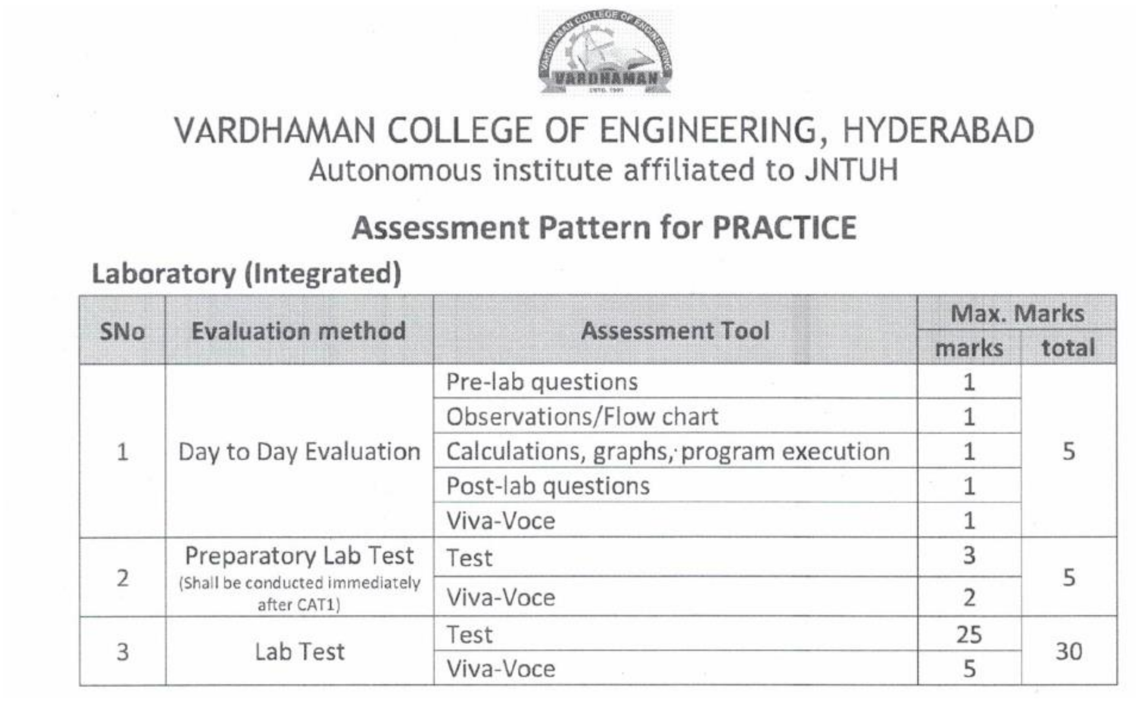
LIST OF PROGRAMS FOR PRACTICE:

No	Title of the Experiment	Tools and Techniques	Expected Skills/Ability
1	Practice commands mkdir, rmdir, cat,nl, ls, cp, mv, rm, man.	Open source Ubuntu/Linux Operating System. C/C++ Compiler	Understand File handling Utilities
2	Practice commands wc, uniq, comm, cmp, diff, ln, unlink,chmod,du,df.		
3	Practice head, tail, sort, grep, egrep, fgrep, cut, paste, join.		
4	Process Management System calls fork (), exec () and wait () system calls.		Apply Process System Calls
5	a) Two-way Communication using Pipes. b) Process Communication using FIFOs.		Implement IPC mechanisms using UNIX/LINUX System calls
6	Implement Shared Memory form of IPC.		
7	Implement Message Queue form of IPC.		
8	Implement Semaphore operations for process synchronization (Producer-Consumer problem)		
9	Simulate cp and ls commands.		Simulate UNIX/LINUX System Calls
10	Simulate head and tail utilities using file handling system calls.		
11	Shell Script programs using control statements		Write Shell Script Programs
12	Shell Script programs using control statements		
13	Program to implement FIFO Page replacement algorithm		Implement Page Replacement algorithms.
14	Program to implement LRU Page replacement algorithm		

ASSESSMENT SCHEME



VARDHAMAN COLLEGE OF ENGINEERING, HYDERABAD
Autonomous institute affiliated to JNTUH

Assessment Pattern for PRACTICE

Laboratory (Integrated)

SNo	Evaluation method	Assessment Tool	Max. Marks	
			marks	total
1	Day to Day Evaluation	Pre-lab questions	1	5
		Observations/Flow chart	1	
		Calculations, graphs, program execution	1	
		Post-lab questions	1	
		Viva-Voce	1	
2	Preparatory Lab Test (Shall be conducted immediately after CAT1)	Test	3	5
		Viva-Voce	2	
3	Lab Test	Test	25	30
		Viva-Voce	5	

Introduction to UNIX

What is UNIX?

- UNIX is a family of multitasking, multiuser computer operating systems that derive from the original AT&T Unix.
- Development started in 1969 at the Bell Labs research center by Ken Thompson, Dennis Ritchie, and others.
    - It is a stable, multi-user, multi-tasking system for servers, desktops and laptops.
    - Proprietary UNIX operating systems (and Unix-like variants) run on a wide variety of digital architectures, and are commonly used on Web servers, mainframes, and supercomputers.
    - In recent years, smart phones, tablets, and personal computers running versions or variants of UNIX have become increasingly popular.
    - UNIX is a Console User Interface (CUI) Operating System.
    - UNIX is Completely written in C Language.

Different Flavors / Variants of UNIX / Types of UNIX

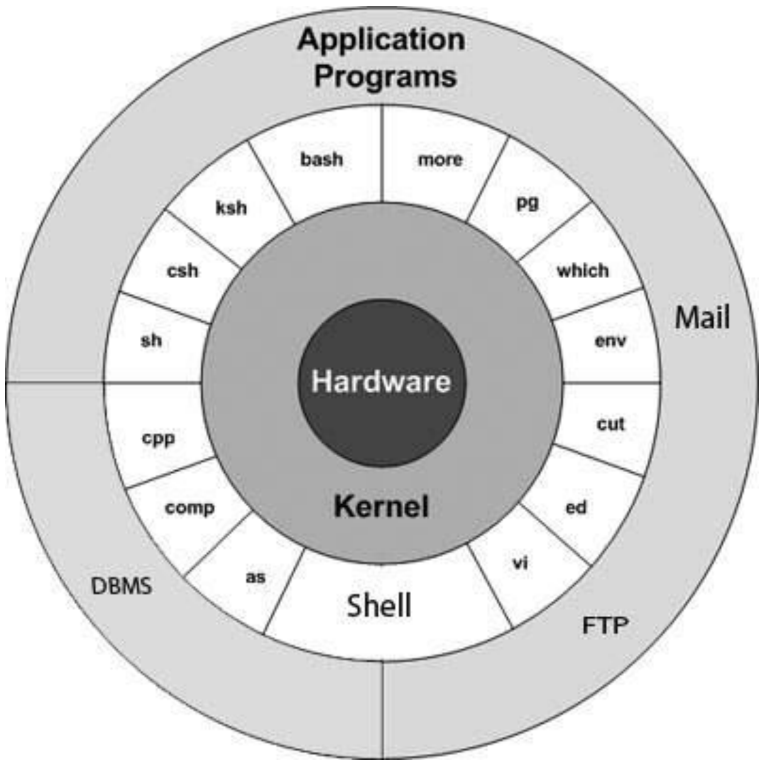
There are many different versions of UNIX, although they share common similarities.

1. BSD (Berkeley Software Distribution) UNIX
2. SUN SOLARIS (Sun Microsystems) [Servers & Work Stations]
3. NOVELL NETWARE (Novell)
4. IBM AIX (International Business Machines)
5. HP – UX (Hewllet Packard)
6. TRU 64 (Hewllet Packard)
7. MacOS X
8. GNU/Linux
9. Fedora Linux [servers and desktop PCs]

The UNIX Architecture

UNIX Operating system Uses Layered Architecture.

The UNIX operating environment is organized into three layers.



1. **The innermost level of UNIX is the kernel. This is the actual operating system, a single large program that always resides in memory. Sections of the code in this program are executed on behalf of users to do needed tasks. Like access files or terminals. Strictly speaking, the kernel *is* UNIX.**
   - **Kernel − the kernel is the heart of the operating system.**
   - **Kernel -** interacts with the hardware directly.
   - **User Communicate with kernel with System calls**
   - **Kernel Performs the functions Like**
     - **File Management**

- Controls Computer Hardware Resources memory, CPU and I/O.
- Memory Management
- Scheduling CPU Time between Processes
- Managing Data Transfer between Files and H/W
- Handling Interrupts
- Controls the Hardware resources
- Controls the access to computer by Several Users

2. **The next level of the UNIX environment is composed of Programs, Commands or Utilities.**

   Commands and utilities of UNIX are simply a set of programs that have become standardized and distributed.

   This level also includes standardized system routines, or application programming interfaces (APIs), that programmers can use to create new programs and utilities.

3. **The final level of the Unix environment, which stands like an umbrella over the others, is the shell.**

   The shell processes your terminal input and starts up the programs that you request. i.e. The Shell is the utility that processes user requests.

   - **The shell acts as an interface between the user and the kernel.**
   - **When a user logs in, the login program checks the username and password, and then starts another program called the shell.**
   - **The shell is a command line interpreter (CLI). It interprets the commands the user types in and arranges for them to be carried out.**
   - **The commands or utilities are themselves programs**

Files and Processes

Everything in UNIX is either a File or a Process.

UNIX Processes

- **A Process is an instance of a program, with its needed environment of file connections, etc., running in the CPU.**
- **In UNIX, every command or program you run becomes a separate process.**
- **Processes can also be combined in interesting ways to do complicated tasks.**
- **A process is an executing program identified by a unique PID (process identifier).**

A UNIX File

is a collection of data. They are created by users using text editors, running compilers etc.

Examples of files:

- **a document**
- **the text of a program written in some high-level programming language**
- **instructions comprehensible directly to the machine (an executable or binary file)**
- **A directory, containing information about its contents, which may be a mixture of other directories (subdirectories) and ordinary files.**

Advantages of UNIX

- Full multitasking with protected memory. Multiple users can run multiple programs each at the same time without interfering with each other or crashing the system.i.e. Multi User and Multi Tasking System.
- Most of UNIX flavors are Open Source. (Freeware)
- UNIX is Extensible and Scalable Operating System.
- Very efficient virtual memory, so many programs can run with a modest amount of physical memory.
- Access controls and security.
- A rich set of small commands and utilities that do specific tasks
- Ability to string (combine) commands and utilities together in unlimited ways to accomplish more complicated tasks .
- A powerfully unified file system. Everything is a file: data, programs, all physical devices and Directories.
- A lean kernel that directly communicate with Hardware.
- Available on a wide variety of machines - the most truly portable operating system.

Disadvantages of UNIX

- The traditional command line shell interface is user hostile -- designed for the programmer, not for the casual user.
- Commands often have cryptic names and give very little response to tell the user what they are doing. Much use of special keyboard characters.
- To use UNIX well, you need to understand some of the main design features. Its power comes from knowing how to make commands and programs interact with each other.
- Richness of utilities (over 400 standard ones) often overwhelms novices. Documentation is short on examples and tutorials to help you figure out how to *use* the many tools provided to accomplish various kinds of tasks.

UNIX COMMANDS

A Utility or Command is a standard UNIX/LINUX program (process) that provides a support process for users.

The various utilities can be

1. **File Handling Utilities**
2. **Process Utilities**
3. **Text Processing Utilities**
4. **Backup Utilities**
5. **Filters**
6. **Disk utilities**
7. **Networking Commands …**

Command Syntax in UNIX

$verb [options] [arguments]

Eg: wc –lc file1 file2

Verb: Command Name, Tells what action to be done.

Eg: wc

Options: Modifies how the action is applied, options are specified using +/- and many options can be specified at a time.

Eg: -lc

Arguments: Provides additional information to the command. (File names, device name etc).

Eg: file1, file2

Basic Commands in UNIX

S.No	Command Name and Syntax	Options and Example
1	**1. date command** **Display the system date and time.** **$ date -s "Sun Dec 18 21:00:00 PDT 2016"** **$ date -u** **$ date +%a** **$ date +%B** **$ date +%d** **$ date +%D** **$ date +%F** **$ date +%T**	**-s : To set the date** **-u : To display Universal Time** **+%a or +%A : To display Weekday name** **+%b or +%B :To display Month name** **+%d: To display current day of month** **+%D: To display current date in mm/dd/yy** **+%F:To display date in YYYY-MM-DD** **+%T:To display time in 24 hours format hh:mm:ss**

2	**2. cal command** ✓ Displays the calendar for a specific month or for a year ✓ It uses arguments. ✓ If arguments are not specified it displays the current month calendar. ✓ If single argument with 4 digits will display whole year calendar (1-9999) ✓ If two arguments are specified, denotes month and year.	**–y: displays complete year calendar.** $cal $cal 12 2017 $cal –y 2017 $cal 2017

3	**echo command**
	✓ Displays a line of text. i.e. It copies its arguments back to the terminal.
	$ echo -e 'Hello \bworld \bhow \bare \byou .'
	$ echo -e 'Hello \nworld \nhow \nare \nyou.'
	$ echo -e 'Hello \tworld \thow \tare \tyou '
	$ echo -e 'Hello \vworld \vhow \vare \vyou '
		-n: Do not output a trailing newline.
		-e: Enable interpretation of backslash escape sequences
		\\ A literal backslash character ("\").
		\a An alert (The BELL character).
		\b Backspace
		\n A newline.
		\t A horizontal tab.
		\v A vertical tab.
		$echo Hello World
		$echo "Hello World"

4	**clear command** ✓ clear the terminal screen, puts the cursor at the top of the screen	$clear
5	**script command** ✓ Used to record an interactive session. ✓ If the argument file is given, script saves all dialogue in file. If no file name is given, the typescript is saved in the file typescript.	-a: Append the output to file or typescript, retaining the prior contents. $script MyFile $date $cal $echo 'hello' $exit $script MyFile -To Use the previous script $script –a MyFile $cal $echo 'hello' $exit
6	**uname command** ✓ displays about the user	-a: print all information -s: print the kernel name -n: print the network node hostname

	(system information).	-r: print the kernel release -I: print the hardware platform $uname $uname –a
7	passwd command ✓ Used to change the user Password ✓ The passwd command changes passwords for user accounts. A normal user may only change the password for his/her own account, while the superuser may change the password for any account.	$ passwd (current) UNIX password: Enter new UNIX password: Retype new UNIX password: passwd: password updated successfully

Week-1	Practice commands – pwd, mkdir, rmdir, cat, nl, ls, cp, mv, rm, man.

S.No	Command Name and Syntax	Options and Example
1	**pwd (Print Working Directory)** ✓ Print the name of current working directory. ✓ Displays the absolute path name of working directory. $pwd	It has no options and arguments $pwd

2	**mkdir (Make Directory)**
	✓ mkdir command in Linux allows the user to create directories (folders in some operating systems).
	✓ This command can create multiple directories at once.
	✓ set the permissions for the directories.
	✓ If the permissions are not specified, the directory typically will have read and execute permissions to all 3 types of users and write only for owners.
		–m: to specify the directory permissions.
		–p: to create sub directories.
		$ mkdir mydir
		$ mkdir -m a=rwx mydir
		//Create the mydir directory, and set its permissions such that all users may read, write, and execute the contents.
		$ mkdir Mydir1 Mydir2 Mydir3
		
		$ mkdir –p /home/ Mydir1 Mydir2
		
		$ mkdir -p first/second/third

3	**rmdir (Remove Directory)**
	✓ rmdir command is used remove empty directories from the filesystem in Linux.
	✓ So if the specified directory has some directories or files in it then this cannot be removed by rmdir command.
	✓ This removes only an empty directory. i.e directory without any sub-directories or files:
		$ rmdir dir1
		
		$ rmdir mydir1 mydir2 mydir3
		
		$ rmdir -p mydir/mydir1

4	cat (Concatenate)
	✓ Used to create simple files.
	✓ To display the text files
	✓ To copy the contents of one file to another
	✓ To append the content to an existing file
	
	The operators used are
	
	>: Redirect the output of the cat command to a file rather than standard output. (Monitor)
	>>: Append the output of cat command to the end of an existing file.
		-b: starting at 1 , display the line number for non blank output lines
		-e: display control and non printing chars followed by a $symbol at the end of each line
		-n: starting at 1, number all output lines.
		
		$cat > myFile1
		……… // creating file
		cntrl + d
		$cat myfile1 myfile2
		//display the contents of files
		$cat myfile1 > myfile3
		//copy contents from one file to another
		$cat >> myfile1
		…….. //Appending to a file
		cntrl+d
		$cat –b myfile1
		$cat –n myfile1

5	nl (Number Lines)
	✓ used to prefix line number for each line in one or more files.
	✓ Numbers all non-empty lines.
	✓ If no file name specified, the command read from keyboard.
		-s: to specify the delimiter (by default the delimiter is white space)
		-w: to specify the width of number format
		-i: line number increment at each line
		-a: number all the lines
		-b string: number only lines that contains specified string.
		
		$ nl myfile.txt
		$ nl –a myfile.txt
		$nl –b “cse” myfile.txt
		$ nl -i 3 geekfile.txt
		or
		$ nl -i3 geekfile.txt
		$ nl –w3 myfile.txt
		$ nl –w3 -s: myfile.txt

6	ls (lists) is a Linux shell command that lists directory contents of files and directories. The contents of long list includes 1. File Type - : regular file d : directory file b : block file l : link file 2. File Permission 3. Number of links 4. User Name 5. Group Name 6. File Size 7. Last Modified data & Time 8. File Name	-l: display long list of working directory or specified directory name. -a: display all the files including hidden files also -1: display all in one column -r: display in descending order of files -R: display recursive list. i.e. Sub directories also. -i: displays i-Node number of a file. -t: It sorts the file by modification time, showing the last edited file first. $ls $ls –l mydir $ls –l $ls –a $ls -1 $ls –r $ls –i myfile $ls –R myfile

7	cp (Copy)
	✓ This command is used to copy files or group of files or directory.
	✓ It creates an exact image of a file on a disk with different file name.
	✓ cp command require at least two filenames in its arguments.
	✓ If destination file already exists, its contents are replaced with source file.
	✓ It copies both text and binary files.
	✓ After copy , the permissions of the file are preserved.
		-v : (verobose) – to see the files are copied.
		-p : (preserve) – If destination file exists, the source file attributes are preserved.
		-i : (Interactive)- If destination file exists, its contents are removed after taking user confirmation.
		-R or –r (Recursive)- Copies all directories, sub directories and their files.
		-b(backup): With this option cp command creates the backup of the destination file in the same folder with the different name and in different format.
		`cp [OPTION] Source Destination`
		`cp [OPTION] Source Directory`
		`cp [OPTION] Source-1 Source-2 Source-3 Source-n Directory`
		cp -R Src_directory Dest_directory
		$ cp -b a.txt b.txt
		$ cp *.txt Folder1 //wild card copy

8	mv (move)
	mv is used to move one or more files or directories from one place to another.
	After move, old file will be lost and new file is found at destination.
	It has two distinct functions:
	(i) It rename a file or folder.
	(ii) It moves group of files to different directory.
	No additional space is consumed on a disk during renaming.
		-i : Interactive
		-f : forced deletion
		-u: move only files that are not in destination.
		-b(backup):
		
		mv [Option] source destination
		`$ mv a.txt geek.txt`
		`$ mv -i geek.txt b.txt`
		$mv dir1 dir2
		$mv –u dir1 dir2
		$mv my* dir1

9	rm (remove)
	rm command is used to remove objects such as files, symbolic links and so on.
	It is used to remove a file entry from the directory.
	It cannot be used to remove directory.
		-f : forced removal
		-i: interactive, ask the user for confirmation before remove.
		
		`rm [OPTION]... FILE...`
		`$ rm b.txt c.txt`
		`$ rm -i d.txt`
		`$ rm –f d.txt`

10	man (manual) Used to display the user manual of any command that we can run on the terminal. It is used to obtain offline documentation of any command. It provides a detailed view of the command which includes NAME, SYNOPSIS, DESCRIPTION, OPTIONS, EXIT STATUS, RETURN VALUES, ERRORS, FILES, VERSIONS, EXAMPLES, AUTHORS and SEE ALSO. If no options, It displays the whole manual of the command.	-k : Displays the related commands. $man [OPTION]... [COMMAND NAME]... $ man printf $man cat $man –k sort

Week-2	Practice commands wc, uniq, comm, cmp, diff, ln, unlink,chmod,du,df.

S.No	Command Name and Syntax	Options and Example
1	**wc command** (word count) • **It is used to find out number of lines, word count, byte and characters count in the files specified in the file arguments.** • **The character count includes new lines also** • **By default it displays four-columnar output.** • **First column shows number of lines present in a file specified, second column shows number of words present in the file, third column shows number of characters present in file and fourth column itself is the file name which are given as argument.** • **When more than file name is specified in argument then command will display four-columnar output for all individual files plus one extra row displaying total number of lines, words and characters of all the files specified in argument, followed by keyword** total.	`$wc [OPTION]... [FILE]...` **$wc myfile1** **$wc myfile1 myfile2** **The options are** **-l: This option prints the number of lines present in a file.** **$wc –l myfile1** **$wc –l myfile1 myfile2** **-w: This option prints the number of words present in a file.** **$wc –w myfile1** **$wc –w myfile1 myfile2** **-c: This option displays count of bytes/characters present in a file** **$wc –c myfile1** **$wc –c myfile1 myfile2** **-L: it can be used to print out the length of longest (number of characters) line in a file.** **$wc –L myfile1** **Example Application:** **To count all files and folders in a file** **$ls ramesh | wc -l**

2.	**uniq command**
	• Reports or filters out the repeated lines in a file.
	• Locates repeated and non repeated lines.
	• Deletes the duplicate lines, keeping the first and deleting the others.
	• To delete, the lines must be adjacent.
	• uniq is the tool that helps to detect the adjacent duplicate lines and also deletes the duplicate lines.
	• uniq filters out the adjacent matching lines from the input file(that is required as an argument) and writes the filtered data to the output file
		`$uniq myfile1`
		`The options are`
		-c: It tells the number of times a line was repeated.(count)
		$uniq –c myfile
		
		`-d: It only prints the repeated lines only once.` (duplicate)
		
		$uniq –d myfile1
		
		-D: It displays all occurrences of the repeated lines(duplicate)
		$uniq –D myfile1
		
		-u: It prints only the unique lines.
		$uniq –u myfile1
		
		-w: limit the comparison to a set number of characters.
		Eg: Wel Done
		Wel Come
		Wel Going
		
		$uniq –w 3 myfile1 //Wel
		
		-i: It is used to make the comparison case-insensitive.
		$uniq –i myfile1
		If lines are not adjacent, they must be sorted before the uniq command.
		Eg: $sort myfile1 | uniq

3	**du command (disk usage)**
	• Summarize the disk usage of the set of files, recursively for directories.
	• Used to estimate file space usage.
	• Used to track the files and directories which are consuming excessive amount of space on hard disk drive.
		`du /ramesh/cse`
		
		`The options are`
		-h: print sizes in human readable format
		-a: printing all files including directories.
		-c: to print grand total.
		-s: summary of file system
		du –s /vce/cse

4	**df command (disk free)**
	• Reports the file system disk space usage.
	• df displays the amount of disk space available on the file system containing each file argument.
	• If no file name is given, the space available on all the currently mounted file system is shown.
	• By default, df shows the disk space in 1 K blocks.
		`$df`
		`$df myfile1`
		
		The options are
		-a: It includes all the dummy files also in the output which are actually having zero block sizes.
		`$df -a`
		
		-h: It prints sizes in human readable format.
		
		`$df -h`
		
		-i: to display the inode information instead of block usage.

5	**chmod command (change mode)**
	• Used to change the access mode of a file.
	• Changing file permissions
	• The chmod command is used to set the permissions of one or more files.
	• A File or Directory is created with a default set of permissions.
	
	UNIX/LINUX provides three types of permissions to a file or directory.
	read – r
	write – w
	execute – x
	The three permissions are given to three types of users
	User(owner) – u
	Group - g
	Others - o
		The permissions can be changed in three ways
		1. Relative permissions
		Eg: Adding writes permissions to user and execute permissions to all.
		`$chmod u+w,ugo+x myfile1`
		`$chmod u+w,a+x myfile1 myfile2`
		`$chmod u+w, +x myfile1`
		
		Eg: Removing Write permissions to others
		`$chmod o-w myfile1 myfile2`
		Eg: Remove Execute permissions to group and add write to others.
		`$chmod g-x,o+w Myfile1`
		
		Eg:`$chmod u+rwx myfile1`
		`$chmod a+rwx myfile1`
		
		Note: Using relative permissions the other permissions are unchanged.

	[ALL : a -> ugo] `$chmod -options mode file/dir` Mode => Specified as MODE who operator permissions (u,g,o,a)+,-,= r,w,x	2. Absolute permissions Using absolute permissions, all permissions are unchanged. 3. Using Octal Codes

6	**cmp command (Compare)**
	• The two files are compared byte by byte; the location of the first mismatch is shown on the screen.
	• Helps to find out whether the two files are identical or not.
	• When cmp is used for comparison between two files, it reports the location of the first mismatch to the screen if difference is found.
	• cmp displays no message and simply returns the prompt if the the files compared are identical.
		`$cmp myfile1 myfile2`
		The options are
		-l: Displays all the differences found in files byte by byte.
		
		`$cmp –l myfile1 myfile2`
		
		-s: suppress list option, no output is displayed. This gives an exit value of 0 if the files are identical, a value of 1 if different, or a value of 2 if an error message occurs.
		
		`$cmp –s myfile1 myfile2`
		
		-b: displays the differing bytes in the output.
		`$cmp –b myfile1 myfile2`
		
		-i: skip a particular number of initial bytes from both the files.
		`$cmp –i 10 myfile1 myfile2`
		
		-n: limit the number of bytes you want to compare
		`$cmp –n 10 myfile1 myfile2`

7	**comm command (common)** Compare two sorted files line by line and write to standard output; the lines that are	`$comm myfile1 myfile2` The options are -1: suppress first column -2: suppress second column

	common and the lines that are unique.
	
	It compares the files line by line and displays the result in 3 columns.
	
	Col1 : Unique lines in file1
	Col2: Unique lines in file2
	Col3: Common lines in both the files.
		`-3:` suppress third column
		
		`$comm -1 myfile1 myfile2`
		
		`$comm -2 myfile1 myfile2`
		
		`$comm -3 myfile1 myfile2`
		
		`$comm -12 myfile1 myfile2`
		
		`$comm -23 myfile1 myfile2`

8	**diff command (difference)**
	• Used to display the differences in the files by comparing the files line by line.
	• It tells us which lines in one file have is to be changed to make the two files identical.
	• When used with same files, it produces a detailed output.
		`$diff myfile1 myfile2`
		
		`The options are`
		`-b : Ignore Trailing Blanks`
		`-w : Ignore White space`
		`-i : Ignore the case in comparison`
		
		`$diff –i myfile1 myfile2`
		
		`In Output`
		`c – Change, d – Delete, a- Append`

9	**ln command(link command)**
	• A file is linked with ln command. The command will take two arguments.
	• The ln command is used to create links between files.
	• Unix/Linux allows two types of links.
	1. Hard Links
	• The default link type is hard link.
	• Ln command executed with no options is a hard link.
	• Hard links must be made within the file system only.
		• If a file is shared by two or more users, instead of providing same copy a link is created for the file in each user.
		
		Hardlink
		ln myfile1 mylink1
		ln myfile1 mylink2
		ln myfile1 mylink3 mylink4
		
		`ln mydir link1 -> invalid`
		
		`ln /home/vce/myfile  /user/ramesh/mylink (invalid)`

	<ul><li>Hard links are created only for regular files but not to directories.</li><li>Hard links refer to specific location of physical data.</li></ul>After each link to a file, the hard link count is incremented by 1.
	Hard links are most useful for keeping file content in a single location by avoiding duplication of what might be a very large amount of data.
	
	2. Symbolic links
	<ul><li>Allows creating a link to a directory, to a different file system or in different partitions.</li><li>A symbolic link also known as a softlink or symlink.</li><li>It is a special file link to a file or directory.</li><li>ln command with –s is used to create symbolic links.</li></ul>
		`-i: used to see the inode number.`
		
		ln –i myfile1 mylink3 mylink4
		
		`Note: I-Node number of links and file will be same.`
		
		Symbolic link
		
		`ln -s myfile1 link1`
		`ln -s mydir mylink`
		
		`ln /home/vce/myfile /user/ramesh/mylink`

10	unlink command<ul><li>Used to remove a link from the file system.</li><li>After each unlink, the hard link count is decremented by 1.</li><li>It is similar to rm command</li></ul>	$unlink mylink1 $unlink mylink2 mylink3

Week-3	Practice head, tail, sort, grep, egrep, fgrep, cut, paste, join.

S.No	Command Name and Syntax	Options and Example
1	head command The head command, as the name implies, print the top N lines number of data of the given input. By default, it prints the first 10 lines of the specified files. If more than one file name is provided then data from each file is preceded by its file name. Displays the beginning of one or more files	head myfile1 //displays top 10 lines head myfile1 myfile2 //displays top 10 lines from both the files $head -4 myfile1 //display top 4 lines The options are -n num: Prints the first 'num' lines instead of first 10 lines $head -n 5 myfile1 myfile2 $head -4 myfile1 myfile2 -c num: Prints the first 'num' bytes from the file specified. Newline count as a single character, so if head prints out a newline, it will count it as a byte. $head -c 5 myfile1 myfile2 -q: It is used if more than 1 file is given. Because of this command, data from each file is not proceeds by its file name. $head -q –n 5 myfile1 myfile2 -v: By using this option, data from the specified file is always preceded by its file name. $head –v myfile1

2.	**tail command**
	The tail command, as the name implies, print the last N number of data of the given input.
	By default it prints the last 10 lines of the specified files.
	If more than one file name is provided then data from each file is precedes by its file name.
	Displays the end of the one or more files.
	
	`$ tail -n 7 state.txt | sort –r`
	
	`$ cat state.txt | head -n 20 | tail -n 5  > list.txt`
	
	`$ head -n 20 state.txt | tail -10`
	
	`$ ls -t | head -n 3 | sort`
		$tail myfile1 //displays last 10 lines
		$tail myfile1 myfile2
		//displays last 10 lines from both the files
		$tail -4 myfile1 //display last 4 lines
		
		The options are
		-n num: Prints the last ‘num’ lines instead of last 10 lines.
		$ tail -n 3 myfile1 myfile2
		$ tail -3 myfile1
		Tail command also comes with an ‘+’ option which is not present in the head command. With this option tail command prints the data starting from specified line number of the file instead of end.
		$ tail +5 myfile1
		
		-c num: Prints the last ‘num’ bytes from the file specified. Newline count as a single character, so if tail prints out a newline, it will count it as a byte.
		$ tail -c -6 myfile1
		$ tail -c 6 myfile1
		
		By +num, it display all the data after skipping num bytes from starting of the specified file and by -num, it display the last num bytes from the file specified.
		$ tail -c +5 myfile1
		-q: It is used if more than 1 file is given. Because of this command, data from each file is not precedes by its file name.
		$ tail –q state.txt capital.txt
		
		-v: By using this option, data from the specified file is always preceded by its file name.
		$ tail –v state.txt capital.txt

3	**sort command**
	SORT command is used to sort a file, arranging the records in a particular order. By default, the sort command sorts file assuming the contents are ASCII. Using options in sort command, it can also be used to sort numerically.
	• SORT command sorts the contents of a text file, line by line.
	• The sort command can also sort by items not at the beginning of the line
	• Sorting is done based on one or more sort keys extracted from each line of input.
	• By default, the entire input is taken as sort key. Blank space is the default field separator
	
	The sort command follows these features as stated below:
	1. Lines starting with a number will appear before lines starting with a letter.
	2. Lines starting with a letter that appears earlier in the alphabet will appear before lines starting with a letter that appears later in the alphabet.
	3. Lines starting with a lowercase letter will appear before lines starting with the same letter in uppercase.
		`$ sort filename.txt`
		The options are
		
		-r Option: Sorting In Reverse Order: You can perform a reverse-order sort using the -r flag. the -r flag is an option of the sort command which sorts the input file in reverse order i.e. descending order by default.
		
		`$ sort -r inputfile.txt`
		-n Option: To sort a file numerically
		// the file with numeric data
		
		`$ sort -n filename.txt`
		`$ sort -nr filename.txt`
		
		-k Option: Unix provides the feature of sorting a table on the basis of any column number by using -k option.
		For example, use "-k 2" to sort on the second column.
		`$sort –k 2 emp.txt`
		//sort by second field.
		
		-c option: This option is used to check if the file given is already sorted or not & checks if a file is already sorted pass the -c option to sort.
		`$ sort -c filename.txt`
		-u option: To sort and remove duplicates pass the -u option to sort.
		`$ sort -u filename.txt`
		
		-M Option: To sort by month pass the -M option to sort. This will write a sorted list to standard output ordered by month name.
		`$ sort –M months.txt`

4

cut command

The cut command in UNIX is a command for cutting out the sections from each line of files and writing the result to standard output.

It can be used to cut parts of a line by byte position, character and field**.**

Basically the cut command slices a line and extracts the text.

```
$ cat state.txt | cut -d '  ' -f 1 | sort –r
```

```
$ cat state.txt | head -n 3 | cut –d ' ' -f 1 > list.txt
```

The Options are

-b(byte): **To extract the specific bytes, you need to follow -b option with the list of byte numbers separated by comma. Range of bytes can also be specified using the hyphen(-). Tabs and backspaces are treated like as a character of 1 byte.**

```
$ cut -b 1,2,3 myfile
$ cut -b 1-3,5-7 myfile
$ cut -b 1- myfile
$ cut -b -3 myfile
```

-c (column): **To cut by character use the -c option. This selects the characters given to the -c option.**

```
$ cut -c 2,5,7 myfile
$ cut -c 1-7 myfile
$ cut -c 1- myfile
$ cut -c -5 myfile
```

-f (field): **To extract the useful information you need to cut by fields rather than columns. List of the fields number specified must be separated by comma.**

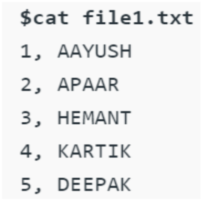
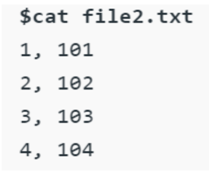
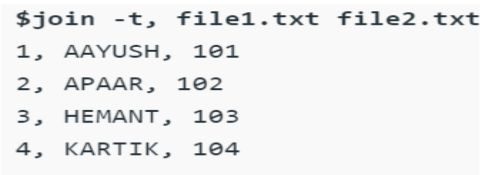
```
$ cut -f 2 myfile
$ cut -f 2 myfile1 myfile2
```

-d **option is used then it considered space as a field separator or delimiter:**

```
$ cut -d " " -f 2 myfile
$ cut -d ":" -f 2 myfile
$ cut -d " " -f 2-4 myfile
```

5	**paste command**
	It is used to join files horizontally (parallel merging) by outputting lines consisting of lines from each file specified, separated by tab as delimiter, to the standard output.
		$ paste file1 file2 file3
		
		The Options are
		
		-d (delimiter): Paste command uses the tab delimiter by default for merging the files. The delimiter can be changed to any other character by using the -d option.
		$ paste -d "|" file1 file2
		
		-s (serial): We can merge the files in sequentially manner using the -s option. It reads all the lines from a single file and merges all these lines into a single line with each line separated by tab. And these single lines are separated by newline.
		$ paste -s file1 file2
		
		$ paste –s –d ":" file1 file2

6	join command • The join command in UNIX is a command line utility for joining lines of two files on a common field. • Suppose you have two files and there is a need to combine these two files in a way that the output makes even more sense. • For example, there could be a file containing names and the other containing ID's and the requirement is to combine both files in such a way that the names and corresponding ID's appear in the same line.	$join file1 file2 $join file1 file2 > file3 The options are using -i option : join command is by default case sensitive. i.e The common field is must be of same case. If not –i is used. $join –i file1 file2 > file3 -t option: Most of the times, files contain some delimiter to separate the columns.

	• join command is used to join the two files based on a key field present in both the files.	 `$cat file1.txt`
		`1, AAYUSH`
		`2, APAAR`
		`3, HEMANT`
		`4, KARTIK`
		`5, DEEPAK`
		 `$cat file2.txt`
		`1, 101`
		`2, 102`
		`3, 103`
		`4, 104`
		 `$join -t, file1.txt file2.txt`
		`1, AAYUSH, 101`
		`2, APAAR, 102`
		`3, HEMANT, 103`
		`4, KARTIK, 104`

7	**grep command** **“Global Regular Expression Print”.** **The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression** **Pattern can be in single or double quotes.** **Regular Expressions** **----------------------------** **grep “li*” myfile*** ***: zero or any number of occurrences** **+: one or any number of occurrences** **? : zero or one occurrence** **^: beginning of line** **$: end of line** **. : any character match** **[pqr]: any one char** **[4-8]: range of chars** **(Wildcard characters)**	**grep [options] pattern [files]** **$grep “cse” myfile1** **The options are** **-c : This prints only a count of the lines that match a pattern** **-h : Display the matched lines, but do not display the filenames.** **-i : Ignores, case for matching** **-l : Displays list of a filenames only.** **-n : Display the matched lines and their line numbers.** **-v : This prints out all the lines that do not matches the pattern** **-o : Display only the matched pattern** **-f : to keep the patterns in a file** **-A n : Prints searched line and nlines after the result.** **-B n : Prints searched line and n line before the result.** **-C n : Prints searched line and n lines after before the result.** **$grep -i "vce" file1 file2** **$grep -c "vce" file1** **$grep -l "vce" f1.txt f2.txt f3.xt f4.txt** **$ grep -o "vce" myfile1** **$ grep -n "vce" myfile1** **$ grep -v "vce" myfile1** **$ grep "^vce" myfile1** **$ grep "vce$" myfile1.txt** **$grep –f patrn.txt myfile1**

8	**egrep command (extended)**
	egrep is a pattern searching command which belongs to the family of grep functions.
	It treats the pattern as an extended regular expression and prints out the lines that match the pattern.
	If there are several files with the matching pattern, it also displays the file names for each line.
	it is faster than the grep command.
		`egrep ‘patern1 | pattern2 | pattern3’ myfile1`
		
		`//to search for multiple patterns`
		
		`egrep –i ‘patern1 | pattern2 | pattern3’ myfile1`
		
		`//ignores the case while searching the pattern`
		
		`Note: All the options and regular expressions of “grep” can also be applied.`

9	**fgrep command (fast)** The fgrep filter is used to search for the fixed-character strings in a file. There can be multiple files also to be searched. It cannot perform search based on regular expressions. Displays all lines that contain that pattern. It is the fastest approach as no substitution of regular expressions.	$fgrep "linux" myfile1 `Note: All the options and regular expressions of “grep” can also be applied.`

Week-4	Process Management System calls fork (), exec () and wait () system calls.
1.	See program below

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

	pid= fork(); //fork a child process (create)

	if(pid < 0)
	{
		printf("\n Fork failed to create a child \n");
		return 1;
	}
	else if(pid > 0) // Parent process
	{
		printf("\n The parent process id is %d \n" , getpid());
		printf("\n Parent Process Running \n");
		wait(NULL); //Parent waits for child to complete
		printf("\n Child Completed \n");
	}
	else
	{
		printf(" \n Child Process running \n");
		printf("\n The child process id is %d",getpid());
		printf("\n The parent of child is %d" , getppid());
		execlp("bin/ls" ,"ls",NULL);
	 }

	return 0;
}
```

Week-5	a) Two-way Communication using Pipes. b) Process Communication using FIFOs.

1.

a) Two-way Communication using Pipes.

```c
#include<stdio.h>
#include<string.h>
#include<sys/types.h>
#include<unistd.h>
void main()
{
	int n,p1fd[2],p2fd[2];
	pid_t pid;
	char msg[100];
	if(pipe(p1fd)==-1)  {
		printf("pipe failed\n");
		return;
	}
	if(pipe(p2fd)==-1)  {
		printf("pipe failed\n");
		return;
	}

	pid=fork();

	if(pid<0)	{
		printf("fork failed\n");
		return;
	}
	if(pid>0)//parent
	{
		close(p2fd[1]);
		close(p1fd[0]);
		printf("enter the msg to  child (Client)--->Parent\n");
		scanf("%s",msg);
		write(p1fd[1],msg,strlen(msg)+1);
		sleep(5);//sleep of 5 seconds
		read(p2fd[0],msg,100);
	printf("the msg given by child (client) is %s --->Parent\n",msg);
	}
```

	```
	else
	        {
	                close(p2fd[0]);
	                close(p1fd[1]);
	                sleep(3);
	           read(p1fd[0],msg,100);
	        printf("the msg given  by parent (Server) is %s --->child\n",msg);
	           sleep(3);//sleep of 3 seconds
	                printf("enter the msg to parent (Server) --->child\n");
	                scanf("%s",msg);
	                write(p2fd[1],msg,strlen(msg)+1);

	        }
	}
	```

2	**b) Process Communication using FIFOs.**
	//FIFO create Process
	```
	#include<stdio.h>
	#include<sys/types.h>
	#include<fcntl.h>
	#include<unistd.h>
	int main()
	{
	        int fd,x;
	        char msg[100];
	        x=mkfifo("abc",0666);
	        if(x==-1)
	        {
	                printf("fifo not created\n");
	                return 0;
	        }
	        fd=open("abc",O_WRONLY);
	        if(fd==-1)
	        {
	                printf("fifo not opened\n");
	                return 0;
	        }
	        printf("Enter the Message to FIFO ");
	        scanf("%s",msg);
	```

```
          write(fd,msg,sizeof(msg));
          close(fd);
          sleep(4);
       printf("Message written to FIFO \n");
          unlink("abc");
          return 0;
}
```

//FIFO Access Process

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<fcntl.h>
#define size 1024
int main()
{
          int fd;
          char buf[size];
          fd=open("abc",O_RDONLY);
          if(fd==-1){
                    printf("fifo not opened\n");
                    return 0;
          }
          read(fd,buf,size);
          printf("\nthe message received from FIFO is --->%s",buf);
          close(fd);
          return 0;
}
```

Week-6	Implement Shared Memory form of IPC.
1.	//Shared memory - create Process – shmcreate.c (code below)

```
//Shared memory - create Process – shmcreate.c
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include<stdlib.h>
#define SHMSZ     27

void main()
{
    char c;
    int shmid;
    key_t key;
    char *shm, *s;

     key = 5678;

     if ((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0)
      {
        printf("Shared Memory cannot be created");
        exit(1);
      }

    if((shm = shmat(shmid, NULL, 0)) == (char *) -1)
      {
        printf("shared memory cannot be attached);
        exit(1);
      }

   // Write alphabhets into shared memory - server
   for (c = 'a'; c <= 'z'; c++)
        *shm++ = c;
    *shm = NULL;

}
```

```
//Shared memory - Access Process – shmaccess.c

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#define SHMSZ     27

void main()
{
   int shmid;
   key_t key;
   char *shm, *s;

   key = 5678;


   if ((shmid = shmget(key, SHMSZ, 0666)) < 0)
    {
      printf("shared memory cannot be opened");
      exit(1);
    }


   if ((shm = shmat(shmid, NULL, 0)) == (char *) -1)
    {
      perror("Can not be attached to shared memory");
      exit(1);
    }

    //Read Alphabets from shared memory
    for (s = shm; *s != NULL; s++)
      putchar(*s);

   putchar('\n');

}
```

Week-7	Implement Message Queue form of IPC.
1.	//Message Queue - create and send Process – msgsend.c

```
//Message Queue  - create and send Process – msgsend.c
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define SIZE 128
typedef struct msgbuf {
            long    mtype;
      char mtext[SIZE];
      } message_buf;

      int main()    {
       int msqid;
       int msgflg = IPC_CREAT|0666;
       key_t key;
       message_buf sbuf;
       size_t buf_length;
        char msg[100];
       key = 1234;

        if ((msqid = msgget(key, msgflg )) < 0) {
           printf("msgget-msg queue can not be created \n");
           return 1;
        }
      printf("\n Message Queue created with ID = %d\n", msqid);
      sbuf.mtype = 1;
      printf("\n  Enter the message to send \n");
      scanf("%s",msg);
      strcpy(sbuf.mtext, msg);
      buf_length = strlen(sbuf.mtext) + 1 ;
      if (msgsnd(msqid, &sbuf, buf_length, IPC_NOWAIT) < 0) {
            return 1;
         }
     else {
        printf("\n Message Sent to Message Queue is %s \n", sbuf.mtext);
         }
      return 1;
 }
```

```
//Message Queue  - Recieve Process – msgrcv.c

#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<stdio.h>
#include<stdlib.h>
#define SIZE 128

typedef struct msgbuf
       {
          long    mtype;
          char    mtext[SIZE];
       } message_buf;

  int main()  {
    int msqid;
    key_t key;
    message_buf rbuf;
    key = 1234;
    if ((msqid = msgget(key, 0666)) < 0)
     {
       printf("msgget- can not open Message Queue");
       exit(1);
     }

    if (msgrcv(msqid, &rbuf, SIZE, 1, 0) < 0)
     {
       printf("msgrcv- error in recieving");
       exit(1);
     }
 printf("\n Message Recievied From the Queue %s \n", rbuf.mtext);

  exit(0);
 }
```

Week-10

Simulate head and tail utilities using file handling system calls. [Source JSLinux with vi editor]

1.

//Head command simulation

Create a directory

Create a text file with content. [eg: demo.txt]

Create a C programme using vi editor.

vi head.c

- ➔ Type “i” in editor to insert data.
- ➔ Type the code
- ➔ Press esc : wq to save and exit. [Esc button + : + wq]
- ➔ cc head.c -> to compile
- ➔ ./a.out. 3 demo.txt => to execute. (command line args)

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc , char *argv[])
  {
    FILE *fp;
   char *line;
   int len=0;
   int count=0;
    if(argc < 3)      {
        printf("Insuffiecent Arguments");
        return -1;
      }
    fp = fopen(argv[2] , "r");
    if(fp == NULL)      {
         printf("File cannot be opened");
         return 1;
     }
    while(getline(&line,&len,fp)!= -1)       {
          count++;
          if(count > atoi(argv[1]))
            break;
             printf("%s", line);
          fflush(stdout);
        }
   fclose(fp);
   return (0);
}
```

2.

//Tail command simulation

vi tail.c

- ➔ Type “i” in editor to insert data.
- ➔ Type the code
- ➔ Press esc : wq to save and exit. [Esc button + : + wq]
- ➔ cc tail.c -> to compile
- ➔ ./a.out. 3 demo.txt => to execute. (command line args)

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc , char *argv[])
  {
     FILE *fp;
    char *line;
     int len=0;
    int count=0,n,i;
     if(argc < 3)      {
        printf("Insuffiecent Arguments");
        return -1;
      }
     fp = fopen(argv[2] , "r");
     if(fp == NULL)      {
         printf("File cannot be opened");
         return 1;
      }
     while(getline(&line,&len,fp)!= -1)       {
          count++;
        }
     printf("\n Total number of lines is %d \n", count);
     rewind(fp);
     n =  count - atoi(argv[1]);
     i=0;
       while(getline(&line , &len , fp) !=-1)        {
             i++;
             if(i > n)
                printf("%s",line);
          }
 fclose(fp);
   return (0);
}
```

Week-13	Program to implement FIFO Page replacement algorithm
1	(code below)

```
#include<stdio.h>
void main()
 {
   int i,j,k,m,n;
   int pf=0,count=0;
   int rs[30],mf[10];
   printf("\n Enter the length of the reference string \n");
   scanf("%d",&n);
  printf("\n Enter the reference string \n");
       for(i=0;i<n;i++)
      scanf("%d",&rs[i]);
  printf("\n Enter the number of frames \n");
  scanf("%d",&m);
     for(i=0;i<m;i++)
       mf[i] =-1;
    printf("\n The FIFO page replacement process is \n");

     for(i=0; i<n ;i++)  {
         for(k=0;k<m;k++) {
             if( mf[k] == rs[i])
                break;
           }
          if(k ==m )
            {
              mf[count] = rs[i];
              count++;
              pf++;
            }
          for(j=0;j<m ;j++)
            printf("\t %d",mf[j]);
          if(k==m)
            printf("\t\tThe page fault no. is %d \n" , pf);
          printf("\n");

          if(count == m)
            count =0;
       }
 printf(" The total pagefaults is using FIFO replacement is ---> %d \n ", pf);
 }
```

Week-14	Program to implement LRU Page replacement algorithm
1	(code below)

```
#include<stdio.h>
void main()
 {
   int i,j,k,min;
   int rs[25], flag[25];
   int n,f,pf=0,next=1;
   int count[20], m[20];

   printf("Enter the length of reference string \n ");
   scanf("%d",&n);
   printf("\n Enter the reference string --->\n");
     for(i=0;i<n;i++)
      {
        scanf("%d",&rs[i]);
        flag[i] = 0;
       }
  printf(" \n Enter the number of frames \n");
  scanf("%d",&f);

   for(i=0;i<f;i++)
     {
       count[i] = 0;
       m[i] = -1;
     }

 printf("\n The Page Replacement process is --->\n" );

    for(i=0;i<n;i++)
     {
       for(j=0;j<f;j++)
        {
          if(m[j]==rs[i])
           {
              flag[i] = 1;
              count[j]= next;
              next++;
           }
        }

         if(flag[i] == 0)
           {
              if(i <f )
               {
                 m[i] = rs[i];
                 count[i]= next;
                 next++;
                }
               else
                {
                 min = 0;
                 for(j=1;j<f;j++)
                   if(count[min] > count[j])
                     min =  j;
                 m[min] = rs[i];
                 count[min] = next;
                 next++;
                }

             pf++;
           }
        for(j=0 ; j<f;j++)
          printf("%d \t",m[j]);


        if(flag[i] == 0)
           printf(" The Page Fault num is ...>%d" ,pf);
        printf("\n");

    }

 printf("\n The number of page faults using LRU is --->  %d \n" , pf);

}
```