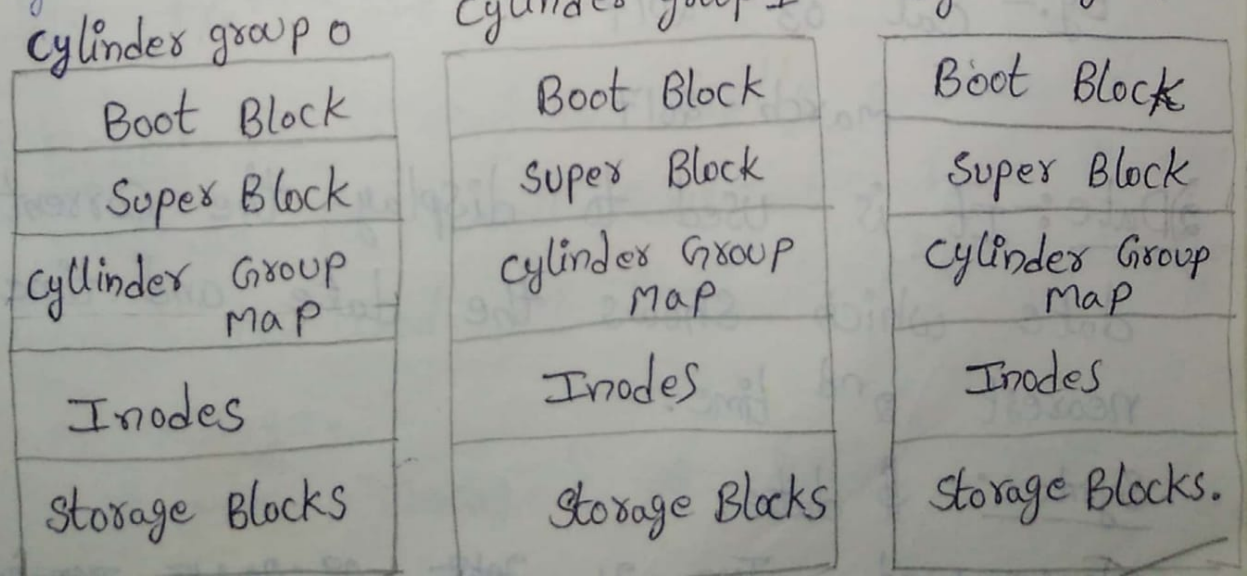
A summary information is kept within the super block. The summary information block is not replicated, but is grouped with the primary super blocks. ~~records changes that take place~~ The summary records changes that take place as the file system is used. In addition, the summary block units the number of Inodes, directories, fragments and storage blocks.

within the file system.

3. Inode block :- An Inode contains all the information about a file except its name, which is kept in a directory. An Inode is 128 bytes. The Inode ~~is~~ information is kept in the cyllinder information block and contains the following: * The type of the file.

* The mode of the file (permissions).
* The no. of hard links to the file.
* The USERID of the ~~another~~ owner of the file.
* The groupID ~~of~~ to which the file belongs.
* The number of bytes in the file.
* An array of 15 disk block Addresses.
* The date & time the file was last accessed and modified and was created.

4) Data Block :- Datablocks, also called as storage blocks contains the rest of the space that is allocated to the file system. The size of these datablocks is determined when a file system is created. By default data blocks are allocated in two sizes: an 8k byte logical block size and 1-k fragment size.



Built block Approach:-

Unix provides a toolkit with set of commands to execute certain command type. Command types are

1) General utilities
2) file utilities
3) Disk utilities
4) network utilities
5) Man Documentation
   ↳ manual
6) pattern matching

21/6/17

1) General utilities:-

1) Cal:- The cal is used to see the calender of any specific month or year.

Syntax:- $ cal [month] [year]

Eg:- cal 03 2017

march 2017

2) Date:- It is used to display the current date which shows the date and time nearest 2nd time.

Syntax:- $ date

Ex:- Wed Jun 21 ~~2017~~ 09:20:15 IST 2017

Options:-

date -d → The date of month i.e., 21

date -y → The year of i.e. The last 2 digits of the year is display by -y i.e., 2017

date -H, -M, -S → The hour, minute, second of time is pt displayed i.e., 09:26:15

date -D → It displays date in the format of MM/dd/YY i.e., 06/21/17

date -T → It displays the time in the format of hh:mm:SS i.e., 09:26:15

date -m → It prints only month value i.e., 06 (1-12)

date -h → To print only month name i.e., June.

date +%m -%h → 06 June

3) Echo:- It is used to display a state or message in Shell script. we have 2 ways which is used in Shell script. 1) To display a message.

$ echo "message". (or) $ echo message

2) To evaluate shell variable

Syntax:-

$echo $variable

By using $ represents adress of that variable

Ex:- $echo $a

There are some differences in echo variable across the shell most of these differences relate to the way echo interprets certain strings knowns as escape sequences.

Syntax: $ echo -e "\a"

Escape sequences		Description
1) \a	–	Bell sound
2) \b	–	backspace (Space bar)
3) \c	–	nonew line (cursor in a same line at current point)
4) \f	–	form feed
5) \r	–	carriage returns
6) \n	–	new line
7) \t	–	tab space

8) \v – vertical tab

9) \\ – Back slash (To print it)

10) \0n (zero n) – ASCII character represents by the octal value n (It can exceed 0377 (octal) and in (Decimal) 255)

11) printf :-

The printf command is evaluable on unix system and is the one you should use instead of echo. printf also accepts all escape sequences used by echo.

Syntax :- printf ("statement %d", a);

Escape sequence	Description
%s	– string
%30s	– It prints a string in a space 30 characters wide
%d	– Integer (or) decimal
%6d	– It prints a decimal in 6 characters wide
%o	– octal
%x	– Hexa decimal

22/6/17

Basic Calculator (BC):-

Unix provides 2 types of Calculators.

1) A graphical object that looks like the text based command.
2) xcalc is available in the window System and it is quite easy to use.
3) BC is less friendly, extremely powerful and remains one of the System's negleted tools.
4) To quit bc then type ctrl+d.

Syntax:- $ bc ↵

Ex:- 10+5
25
ctrl+d ↵
$

Man:- It is also used for general purpose. Command to display the manual instructions of a Command.

Syntax:- $ man - Command.

Ex:- $ man date ↵

-m
-H
-D
-h

} description will be displayed

all options of date

passwd:

It is used to change the password immediately.

Syntax:- $ passwd.

Ex:- $ passwd ↵

Passwd: changing password for user.

Enter login Id:
old password:
new password:
Re-enter password:
Passwd (~~system~~ SYSTEM): password Sucessfully changed for user.

Who:- Unix maintains an account of all users who are logged on to the systems. The who command display an information listing of systems.

System:- $ who ↵

Uname:- The uname command displays certain features of o.s running on a machine under the username.

Syntax:- $ uname ↵

o/p:- Sun os

1) uname -r:- It displays version of operating system.

2) Uname :- It displays domain name of OS ↓ company details

Satya scoves

tty (Teletype terminal) :- terminal → Endpoint (or) file. The tty command is an obvious reference to the device that has now become absolute. This command is simple and needs no argument.

Syntax :- $ tty.

i) tty -l :- It will prints synchronous lines prints.

ii) tty -s :- It will return only codes.

0 (or) 1.

0 → It Indicates terminal

1 → It indicates not a terminal.

Clear :-

It is used to clear the console and the cursor blinks at starting point.

Syntax :- $ clear ↵

23/6/17

2) file utilities :-

i) pwd :- (present working directory)

It displays the absolute path name which prints the current working directory.

Syntax :- $ pwd ↵

ii) cd (change directory) :-

It is used to change the current directory in to specified directory.

Syntax :- $ cd [directory] ↵

Ex :- $ cd D:

iii) mkdir :-

Directories are created with mkdir command. The command is followed by the name of the directory to be created.

Syntax :- $ mkdir ~~Aditya~~ ↵ [directory name] ↵

Ex :- $ mkdir Aditya ↵

C:\>$

C:Aditya\>$

iv) ls :- (listing of directories and files)

It is used to list the files of our directory

Syntax :- $ ls ↵

To get the list of files starts and ends with a characters we can get

information while using wild card *.

Syntax:- $ ls character*

// listing of files starts with a specified character.

$ ls *character

// listing of files ends with a specified character.

Ex:- $ls S* ↵

$ls *s ↵

note:- hidden files are not displayed with simple ls command.

options	description
-x	- It displays multi columnar output.
-F	- It marks executable with *, directories with '/' and symbolic links with @.
-a	- It shows all hidden files (or) dire all files begining with a character 'a'.
-R	- It displays Recursive list.
-r	- It sorts file names in Reverse order using ASCII value.
-l	- long listing in ASCII collating sequence showing 7 attributes of a file.
-d dirname	- It lists only directory name files.
-t	- It sorts file names by last modification time.
-lt	- It sorts long listing of files with last modification time.
-u	- It sorts filename by last access time.
-lu	- It sorts long listing of files by last access time.
-lut	- long listing of files sorted by last access time
-i	- It displays Inode number.
-R/more	- It list all the files and files in all directories in one page at a time.

↓ Cat:- It is used to display and create a file. Cat is one of the most well known used to display the content of a small file on the terminal.

Syntax:- $ cat filename ↵

Ex:- $ cat ex1 ↵

.....

.....

Ctrl+d

options:-

1) cat -v:- It is used for displaying text files only.

2) cat -n:- It is used to display numbers to the lines of a text files

note:- press Ctrl+d to come out of from the cat command environment.

note:- '>' symbol is used to create a file if it is not exist in the directory.

$ cat > filename.

24/6/17

vi, cp:- It is used copy a group of files (or) a file in to another file.

$ cp file1 file2

data → copied into file2

($ cat > add.txt → to create a new file (or) add text to that file

$ cat file1 → to display content)

Syntax: $rm -i file1 file2

-R (or) -r :-

Normally Rm cmd wont remove directories directly but with this option it will. it behaves Partially like with $rm dir.

-f :-

rm prompts for removal if a file is write protection. The -f option overrides. this minor Protection and forces removal.

Note :-

If the -r cmd combines -f it could be most risk thing to do. It deletes everything in current dir and below.

Syntax: $rm -rf * ⟶ don't use.

mv :- The mv commands renames the files it has 2 distinct functions.

(i) It renames a file or directory.

(ii) It moves a group of files to a different directory.

Syntax: $mv file1 file2

Ex: $mv adi1 adi2

Note :- mv simply replaces the filename in the existing directory with the newname.

more :

more command will display a page at a time and Then wait for input which is after spacebar

Syntax: $more filename more.

Note :- here the data print in page wise. we can see the next page by using spacebar.

lp :-

The lp command prints a singleed copy of the file.

Syntax :- $lp filename.

file :-

The file command to determine the type of file especially. of an ordinary file it can be used with one or more file names as outputs.

Syntax: $file filename

Eg: $file archieve.zip

archieve.zip : zip archieve

wc :- (word count) :-

The wc command is used to count no. of lines & words

syntax : $wc filename

Eg: $wc big.c ↵

option	Description
-l	* a line is any group of characters not containing a new line character & word is in group of characters n)o
-w	* a word is in group of characters not containing a space, tab or new line.

Syntax :-

i. $ wc -l filename → no. of lines count
ii. $ wc -w filename → no. of words count

13. od :- used to display the data of file in terms of octal representation.

Syntax : $ od filename.

Option	Description
-b	It trims the data and displays this value for each character.
-c	which displays the character with respective to octal value.

Syntax : $ od -bc
Ex : white

14. cmp :- used to compare two files or more to find whether they are identical or not so, one after can be deleted.

Syntax : $ cmp file1 file2

15. comm :- used to list the differing entries in different columns and it refer to sorted files

Syntax :- $ comm file1 file2

16. diff :- used to display the file differences

syntax : $ diff file1 file2.

* Backup utilities :-

Compressing and archieving files :-

Compress means to reduce the size of a file archieve means grouping set of files into a file or grouping directory into another directory.

1. gzip and gunzip :-

To compress files gzip is used and to uncompress files gunzip is used.

Syntax : $ gzip file1 file2 file3...

2. gzip -r :- It compresses all files into a folder or directory.

3. gzip -d (or) gunzip :- It is used to unzip the directory or uncompress the file.

File Security permission :- For security purpose user can provide permission to the file of directory. the access permission can be provided for the owner are

1. Read
2. Write
3. Execute

While using ls -l command it shows the access right.

file ownership :- When the System adminstrator creates a user account we have to assign these parameter to the user

[UID] → It maintains both its name & numeric representation

[GID] → It maintain both its name

File permission :- UNIX follows a 3 tier file production System that determines a file access the 3 groups are maintain to provide access permission to a file.

1st group	2nd group	3rd group
rwx	rwx	rwx
owner/user	group	other
UID	GID	
Ex: r w x	r - x	- - x
user	group	other

chmod (change mode):- To control the access permission for a file different users chmod is used chmod It can be accessed by only owner. To change the permission for all categories of user the file owner has to indicate.

1. Which permission is to be changed.
2. Whether the permission is to be assigned (OR) removed.
3. The file for which the permission are to be changed.

Syntax:- $ chmod [permission] file name

Ex:- chmod g+rwx file 1 green color is file 1
chmod u+rwx file 2
chmod o+x file 3

4. for changing file access permission [FAP] for owner only use the following syntax

Syntax:- $ chmod u+ [permission] file name

Absolute permission:- The Expression used by chmod is a string of 3 often numbers. If represents permission of each categories by one often digit can be define as below

Binary	octal	permission	significance.
000	0	- - -	No permission
001	1	- - x	only Execution
010	2	- w -	only write
011	3	- w x	write & Execute
100	4	r - -	only read
101	5	r - x	both read & Execute
110	6	r w -	both read & Execute
111	7	r w x	read, write, Execute

① 7 3 2
111 011 010
rwx -wx -w-

② 5 2 1
101 010 001
w-x -w- --x

③ 7 5 2
110 101 010
rwx r-x -w-

④ 6 3 4
110 011 100
rw- -wx r--

Chown (change owner):- Chown transfers ownership of a file to user, & it seems that it can optionally change the group as well. changing ownership requires super user permission, so, Let first change status to the Super user with the command "su"

Syntax:- $ su ↵
passwd ****

After the password successfully enters SU return a # prompt used by they route to renounce the ownership of the file use chown in the following way

Step 1 :- Enter # ls -l

step 2 :- # chown new user file name

step 3 :- # ls -l ↵ file name

step 4 :- # Exit

chgrp (change group) :- By default the group owner of a file is the group to which the owner belongs the chgrp owner changes a files group owner

Syntax :- $ chgrp new group name file name

Ex :- $ chgrp group1 file1.txt

ls -l for checking.

3/7/17

process utilities :-

A process is simply an instance of a Running program. It is said to be born when the program starts executes and remains alive as long as the program is active. After execution is Completed the process is said

min - 0 - low
max - 10 - high
middle - 5 - normal

kill:- kill is a utility that sends a signal to the process you identify it is used to kill any process that the user can own. In addition, the super user ~~can~~ kill almost any process on the system. The kill command can kills the signal with process Id.

Syntax:- $kill Signal PId↵ → To kill single process

$kill Signal PID PID PID↵ → To kill multiple processes.

bg:- It process a Job in to background

Syntax:- $bg Command↵

Ex:- $bg ls -l↵

fg:- It brings a Command execution in to foreground.

Syntax:- $fg Command↵

Ex:- $fg ls -l↵

pkill:- To terminate more than one process at a time pkill is used. In this we don't use pId it can be identified with process name.

Syntax:- ~~$ pkill~~

$pkill Signal processname processname processname↵

Disk utilities:-

The utilities which will display the information about the disk is known as the following mentioned commands are used Common in disk utilities

1. df
2. du
3. mount
4. umount.

1) df:- It is used to report the no. of free disk blocks and files. The units are measured in the form of total kilo bytes.

Syntax:- $df [option]

-h:- It displays sizes in earlier to read units.

4/7/17

du:-

It is used to summarize disk space allocated to each other (Each directory) this will give you the size of the disk in kilobytes used per directory and includes sub total for each directory.

Note:- du command without any argument displays disk usage of all files, sub-directories of current working directory

Syntax:- $ du <directory name> ↵

mount:-

This action makes a file system accessable to the system and to the user. The mount Command not only lists which file systems are currently mounted it also provides you as the root user with a method of mounting file system

Syntax:- #mount <Path>

Ex:- #mount C:\programfiles\Java\bin C:\programfiles\Java\lib

umount:-

The unmounting process to detach the filesys-tem from its mount point in the directory hierarchy. This action makes a file system unavailable to users. This is used by only super user (administrator)

Syntax:- #umount [option] mount point.

option	description
-f	- This will unmount the filesystem from disk and it will be not available for any of the user.
all	- This will unmount all the available mount points can perform the unmounting of file system.

Ex:- #umount -f <directory path>

Network utilities:-

1) Remote login:-

File Attributes:-

ln:- It is used to provide link b/w two files. It is of two types

1) Symbolic link
2) hard link

hard link:-

In hard link the original file Inode no. allocated to the link in the same block we can delete original file that can be accessed by links and it can access the file by using Inode number and the link provided by the original file.

Note:- Hard link does not provide links b/w directories.

Syntax:- $ ls file1 file2

Symbolic link:-

Symbolic link is provided b/w 2 files with different Inode numbers. It seperates blocks in the disks is allocated to link b/w multiple files. We can access data from link file or original file. If we delete original file we cannot access the content of the original file with symbolic link. It can also creates links b/w directories.

Syntax:- $ ls -s file1 file2.

Search/filter Command:-

grep:- (globally search regular expression and print) grep is case sensitive.

This command is used to search the string (or) expression in a file and prints no. of lines those contains the searched string (or) expression.

Syntax:- $ grep exp filename.
(or)
String

Ex:- $ grep "This" file1.

options	description
[ ]	- It prints the character in both cases Ex:- "[Tt]his".
^	- It prints the lines which starts with a given string. Ex:- "^The".
$	- It is used to print no. of lines which ends with a Search 'keyword. Ex:- "stop$".

grep is categorized in to 2 types.

1) fgrep
2) egrep

1) fgrep:-

It is used to search the set of strings in a file and print lines those contains any one of the given strings, on the strings in the command are seperated by new line.

Syntax:- fgrep 'str1'↵
'str2'↵
'str3' ↵ filename
(space)

2) egrep:-

egrep is resemble of fgrep but strings are seperated by the delimeter pipe(|)

Syntax:- egrep 'str1' | 'str2' | 'str3' ↵ filename
(space)

Unique:-

This command used to print lines unit time even though those lines repeated more than one time in the file.

Syntax:- $uniq filename↵

-c:- This command will print the line with count. This count represents no. of times the line is repeated in the file

Syntax:- $uniq -c filename

Ex:- $uniq -c file1

Cut:- This command is used to display required fields or words from a file.

Syntax:- $cut [option] filename

options	description
-c	- It is used to cut required data column wise.
-f	- It displays fields/words of every line in the files.
-d	- This command is used for seperation of words.

Ex:- $cut -c 1,3 file1

$cut -f 1,3 file1

$cut -d "delimeter" -f filename

Ex: :,,,;--

Ex:- $cut -d ":" -f file1

o/p:- This : is : Aditya

file1

This is Aditya

Paste:-

By using paste command we can add two files Vertically.

Syntax:- $paste file1 file2

Ex:- $paste f1.txt f2.txt

f1.txt

Sno	Sname
1	abc

f2.txt

mobile
9xxxx

o/p:-

Sno	name	mobile
1	abc	9xxx

option:-

-d:- It is used to seperate the word with delimeter b/w merged file data

Syntax:- $paste -d "delimeter" file1 file2.

(or) seperator

-s:- It is used to merge subsequent lines from one file.

7/7/17

Sort:- It is used to sort of file in alphabetical order

Syntax:- $sort [option] filename

options	description
-c	- It checks whether the file is already sorted or not.
-b	- It ignores spaces & tabs.
-d	- It ignores punchuations (,.;:")
-i	- It ignores non-printable characters.
-n	- It sorts all the lines of file in arithematic order.

Wed Sep 6 13:28:35 IST 2017

↓ 12 ↓↓ 1920

tr:- This command can be used to replace a character with another character with in the file.

Syntax:- $ tr 'replaced character' 'new character' filename ↵

Ex:- $ tr '[a-z]' '[A-Z]' file1.txt.

pr:-
It is used to prepare a file for printing by adding suitable setting and formatted text.

Syntax:- $ pr filename

option:-

-l:- It is used to print no. of line in a page from the file.

Syntax:- $ pr -l 54 file1 → To adjust 54 lines for a page

-t:- It is used to provide tabspace b/w words or characters.

Syntax:- $ pr -t 3 file1 → To give 3 tab spaces and go to next line

n(=1,2,3---):-
It defines no. of pages to be printed in one time.

Syntax:- $ pr +10 file1 → To print 10 lines of file1

Sed (Stream editor):-
Sed combines both interactive and non-interactive operations on data stream (add, modify, delete, etc.,). It performs several operations with the help of filters.

options:-

q:- To quit

-n:- It prints range of lines from stream editor.

-e:- It is used to combine multiple operations

Ex:- $ sed -3q file1 → It prints 3 lines and then it quit.

$ sed -n '1$p' file1 → To print all lines using $

$ sed -n '7,11p' file1 → To print from 7 to 11 lines

$ sed -n '3,5p' -e '7,11p' -e '20,23p' file1 → To combine multi actions by -e.

Inserting lines:-
"i" is used to insert lines by using sed command.

Note:- we can insert lines at the place of cursor.

Syntax:- $sed 'i' 'text' filename.

Ex:- $sed '4i Aditya saiAditya' file1.

deleting lines:-

By using "d" option it deletes files (or) directories.

Syntax:- $sed '10d' filename → To delete no. of lines in file

(or)

$sed 'directory/d' filen → To delete a direct

(or)

$sed 'filename/d' → To delete a file

10/7/17

Network Utilities:-

1, Remote login:- Remote login is used to provide network connection b/w multiple system it can be accessed with the help of TCP/IP.

In Remote login we are having 2 logins

→ 1) rlogin

→ 2) Telnet

These two commands are providing the Remote login.

Syntax:- $ rlogin Ipaddress

$ Telnet Ipaddress

Note:- rlogin and Telnet are same commands which provides accessibility of network whether it satisfies Credentials (username & password) by the administrator for the Ip address can be formed by the command IP Corney in Dos.

Ftp (file transfer protocol):-

This protocol is used to transfer file between hosts.

Syntax:- $ftp remote Ipaddress && (or)

$ftp remote hostname.

finger:-

It is used to knows the information about the user whether he/she log in to the System (or) not.

17/7/17

Syntax:- $ finger name.

Re-direction operators:-

18/7 Shell programming:-

i) Open an Editor to write a program such a c, shell.

ii) Type the syntax vi filename.extension (vi filename.c / vi filename.sh)

iii) press "i" button to convert ~~or~~ the Console from normal mode to insert mode.

iv) Then write the program in to editor and press ESC to come out of from the insert mode

v) Type :wq to save and quit from the editor to the Command prompt.

Execute the program by the following

Syntax sh filename.sh (or) cc filename.c
:wq ./a.out filename↵
:q! → without save

Shell variables:-

* In scripts, the variables are not defined by a datatype. By default it is treated as character string
* By using shell we can perform integer operation and comparision.

$a = 10

* for comparision operators

<	-lt
>	-gt
<=	-le
>=	-ge
!=	-ne
==	-eq

Types of variables:-

1) User defined variables
2) Environment variables
3) positional parameters

Rules for user defined variables:-

The user can create a variable name with combination of alphabhumeric & underscore.
The variable must start with alphabet.
The variable should not contain special symbol such as /, :, etc., -

19/7/17

Environment variables:-

The variable which are effect the behaviour shell (or) user interface. They are path and manpath. All the commands are placed in \bin (or) \user\bin.
Each process as an environment that considered as Group of variables that holds information. The information is reference to process. updating (or) adding new shell variables causes the updation of its environment to its child process.
PATH variable is used to locate all executable files of a Command which is typed by an user.

1) file substitution :-

ls * :-
It substutes all the files in the current directory. Ex:- ls *

ls ? :-
It represents a character in file name.
Ex:- ls a? & ls a?x?

ls [K-M] :-
It displays all the files which is having the initial character ranges from K to M in current directory

ls [!a-b] :-
It represents the file whose name not start with a or b from current directory

2) I/o Redirection :-

< :-
It is used for standard input.
Syntax:- Cat ex_1 < ex_2

>:-

Syntax:- cat >ex1

>>:-

It is used to append the text to another file

Syntax:- cat ex2>> ex1

<<:-

This Command represents the data is available at a particular line Command.

Syntax:- grep "The" << text

3) process Execution:-

;:-

Ex:- cat file1 ; ls ; rm file

( ):-

It is used execute multiple process (Command) in single point.

( ):-

A sub shell is started to execute this Command. After completion it terminates from Control and enters into current shell.

Ex:- shell program

```
- - - -
- - - - -
(cat ex1) // function calling
- - - -
- - - - -
```

&:-

It is used to place the Job background.

21/7/17

&&:-

It is used to execute multiple commands

Syntax:- $Command1 && Command2

Here Command1 executes succesfully then Command2 executes.

||:-

Syntax:- $Cmd1 || Cmd2

Here Command2 executes when only Cmd1 fails. If Cmd1 executes then Cmd2 never executes.

Quoting metacharacters:-

\:-

It is used to place a real characteristics of multiplication (*).

' ':-

It is used to display characters.

" ":-
It is used to print statements along with echo command.

` `:- (Back quote)
It replace command that encloses with an expression output.
Ex:- `expr $i + 1`

Special Characters:-

$0:-
It defines the position of the command.

$*:-
It list outs all the commands line arguments as a string.

./a.out part1 → command line argument.

$#:-
It indicates no. of command line argument.

$@:-
It indicates list of command line arguments and each quoted string treated as seperate argument.

$$:-
It indicate PID of current shell.

$?:-
It defines the status of recent executed command.

$!:-
It defines PID of most recent background job.

Shell Command:-

1. echo:-
It is used to print input and output stmts, values of variables in shell programs.
Syntax:- $echo $a
$echo "welcome"

2. expr:-
It is used to execute arithematic operations.
Syntax:- `expr $a + $b`

3. #:-
It is used for single line comment.

4. \n:-
It is used for to jump the position to new line.

5. \c:-
It is used to provide the cursor blinking at the starting position.

6. \b:-
It is used for carriage return.

22/7/17

Control Statements:-

1) Conditional Statements:-

Simple if:-

```
if [expr/condition]
then
   Statements
fi
```

if else:-

```
if [expr/condition]
then
   Statements
else
   statements
fi
```

if-else-if:-

```
if [expr/condition]
then
   Statements
elif [expr/condition]
then
   Statements
fi
```

Nested if:-

```
if [expr/condition]
then
   if [expr/condition]
   then
      statements
   else
      statements
fi
else
   statements
fi
```

Write a program to print a number Should be non-zero.

C:-

```
#include<stdio.h>
main()
{
int n;
printf("enter n value");
scanf("%d",&n);
if(n!=0)
printf("%d",n);
}
```

Shell:- file1.sh

```
echo "enter n value"
read n
if [$n -ne 0]
then
echo "The no. is $n"
fi
```

o/p:-

```
sh file1.sh
enter n value 5
the no. is 5
sh file1.sh
enter n value
0
$
```

write shell program to check the given number is whole number. wholen.sh

C:-

```
#include<Stdio.h>
main()
{
int n;
printf("enter n value");
Scanf("%d",&n);
if(n>=0)
printf("%d is a whole number",n);
else
print f("not a wholenumber");
}
```

Shell:-

```
echo "enter n value"
read n
if [$n -ge 0]
then
echo "The no. $n is wholenumber"
else
echo "The not a wholenumber"
fi
```

o/p:- enter n value
-5
not a wholeno.
enter n value
6
the no. 6 is wholeno.

3. write a Shell program the biggest no. of 3 given numbers (nested if). big3.sh

C:-

```
main()
{int a,b,c;
printf("Enter the a,b,c values");
Scanf("%d%d%d",&a,&b,&c);
if(a>b)
{ if(c>a)
printf("%d c is bigger",c);
else
printf("%d A is bigger");
}
else if(b>c)
printf("%d B is bigger");
else
printf("c is big");
}
```

o/p:- enter a,b,c values
7 5 6
7 is big

Shell program:-

```
echo "enter a,b,c values"
read a b c
if [$a -gt $b]
then
if [$c -gt $a]
then
echo "$c is big"
else
echo "$a is big"
```

```
fi
elif [$b -gt $c]
then
  echo "The biggest value is $b"
else
  echo "$c is bigger
fi
```

4) write a shell program to display the value of a no. and check +ve or -ve. (if-else-if).

prn.sh

C:-

```
main()
{
  int c;
  printf("Enter n value");
  scanf("%d", &n);
  if(n>0)
  printf("n is +ve integer");
  else if(n<0)
  printf("n is -ve integer");
  else
  printf("The n value is zero");
}
```

Shell program:-

```
echo "enter n value"
read n
if [$n -gt 0]
then
  echo "The n is +ve"
elif [$n -lt 0]
then
  echo "The nvalue is negative"
else
  echo "The n is zero"
fi
```

o/p:-

```
enter nvalue
-6
n is negative
enter nvalue
0
n is 0
enter nvalue
5
n is positive
```

25/7/17

5) write a shell program to check whether the given no. is prime or not.

Case structure:- (in c it is switch)

```
Case value in ch.....
Choice 1) ____
          ____
          ;; // break
Choice 2) ____
          ____
          ;;
Choice 3) ____
          ____
          ;;
  .
  .
  .
Choice *) ____
          ____
          ;; esac
```

While examples:-

* To print numbers upto given range (n numbers):-

num.c

```
echo "enter n value"
read n
i=1
while [$i -le $n]
do
 echo $i
 i=`expr $i + 1`
done
```

O/P:- enter n value
6
1
2
3
4
5
6

* Sum of 'n' numbers:-

sumn.sh

```
echo "enter n value"
read n
i=1
tot=0
while [$i -le $n]
do
 tot=`expr $tot + $i`
 i=`expr $i + 1`
done
echo "The sum of $n numbers: $tot"
```

O/P:- enter n value
12
the sum of 12 nos: 78

* factorial of a number:-

factorial.sh

```
echo "enter n value"
read n
f=1
while [$n -gt 1]
do
 f=`expr $f \* $n`
 n=`expr $n - 1`
done
echo "factorial is :$f"
```

O/P:- enter n value
6
factorial is: 720

* Reverse of a number:-

revnum.sh

```
echo "enter n value"
read n
s=0, i=0
while [$n -gt 0]
do
 i=`expr $n % 10`
 s=`expr ($s \* 10) + $i`
 n=`expr n / 10`
done
echo "The reverse of a number : $s"
```

* Perfect number:-

```
echo "enter a number"
read n
i=1
s=0
while [$i -le `expr $n / 2`]
do
 if [`expr $n % $i` -eq 0]
 then
  s=`expr $s+$i`
 fi
```

26/7/17

Unit - II

Unix files

*unix file structure :-

*unix file Architecture heirarchy :-

System Calls :-

1) creat :-

Syntax :- int creat (char *path, mode_t mode)

Ex :- int fd → file descriptor.

fd = creat ("file1", 0664)

Modes :-

Binary format for others :-

Decimal	accessmode	mode
0	000 (---)	—
1	001 (--x)	S_IXOTH
2	010 (-w-)	S_IWOTH
3	011 (-wx)	S_IWXOTH
4	100 (r--)	S_IROTH
5	101 (r-x)	S_IRXOTH
6	110 (rw-)	S_IRWOTH
7	111 (rwx)	S_IRWXOTH

for Group :-	for User :-
S_IWGRP	S_IWUSR
S_IRGRP	S_IRUSR
S_IXGRP	S_IXUSR

Ex :- S_IRWXUSR = S_IRUSR | S_IWUSR | S_IXUSR

Note :- If fd = +ve value it is successful in creating a file.

2) fd = -1 there is an error while creating a file1.

2) open :-

Syntax :- int open (char *path, O_flag);

Ex :- int fd;
fd = open ("file1", O_RDONLY);

flag values :-

O_RDONLY
O_WRONLY
O_TRUNC
O_CREAT

28/11/24

3) Read :- & write :-

Syntax :- int read (fd, char *buffer, unsigned bytes);

Syntax :-

int ~~read~~ write (fd, char *buffer, unsigned bytes);

4) close :-

Syntax :- close (fd);

5) Header files :-

1) #include <fcntl.h> → To access flags for file operation.
2) #include <sys/types.h> → It is used to set types for file operation.

6) lseek :- (unformated - which focuses on data and not focuses on any other special characters like EOF)

Syntax :-

lseek (int fd, long offset, int whence);

offset values :-

offset

#define L_SET → defines starting position, here whence = 0
#define L_INCR → Current position, whence = 1
#define L_XTND → End position, whence = 2
#define SEEK_SET → whence = 0
#define SEEK_CUR → whence = 1
#define SEEK_END → " = 2

whence	description
0	offset bytes starts into the file
1	Current position in the file offset
2	End of the file

8/12

7) dup, dup2 :-

Syntax :- int fd1, fd2;

fd2 = dup (fd1);

The dup system call duplicates an

Open file descriptor and it returns to new file descriptor

Note:-
It has the same file pointer i.e., Both file descriptors share one file pointer with same access permissions.

dup2:-
dup2 is also provides duplication of files along with multiple file descriptors

Syntax:- int dup2(fd1, fd2); (fd1 = old, fd2 = new)

1) If it returns +ve value duplication is successful
2) If it returns -1 there is an error while duplicating.

8) Stat & fstat():- Structure of stat:-

```
Struct stat
{
dev_t    st_dev   -> device files
ino_t    st_ino   -> returns inode
mode_t   st_mode  -> octal no
short    st_link  -> which link i.e., hard (or) symbolic
uid_t    st_uid   -> user id
gid_t    st_gid   -> group id
dev_t    st_rdev  -> revised devise
off_t    st_size  -> offset value
time_t   st_atime -> last access time
time_t   st_mtime -> modified time
time_t   st_ctime -> compile time
long     st_blocksize -> file data size is defined
};
```

Syntax:- stat(*) & fstat():-

```
fd=open("filename", O_RDONLY | O_WRONLY);
Struct stat statvariable;
int stat(char *filename, &statvariable);
int fstat(int fd, &statvariable);
```

Program:-
write a program using the system calls create, open, read, write, close and lseek.
syscal.c

```
#include<fcntl.h>
#include<sys/types.h>
#include<stdio.h>
#include<sys/stat.h>
#include<string.h>
#define BUFSIZE 1024
int main()
{ char buf1[BUFSIZE];
  Struct stat x,y;
  int fd1,fd2,n;
  char buf[]="This is Sample buffer";
  Char msg[]="This is added text in the file";
```

```
fd1 = Creat("test.txt", 0644);
if(fd1 == -1)
{
  perror("error");
  exit(-1); → waming: incompatible implicit declaration of built-in function 'exit'
}
else
{
  printf("file sucessfully created.\n fd=%d", fd1);
  fd2 = open("test.txt", O_WRONLY | O_RDONLY);
  write(fd2, buf, strlen(buf));
  lseek(fd2, 0, SEEK_END);
  write(fd2, msg, strlen(msg));
  close(fd2);
  fd2 = open("test.txt", O_RDONLY);
  while(n = read(fd2, buf1, BUFSIZE) > 0)
  printf("The content in the file is:\n%s\n", buf1);
  close(fd2);
}
}
```

op:- ./a.out sysc1.c ↵
file sucessfully created.
fd=3
content in the file is :
this is sample buffer this is added text in the file.

3) write a shell script for sorting, searching and insertion and ~~delion~~ deletion of a element in a list.

```
echo "Enter filename"
read fn
echo 1.Sorting 2.Searching 3.Deletion
read a
case $a in
  1) sort $fn ;;
  2) echo "Enter search keyword"
     read word
     grep "$word" $fn ;;
  3) echo "Enter word"
     read word1
     grep -r "$word1" $fn ;;
  *) echo "Enter possible value"
  esac
```

write a program to create which reads source filename and destination filename using command line arguments and then converts the data of a file into specific format i.e. from lower case to upper case and vice versa.

C program:-

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

```
#include<unistd.h>
#include<fcntl.h>
int main()
{
int fd1, fd2, n, i;
char buf[30];
fd1 = open("text file1", O_RDONLY);   // should be already created.
fd2 = open("filename file2", O_CREAT | O_RDONLY | O_TRUNC | O_WRONLY);  // already created or no need
while((n = read(fd1, buf, 30)) > 0)
{
for(i=0; i<n; i++)
{
if((buf[i] >= 97) && (buf[i] <= 122))
buf[i] = buf[i] - 32;
}
write(fd2, buf, n);
}
return 0;
}
```

```
$ cat file2
THIS IS ADITYA COLLEGE
2 MCA 3RD SEM
THIS IS RAVALI
```

Write a program to demonstrate Stat system call.

```
#include<sys/stat.h>
#include<sys/types.h>
#include<stdio.h>
#include<unistd.h>
int main(int argc, char *argv[])
{
struct stat buff;
char filename[30];
printf("enter filename");
scanf("%s", &filename);
if(stat(filename, &buff) == -1)
{
perror("filename");
exit(1);
}
else
{
printf("filename = %d\n", buff.st_uid);
printf("Groupowner = %d\n", buff.st_gid);
printf("Inode number = %d\n", buff.st_ino);
}
exit(0);
}
```

```
o/p:- a.out statp.c
enter filename
file1
User id = 507
group Owner = 507
Inode number = 260995
```

5/8/17

Directory System Calls:-

```
int mkdir(char *path, mode_t mode)
int rmdir(char *path)
int chdir(char *path)
char *getcwd(char *buf, int size)
int opendir(char *path)
int readdir(decriptor)
closedir(struct dir*d)
```

Directory structure:-

```
type def struct DIR
{
int dd_fd;
off_t dd_loc;
size_t dd_size;
struct dirent *dd_buf;
} DIR;
struct dirent
{
long d_ino;
kernal off_t d_off;
unsigned int d_reclen;
char d_name[256];
```

(Streames classes)

Standard I/O functions:-

1) opening Stream:-

```
FILE *fopen(char *path, char *mode)
FILE *fdopen(int fd, char *mode)
FILE *freopen(char *path, char *mode, FILE *stream)
```

r - read
w - write
r+ - read & write
w+ - write & read
a - append
a+ - append & read.

fstat():-

statp.c

```
#include<sys/stat.h>
#include<sys/types.h>
#include<stdio.h>
#include<unistd.h>
int main(int argc, char* argv[])
{
struct stat x;
int fd;
if(argc==1)
{
printf("Usage: %s [file]\n", argv[0]);
return -1;
```

fd = open(file);

```
if(fstat(fd,&x)<0)
{
  perror("fstat");
  return -1;
}
printf("size:%d\n inode:%d\n", x.st_size,
                               x.st_ino);
return 0;
}
```

o/p:- size:52 inode:2609995

17/8/17

Stream flushing:-

fflush:-

It is used to empty the buffer using stream class. It return a zero value on success and it returns ~~eof~~ EOF on failure

It is used to write the buffer of the stream to a device or a file.

Syntax:- int fflush(FILE *stream);

Remove & Rename:-

Syntax:- int remove(char *path);
int rename(char *oldpath, char *newpath);

The Remove System Call is used to remove the file from the directory.

The Rename Systemcall is used to change the name of the file with the existing name of file

The both of the system calls returns +ve integer on success. otherwise it returns -1.

Creating temperory file:-

Syntax:- FILE *tmpfile(void)

It is used to create temperory file and it returns a pointer to a file of stream which is considered as a temperory file. And it dissappears that magically goes away when our program is out of memory.(control)

It returns a null pointer.

Stream buffering:-

i.e., char by char data is displayed (eg) → unbuff eg) System Call

int SetBuf(FILE *stream, char *buf);

int Setbuffer(FILE *stream, char *buf, size_t size);

(eg) whole data is displayed at a time when we get size of file. eg) no buffer system call

int Setlinebuf(FILE *stream);

(eg) Line by line is loaded and then displayed.

int setvbuf(FILE *Stream, char *buf, int mode, size_t size);

It is used to access any of the above 3 System Calls.

mode → IONBF

30/8/17

Stream positioning :- fseek (formatted - which focus on data and also the ↓, IP, seeking information at a particular position)

```
int fseek(FILE *stream, long offset, int whence);
    SEEK_SET 0
    SEEK_CUR 1
    SEEK_END 2
```

position is identified using file stream

```
long ftell(FILE *stream) // return current position
void rewind(FILE *stream)   (to set cursor at starting)
int fgetpos(FILE *stream, fpos_t *pos);   (to get position of cursor using position pointer.)
int fsetpos(FILE *stream, fpos_t *pos);
```

Reading and writing of characters :- read → return data (char), writ → return type is int → it counts no. of characters → variable

```
char fgetc(FILE *stream);   // read → temperory data (buffer) → generate o/p easily
int fputc(char V, FILE *stream)   write → permanant data → to generate o/p → open the file
int
Char *gets(Char *s, int n, FILE *stream)   (size)
   ↳ to locate that buffer   ↳ pointing buffer   ↳ file size
int fputs(char *s, FILE * stream)
```

(char info/data)

get → read → temp data (buffer) → o/p easily (variable)

put → write → permanant data → o/p → open the file

(int size)

Unit - III

UNIX PROCESS

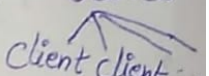
process is a instance of running program is called process.

→ each process is divided into segments (pages).

→ Every work is done by a process only, in o.s

~~→ A server process executes the client process to send request to server process request from client process from well known~~

→ The client process to send request to server process. A server process executes/running a process to listen request from client process from well known port. that process is called daemon process i.e., default process.



→ Server process executes client and also get Command from outside also.

→ Every process has a unique id i.e., PID.

→ PS Command shows presently execute process is shown.

PS-X it shows all present executes process with their attributes.

→ Every o.s has a process. of PCB (process control block).

→ The o.s represents as the PCB.

→ [illegible] will has its own address space

→ When a process ... process is called child process

→ When a process creates that process is called parent process

→ Every process needs a resources that should be allocated to by the o.s

fork syntax:-

```
pid_t fork();
```

Write a program to create two process to run a for loop which adds numbers 1 to n, say one process adds odd numbers and the other adds even numbers.

program:-

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
main()
{ int a,b,esum=0,osum=0,pid;
int i;
printf("enter a range");
scanf("%d %d",&a,&b);
pid=fork();
for(i=a;i<=b;i++)
{
 if(pid==0)
  {
   if(i%2==0)
   esum=esum+i;
  }
 else
  {
   if(i%2!=0)
    {
     osum=osum+i;
    }
  }
}
printf("\n pid=%d\n ****\n",pid);
printf("The sum of even numbers is :%d\n",esum);
printf("The sum of odd numbers is :%d\n",osum);
}
```

output:- enter ab range 1 21

```
pid = 19726
****
The sum of even no.s is :0
The sum of odd no.s is :121
pid = 0
****
The sum of even no.s is :110
The sum of odd no.s is :0
```

11/9/17

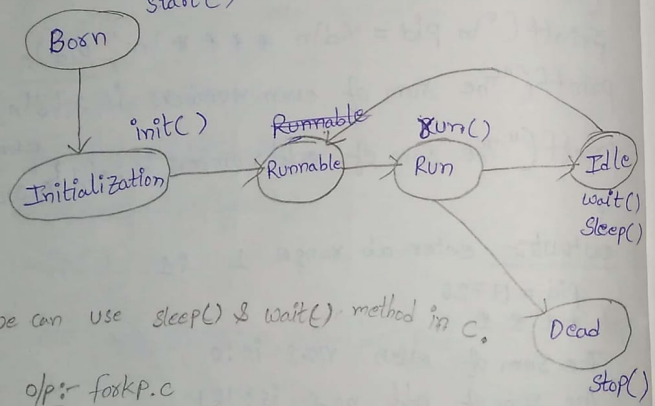
fork():- (fork, vfork)

pid_t fork()

Program:- (forkp.c)

```
#include<stdio.h>
Void main()
{ int pid;
  pid = fork();
  if(pid==0)
  printf("The child process is created");
  else
  printf("Child process is not created");
}
```

* Thread Life Cycle :-



we can use sleep() & wait() method in C.

o/p:- forkp.c

child process is not created 4637 (or) 13427

$ The child process is created 0.

12/9/17

Vfork():- The vfork() function creates a new process without fully copying the address space of the old process. (parent process).

When a new process is a create with vfork() the parent process is temperorily suspended and a child process might borrow the parent address space. This continues until child process executes.

Exit() (or) execve() Calls.

fork:-

parent

fork() | copy

child

Vfork():-

vfork()

parent — address

child.

Sample program:- cp fork.c

```
int main()
{
 int X=10;
 if(fork()==0)
 {
  sleep(10);
  X = X+10;
  printf("for child %d", X);
 else
  printf("from parent %d", X);
```

o/p:- from parent 10

$ for child 20

}

wait():-

The wait() call control execution of process this system call force the parent to suspend the execution until the child process finished this wait() call returns process Id of a child process that finish.

If child finishes before the parent calls then wait is called by the parent return immediately with a Error message.

Syntax :- (or) prototype :-

```
int wait (status)
int *status;
```

status is an integer where by unique system stores the value returned by the child process.

Record programs:-

1) write a program to display wishes depends ~~regarding~~ on the user logon time i.e., good morning, good afternoon, wishes.sh

program:-

```
b='date | cut -c12-20'
a='date | cut -c12-13'
if [ $a -le 12 ]
then
 echo "hai good morning"
 echo "current time is : $b"
elif [ $a -ge 12 -a $a -lt 15 ]
then
 echo "Good Afternoon"
 echo "current time is : $b"
elif [ $a -ge 16 -a $a -le 18 ]
then
 echo "Good evening"
 echo "current time is : $b"
else
 echo "Good night"
 echo "current time is : $b"
fi
```

o/p:- Good Afternoon
current time is : 14:01:27

2) write a shell script which works similar to wc command to count no. of lines and words without using wc. wwords.sh

```
line=0
word=0
echo "enter filename"
read fname
exec < $fname
while read a
do
line=`expr $line + 1`
set $a
word=`expr $word + $#`
done
echo "filename:" $fname
echo "*******"
echo "lines:" $line
echo "words:" $word
```

o/p: enter filename
files
filename: files
* * * *
lines: 3

3) write a shell script which delete all lines which containing the word unix in the files supplied as an argument to the shell script delline.sh

```
echo "enter filename"
read fname
grep -v 'unix' $fname
```

o/p: enter filename
unix
these is shell programming
are c programming

4) write a shell script which display a list of files in the current directory to which you have read, write and execute permissions. afatfile.sh

```
echo "Access permissions of a file in current directory"
for fn in `ls -l`
do
if [ -r $fn -a -w $fn -a -x $fn ]
then
echo "The file $fn has read, write execute permissions"
fi
done.
```

o/p: Access permissions of a file in current directory
the file abc has read, write and execute permissions
the file def has read, write and execute permissions

exec family function:- 1) execl:-

1) int execl (filename, arg0, arg1, arg2, ----, NULL);
   char *filename, arg0, arg1, -----

2) execv:-
   int execv (filename, argv)
   char *filename, * argv[]

3) execle:-
   int execle (filename, arg0, arg1, ---- NULL, envp);
   char *filename, arg0, arg1, ------, *envp[]

4) int execlp (filename, arg0, arg1, ----, NULL);

5) int execvp (filename, argv);

9/9/17

Execv function:- (Note:- observe the difference b/w the exec all these 4 functions.)

(pexecv.c)

```
#include<stdio.h>
#include<unistd.h>
main()
{
char *a[4];
a[0] = "ls";
a[1] = "-a";
a[2] = "-l";
a[3] = 0 (zero);
execv("\bin\ls", a);
}
```

execl:-

```
#include<stdio.h>
#include<unistd.h>
main()
{
printf("Before execl function\n");
execl("\bin\ls", "ls", "-a", "-l", 0(zero));
printf("after execl function\n");
exit(0);
}
```

execlp:-

```
#include<stdio.h>
#include<unistd.h>
main()
{
printf("Before execlp function\n");
execlp("ls", "-l", 0(zero));
}
```

```
#include<stdio.h>
#include<unistd.h>
main()
{
 char *a[4];
a[0]="ls";
a[1]="-a";
a[2]="-l";
a[3]=0;
execvp("ls",a);
}
```

17/9/17

Threads:-

handling multiple threads

Signals:-

Signal is called as Software Interrupt.

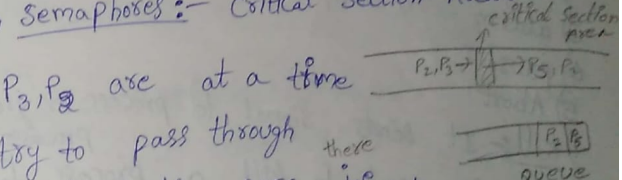
Types of Signals:-

→ To handle a process to create, open, and to manage process we use Signals Kill, P.Kill signals.

→ Thread Synchronization with resource allocation.

By sending a signal to Resource by the Thread. The reply will be in 0,1

→ Semaphores:- Critical section Area.



P_3, P_2 are at a time try to pass through there Critical section Area. i.e., known as Critical section Area problem.

Then a signal send to processes that one process should be go first and next the other.

The user defined signals are also generated while executing a process.

predefined signals are there is the header files in the <Signal.h> header file.

Syntax:-
int signal (SIGUSR1, userdefined signalname);
to create a userdefined signal

System calls to alert:-
1) kill
2) raise
3) alarm
4) pause
5) Abort
6) system
7) sleep function

1) kill:- It sends signal to process or groups of processes.
It is used to kill a process forcefully

Ex:- int kill (pid_t pid, int signo);

2) raise:-
It allows a process to send a signal to itself.

Ex:- int raise (int signo);

→ Both return: 0 if ok, -1 on error.
(or)
+ve integer.

3) Alarm:- #include<unistd.h>
Syntax:-
unsigned int alarm (unsigned int seconds);

4) pause:- To suspend a process for some time
Syntax:- #include<unistd.h>
int pause (void);
Returns -1 with errno set to EINTER.

5) Abort:-
It causes abnormal program termination
Syntax:- #include<stdlib.h>
void abort (void)
It never returns
It sends SIGABRT signal to the caller.

6) System:-
It is convenient to execute a command string from within a program.
Syntax:- #include<stdlib.h>
int system (const char * cmdstring);
It returns non-zero if a Command

processor is available.

7) Sleep function:-

Syntax:- unistd.h

~~unist~~

unsigned int sleep(unsigned int seconds);

program to demonstrate signal handler?

```
void sigusr (int signo)
{
 if (signo == SIGUSR1)
  {printf("The recieved signal is sigusr1");
 }
 else if (signo == SIGUSR2)
  {printf("Recieved signal is sigusr2");
 }
else printf("The recieved signal: Invalid");
 exit(0);
}
int main()
{
 if (signal(SIGUSR1, sigusr) == ~~SIG_U~~ SIG_ERR)
  {perror("SIG USR1");
   return -1;
 }
else
 if (signal(SIGUSR2, sigusr) == SIG_ERR)
 {perror("SIGUSR2");
  return -1;
}
for(;;)
pause();
return 0;
}
```

O/P: [0] 9024
kill -s SIGUSR1 9024
The recieved signal is sigusr1
kill -s SIGUSR1 9024
-bash: kill: Invalid signal specification
SIGUSRT

12.9.17 Unit-4

Part-I Memory Management

ANSI C specifies 3 functions of memory allocation:

```
#include<stdlib.h>
void *malloc(size_t size);
void *Calloc(size_t nobj, size_t size);
void *realloc(void *ptr, size_t newsize);
```

All 3 return: non-null pointer if OK, NULL on error.

```
void free(void *ptr);
```

The function free causes the space pointed to by ptr to be deallocated.

Calloc - it schedules the n jobs that which will be executed. first priority wise.

realloc - again allocates some more memory to the Jobs execution

→ The type of lock desired:

F_RDLCK

F_WRLCK

F_UNLCK

→ The starting byte offset of the region being locked or unlocked (l_start and l_whence)

Types of locks:-

1) Shared lock:-

Read locks are shared locks. Any no. of process can have read lock on the same object at a time.

2) Exclusive lock:-

write locks are exclusive locks.

Advisory locks:-

In unix, advisory locking is done with flock() and lockf() commands.

Both fcntl and flock offer advisory locking.

Advisory locking is locking that requires the cooperation participating processes.

advisory locking is sometimes called unenforcing locking.

Mandatory locking:-

forced locking. It forces a process to lock the file. The locks are Read or write

Deadlock:-

Deadlock occurs when two processes are waiting for a resource and no two processes are executing then it is known as Deadlock.

* IPC:-

There 5 applications

1) pipes
2) Named pipes (FIFO)
3) Message Queues
4) Shared memory
5) Semaphores

20/9/17

Part-II IPC

* Inter process Communication (IPC):-

1) Pipes:-

pipes have two limitations.

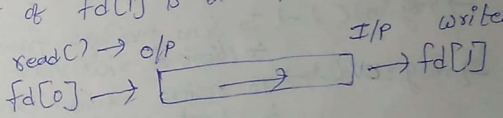
→ They have only the half duplex

→ Pipes can be used only between 2 processes

Syntax:- #include<unistd.h> #include<sys/stat.h>
#include<sys/pipe.h>
int pipe(int filedes[2]); #include<sys/ipc.h>
#include<sys/types.h>

Returns: 0 if ok, 1 on error

→ Two file descriptors are returned through the filedes argument: filedes[0] is open for read(), and fd[1] is open for writing. The output of fd[1] is the input for fd[0].



Popen and pclose:-

No need of Popen & pclose in this unix.

because

fd = pipe(fd[2]);

close (fd);

2) FIFOs:- (named pipes)

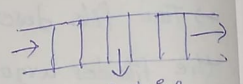
→ Pipes can be used only between related processes when a common ancestor has created.
With FIFOs, unrelated process can exchange data.

→ creating a fifo is similar to creating a file.

```
#include <sys/stat.h>
int mkfifo (const char *pathname, mode_t mode);
```

Returns : 0 if OK, -1 on error.

3) Message Queues:-



4 system calls

→ msgget() → it creates a message queue
→ msgsnd() → To send message from server to client or vice versa
→ msgrcv() → To recieve messages from server to client or vice versa.
→ msgctl() → To delete a message queue

* A message queue is a linked list of messages stored with in the kernel and identified by a message queue identifier.

we'll call the message queue just a queue and its identifier.

* A new queue is created or an existing queue.

* messages are fetched from a queue by msgrcv.

* Each queue has the following msqid_ds structure associated with it:

21/9/17

Data structure:- (ss)

```
struct msqid_ds → data structure
{
  struct ipc_perm msg_perm;   // permission structure (permissions provided by this. The access perm are oth, grp, usr & Read, write specify)
  msgqnum_t msg_qnum;         // queue maintained
  msglen_t msg_qbytes;        // queue size
  pid_t msg_lspid;            // last send process id
  pid_t msg_lrpid;            // last recieve process id
  time_t msg_stime;           // send time
  time_t msg_rtime;           // recieve time
}
```

Syntax for msgget():-

#include <sys/msg.h>
int msgget (key_t key, int flag);
(key provides communication b/w multiple system; key value to identify both sides)

Returns: message queue ID if OK, -1 on error

→ msgget is used to either open an existing queue or create a new queue.

→ On success, msgget returns the non-negative queue ID. This value is used with the other three msg queue functions.

msgctl():- (ctl - control)

→ The msgctl function performs various operations on a queue.

#include<sys/msg.h>
int msgctl(int msqid, int cmd, struct msqid_ds *buf);

→ Returns: 0 if OK, -1 on error

→ The cmd argument specifies the command to be performed on the queue specified by msqid.

→ IPC_STAT -- fetch the msqid_ds structure for this queue, storing it in the structure pointed to by buf.

→ IPC_SET --- Copy the following field from the structure pointed to by buf to the msqid_ds structure associated with this queue: msg_perm.uid, msg_perm.gid, msg_perm.mode, and msg_qbytes.

→ IPC_RMID -- Remove the message queue from the system and any data still on the queue. This remove is immediate.

22/9/17

msgsnd():-

→ Data is placed into a message queue by calling msgsnd.

#include<sys/msg.h> (provides comm b/w 2 programs to msg queue id)
int msgsnd(int msqid, const void *ptr, (ptr: msg sent using it.)
size_t nbytes, int flag);
(nbytes: size of msg; flag: send successful or not.)

Returns: 0 if OK, -1 on error.

→ when msgsnd returns successfully, the msqid_ds structure associated with the message queue is updated to indicate the process ID that made the call (msg_lspid),

the time that the call [illegible](msg_stime), and that one more message is on the queue (msg_qnum).

msgrcv():-

→ Messages are retrieved from a queue by msgrcv.

```
#include<sys/msg.h>
ssize_t msgrcv(int msqid, void *ptr,
        size_t nbytes, long type, int flag);
```

0/1 values

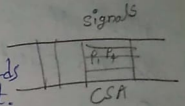
Returns: size of data portion of message if ok, -1 on error. 0 - not recieved msg

→ when msgrcv succeeds, the kernel updates the msqid_ds structure associated with the message queue to indicate the caller's process ID (msg_lrpid), the time of the call (msg_rtime), and that one less message is on the queue(msg_qnum).

3/10/17

* Semaphores:- Section Area the priority is given to process

→ If the value of the semaphore is 0, the process goes to the sleep until the semaphore value is greater than 0. when the process wakes up, it returns to step 1.



→ The kernel maintains a semid_ds structure for each semaphore set.

1) semget( ) → Creates a Semaphore
2) semop( )
3) semctl( )

System V :- message queue, Semaphore, Shared memory.
↓ Question indirectly asked in exams.

Structure:-

```
struct semid_ds
{
  struct ipc_perm sem_perm;
  unsigned short sem_nsems;   → no. of semphores
  time_t sem_otime;   → out time or insert time
  time_t sem_ctime;   → compile time
};
```

1) semget():-

→ The first function to call is semget to obtain a semaphore ID.

```
#include<sys/sem.h>
int semget(key_t key, int nsems, int flag);
```

It creates nsems for mpegs

which process is given priority

Returns: Semaphore ID if ok, -1 on error.

→ The no. of semaphores if a new set is being created (typically in the server), we must specify nsems. if we are referencing an existing set (a client), we can specify nsems as 0.

* Shared memory:- Segments form to store chesta...

→ Shared memory allows two or more processes to share given region of memory. This is the fastest form of IPC, because the data does not need to be copied b/w the client and the server.

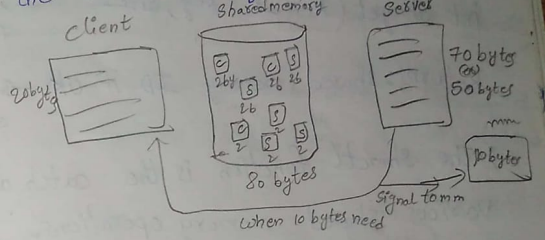
→ The kernel maintains a structure with at least the following memory members for each shared memory segment:

→ System calls for this are 4. They are

1) shmget( ) → To create
2) shmat( ) → attaching memory Segments
3) shmdt( ) → detatching memory Segments
4) shmctl( ) → to control all the segments.

→ If the memory is individually provided to client and server process we can get drawbacks. They are: 1) Memory wastage or 2) Data loss

Then the shared memory concept is introduced



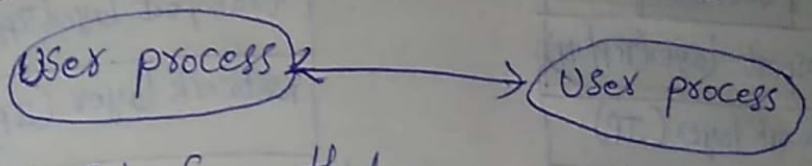
Structure:

```
Struct shmid_ds
{
  struct ipc_perm shm_perm;
  size_t shm_segsz;  → segment size i.e. no. of blocks in a segment
  pid_t shm_lpid;    → last accessing pid of process lastly used
  pid_t shm_cpid;    → last compilation pid.
  shmatt_t shm_nattch;  → no. of attachments
  time_t shm_atime;  → attached time
  time_t shm_dtime;  → detatched time
  time_t shm_ctime;  → compilation time
```

5/10/17

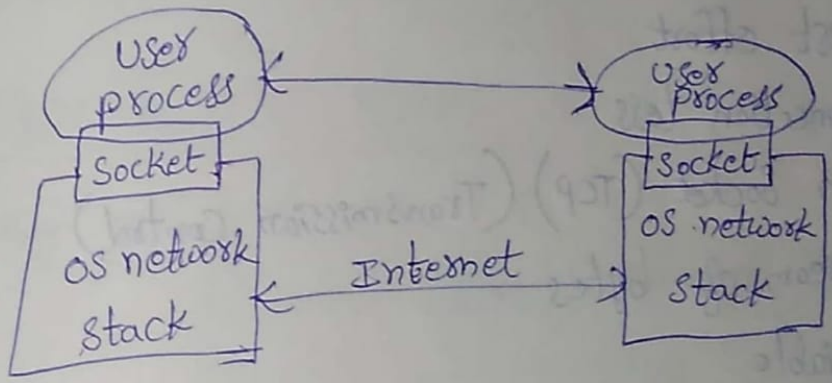
Sockets:-

Socket and Process Communication:-



The interface that OS provides to its networking subsystem.

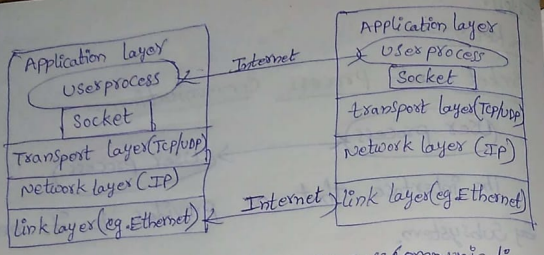
→ Socket is interface which provides communication b/w systems when they are in a network



→ The socket are used to involve the O.S of the systems.

→ The socket will change the system into compatible mode

[Types of networks → H/w, 4 → LAN, WAN, MAN, PAN, Internet, wireless]



Two types of Application process communication:-

→ Datagram Socket (UDP) (User datagram)
* Collection of messages
* Best effort
* Connection less

→ Stream Socket (TCP) (Transmission Control)
* Stream of bytes
* Reliable
* Connection oriented

Socket Identification:-

→ Communication protocol
* TCP (Stream Socket): streaming, reliable
* UDP (Datagram Socket): packets, best effort

→ Recieving host
* Destination address that uniquely identifies the host.

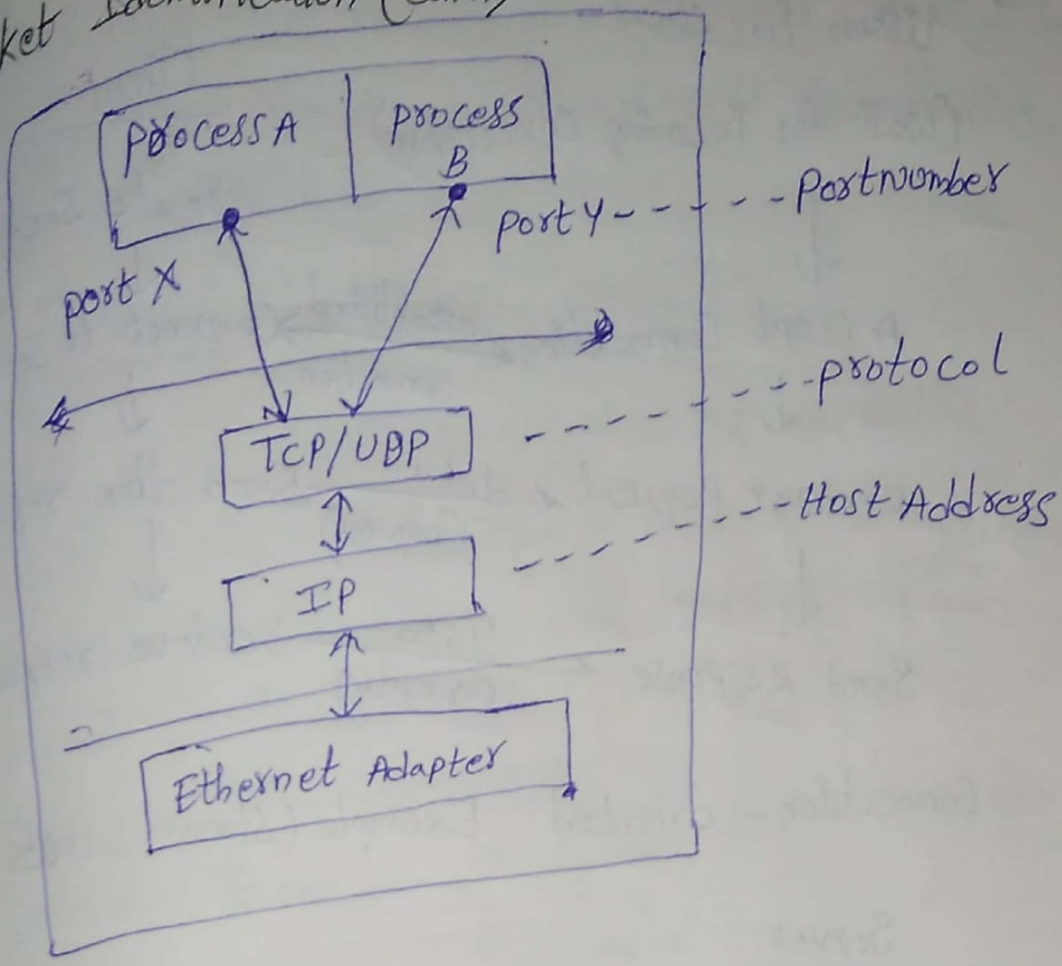
* An IP address is a 32-bit quantity

→ Recieving Socket
* Host may be running many different process.
* ~~Definition~~
* Destination port that uniquely identifies the socket.
* A port number is a 10-bit quantity.

Two types of Adress:-

They are 1) IP address
2) Socket address (Port or physical addresses)

Socket Identification (Cont.)



Clients and Servers:-

→ Client program
* Running on end host
* Requests service
Eg:* web browser

→ Server program
* Running on end client
* provided service
Eg:- web server

client-server Communication Stream Sockets (TCP): Connection-oriented.

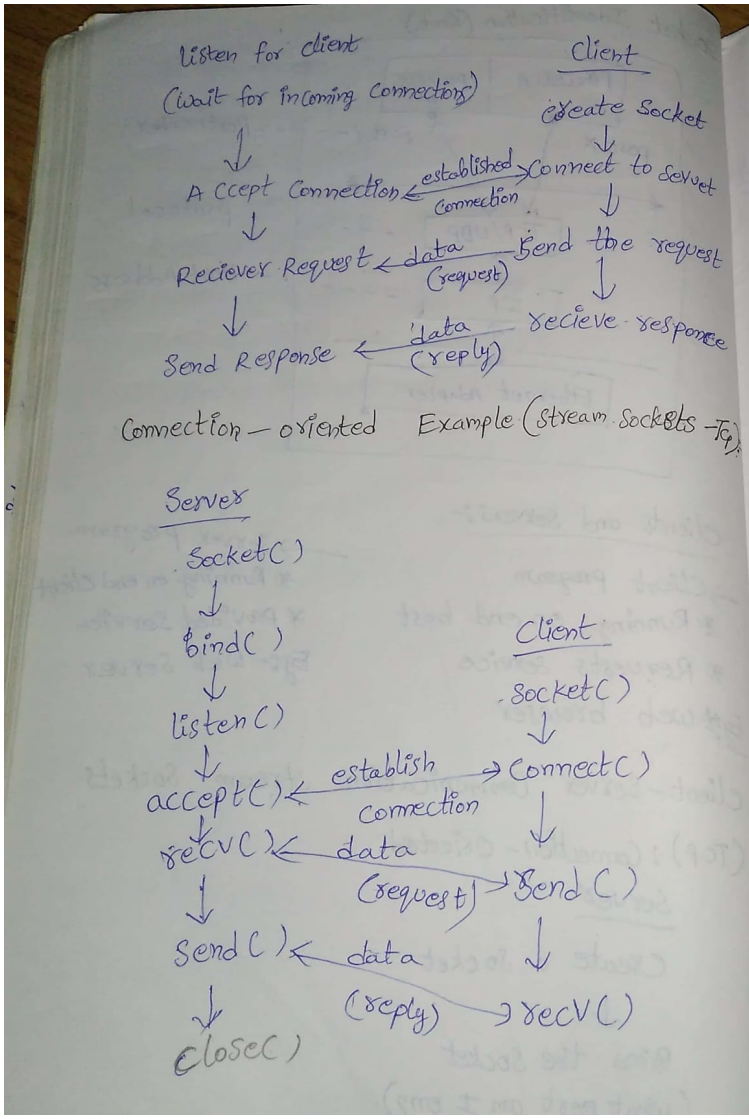
Server

Create a Socket

↓

Bind the Socket
(What port am I on?)

↓



Connection – oriented Example (stream Sockets - TCP)

client: Learning Server Address/port (cont.)

→ Data structure to host address information.

```
struct addrinfo
{
  int     ai_flags;
  int     ai_family; // eg. AF_INET for IPV4
  int     ai_socktype; // eg. sock_STREAM for TCP
  int     ai_protocol; // eg. IPPROTO_TCP
  size_t  ai_addrlen;
  char    *ai_canonname;
  struct  sockaddr  *ai_addr; // Point to Sockaddr Struct
  struct  addrinfo  *ai_next;
}
```

→ Example

```
hints.ai_family = AF_UNSPEC; // don't care IPV4 or IPV6
hints.ai_socktype = sock_STREAM; // TCP stream sockets.
int status = getaddrinfo("www.cnn.com", "80", &hints,
                         &result);
// result now points to a linked list of 1 or more
   addrinfos.
// etc.
```

System Calls:-

1) Create a Socket:-

→ int Socket(int domain, int type, int protocol);
(type: which type of protocol tcp or UDP)

Returns a file descriptor (or handle) for the Socket.

→ Domain:- protocol family
  → PF_INET for IPV4
  → PF_INET6 for IPV6

→ Type: semantics of the communication
  * SOCK_STREAM: Reliable byte stream (TCP)
  * Sock_DGRAM: message-oriented service (UDP)

→ protocol: specific protocol
  * UNSPEC: unspecified
  * (PF_INET and SOCK_STREAM already implies TCP)

→ Example

```
Sockfd = Socket(result→ai_family,
                result→ai_socktype,
                result→ai_protocol);
```

2) Connect:-

→ int Connect(int sockfd, struct sockaddr* my server_address, socketlen_t addrlen);

→ Args: Socket descriptor, server address, and address size.

→ Returns 0 on success, and -1 if an error occurs.

3) send():-

int send(int sockfd, void *msg, size_t len, int flags)

→ Args: socket descriptor, pointer to buffer of data to send, and length

→ Returns

4) Recieve:-

int recv(int sockfd, void *buf, size_t len, int flag);

5) bind:-

int bind(int sockfd, struct sockaddr *my_addr, socklen_t addrlen);

6) accept:-

int accept(int sockfd, struct sockaddr *addr, socketlen_t *addrlen);

7) listen:-

int listen(int sockfd, int backlog);

8)

mid-I

1) Explain about unix directory structure?
2) write program to print list of prime no.s in b/w arange x & y.
3) write a shell program to check whether the given no. is palindrome or not.
4) Explain about Stat & fStat function with example program.
5) Explain kinds of files and explain file structure
6) Explain shell variables & shell meta characters
7) Explain about directory system calls.
8) Explain about the following system calls.
   1) create
   2) Read
   3) open
   4) write
   5) lseek
   6) close & example
9) Explain about process and process structure