

# **LINUX PROGRAMMING**

## **LABORATORY MANUAL**

**B.TECH**

**(IV YEAR – I SEM)**

**(2022-2023)**

### **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



### **MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY**

**(Autonomous Institution – UGC, Govt. of India)**

Recognized under 2(f) and 12 (B) of UGC ACT 1956

Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – ‘A’ Grade - ISO 9001:2008  
Certified) Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

## **DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

### **VISION**

- To acknowledge quality education and instill high patterns of discipline making the students technologically superior and ethically strong which involves the improvement in the quality of life in human race.

### **MISSION**

- To achieve and impart holistic technical education using the best of infrastructure, outstanding technical and teaching expertise to establish the students into competent and confident engineers.
- Evolving the center of excellence through creative and innovative teaching learning practices for promoting academic achievement to produce internationally accepted competitive and world class professionals.

## **PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)**

### **PEO1 – ANALYTICAL SKILLS**

To facilitate the graduates with the ability to visualize, gather information, articulate, analyze, solve complex problems, and make decisions. These are essential to address the challenges of complex and computation intensive problems increasing their productivity.

### **PEO2 – TECHNICAL SKILLS**

To facilitate the graduates with the technical skills that prepare them for immediate employment and pursue certification providing a deeper understanding of the technology in advanced areas of computer science and related fields, thus encouraging to pursue higher education and research based on their interest.

### **PEO3 – SOFT SKILLS**

To facilitate the graduates with the soft skills that include fulfilling the mission, setting goals, showing self-confidence by communicating effectively, having a positive attitude, get involved in team- work, being a leader, managing their career and their life.

### **PEO4 – PROFESSIONAL ETHICS**

To facilitate the graduates with the knowledge of professional and ethical responsibilities by paying attention to grooming, being conservative with style, following dress codes, safety codes, and adapting themselves to technological advancements.

## **PROGRAM SPECIFIC OUTCOMES (PSOs)**

After the completion of the course, B. Tech Information Technology, the graduates will have the following Program Specific Outcomes:

1. **Fundamentals and critical knowledge of the Computer System:-** Able to Understand the working principles of the computer System and its components , Apply the knowledge to build, asses, and analyze the software and hardware aspects of it .
2. **The comprehensive and Applicative knowledge of Software Development:** Comprehensive skills of Programming Languages, Software process models, methodologies, and able to plan, develop, test, analyze, and manage the software and hardware intensive systems in heterogeneous platforms individually or working in teams.
3. **Applications of Computing Domain & Research:** Able to use the professional, managerial, interdisciplinary skill set, and domain specific tools in development processes, identify the research gaps, and provide innovative solutions to them.

## PROGRAM OUTCOMES (POs)

**Engineering Graduates should possess the following:**

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design / development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi disciplinary environments.
12. **Life- long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

# **MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY**

**Maisammaguda, Dhulapally Post, Via Hakimpet, Secunderabad – 500100**



## **DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

### **GENERAL LABORATORY INSTRUCTIONS**

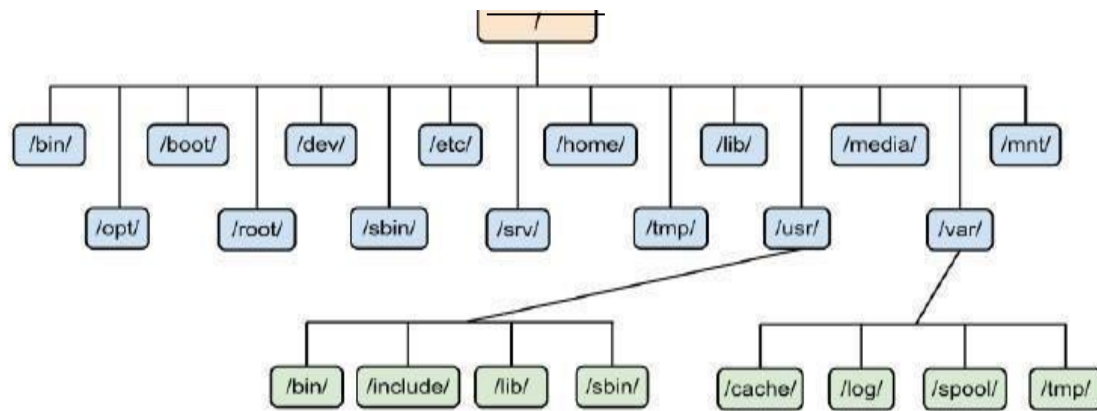
1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis/ program/ experiment details.
3. Student should enter into the laboratory with:
  - a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
  - b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.
  - c. Proper Dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system/ seat is kept properly.

**HEAD OF THE DEPARTMENT**

**PRINCIPAL**

# INDEX

S.No.	LIST OF PROGRAMS	Page no.
WEEK 1	a) To Install Ubuntu Linux and <b>LINUX Commands</b> (File Handling utilities, Text processing utilities, Network utilities, Disk utilities, Backup utilities and Filters)	1-
	b) Write a Shell Script that accepts a file name, starting and ending line numbers as arguments and displays all lines between the given line numbers.	
	c) Write a shell script that deletes all lines containing the specified word in one or more files supplied as arguments to it.	
	d) Write a shell script that displays a list of all files in the current directory to which the user has read, write and execute permissions.	
WEEK 2	a) Write a shell script that receives any number of file names as arguments checks if every argument supplied is a file or directory and reports accordingly. whenever the argument is a file it reports no of lines present in it	29-32
	b) Write a shell script that accepts a list of file names as its arguments, counts and reports the occurrence of each word that is present in the first argument file on other argument files.	
WEEK 3	a) Write a shell script to list all of the directory files in a directory	33-35
	b) Write a shell script to find factorial of a given number.	
WEEK 4	a) write an awk script to count number of lines in a file that does not contain vowels	36-38
	b) write an awk script to find the no of characters ,words and lines in a file	
WEEK 5	Implement in c language the following Unix commands using system calls a) cat      b) ls      c) mv	39-42
WEEK 6	Write a C program that takes one or more file/directory names as command line input and reports following information a) File Type                                  b) Number Of Links c) Time of last Access                        d) Read ,write and execute permissions	43-44
WEEK 7	Write a C program to list every file in directory, its inode number and file name	45
WEEK 8	a) Write a C program to create child process and allow parent process to display “parent”and the child to display “child” on the screen	46-48
	b) Write a C program to create zombie process	
	c) Write a C program to illustrate how an orphan process is created	
WEEK 9	a) Write a C program that illustrate communication between two unrelated process using named pipes	49-52
	b) Write a C program that receives a message from message queue and display them	
WEEK 10	a) Write a C program to allow cooperating process to lock a resource for exclusive use( using semaphore)	53-56
	b) Write a C program that illustrate the suspending and resuming process using signal	
	c) Write a C program that implements producer-Consumer system with two processes using semaphore	
WEEK 11	Write client server programs using c for interaction between server and client process using Unix Domain sockets	57-60
WEEK 12	Write a C program that illustrates two processes communicating using Shared memory	61-62



### System Administration Tools

1. UNIX comes with its own tools such as SAM on HP-UX.
2. Suse Linux comes with Yast
3. Redhat Linux comes with its own gui tools called redhat-config-\*.

However, editing text config file and typing commands are most popular options for sys admin work under UNIX and Linux.

### UNIX Operating System Names

*A few popular names:*

1. HP-UX
2. IBM AIX
3. Sun Solairs
4. Mac OS X
5. IRIX

### Linux Distribution Names

*A few popular names:*

1. Redhat Enterprise Linux
2. Fedora Linux
3. Debian Linux
4. Suse Enterprise Linux
5. Ubuntu Linux



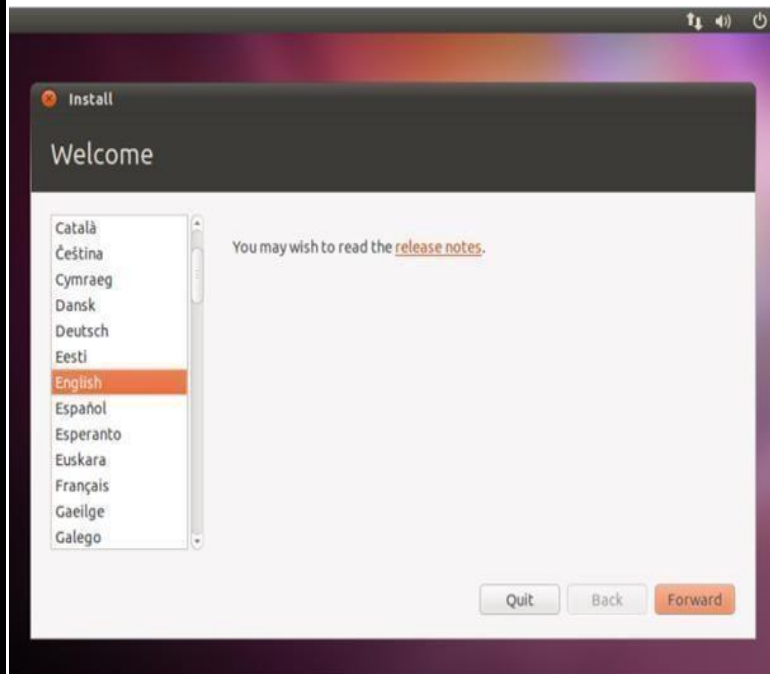
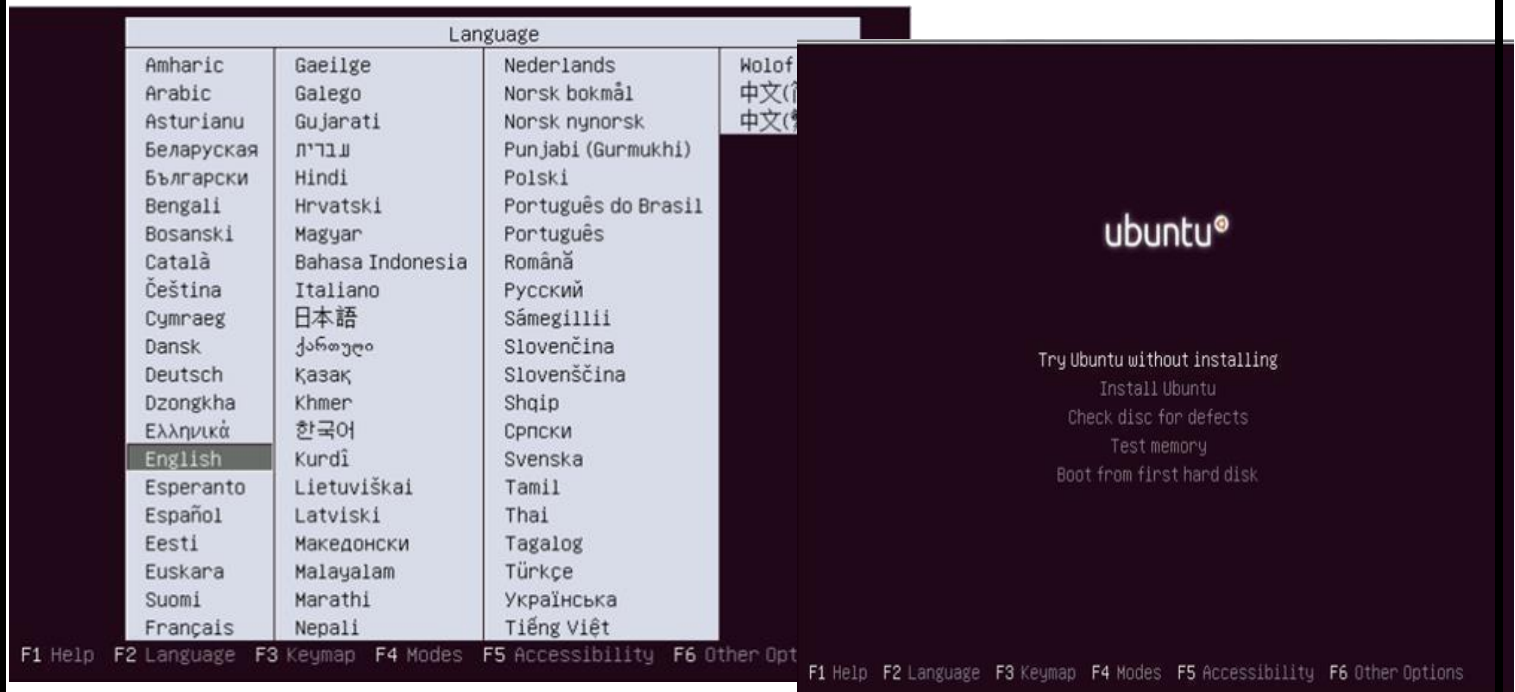


## EXPERIMENT NO: 1. a)

### To Install Ubuntu Linux – Complete Step by Step

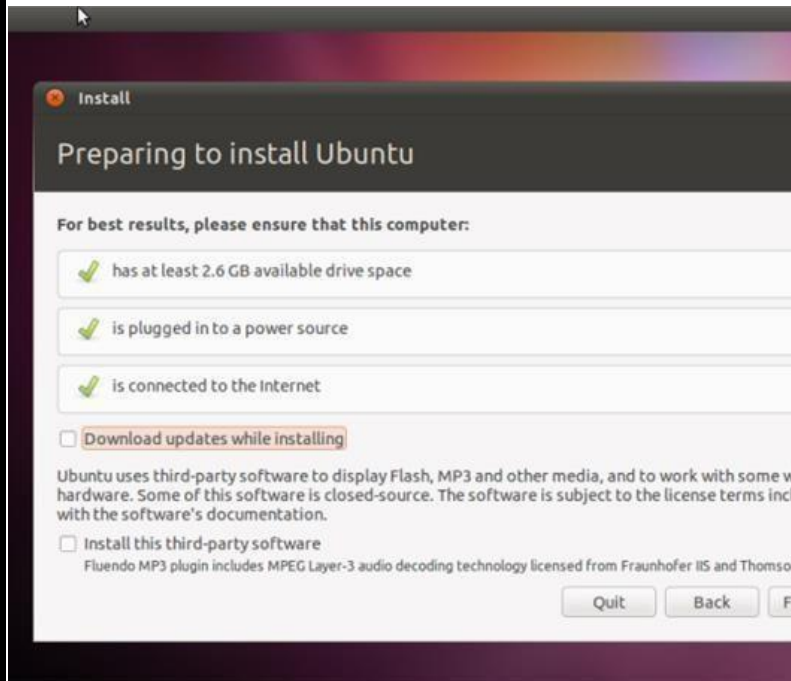
**Step 1 :** Insert the ubuntu cd in the cd drive and boot the computer from cd. First of all you will be prompted to select language. elect English or other language according to your preferences.

**Step 2 :** Now you will see ubuntu menu, you can choose **Try ubuntu without installing** option to try ubuntu without actually installing it on your hard drive. For installing ubuntu choose the second option **Install Ubuntu**.

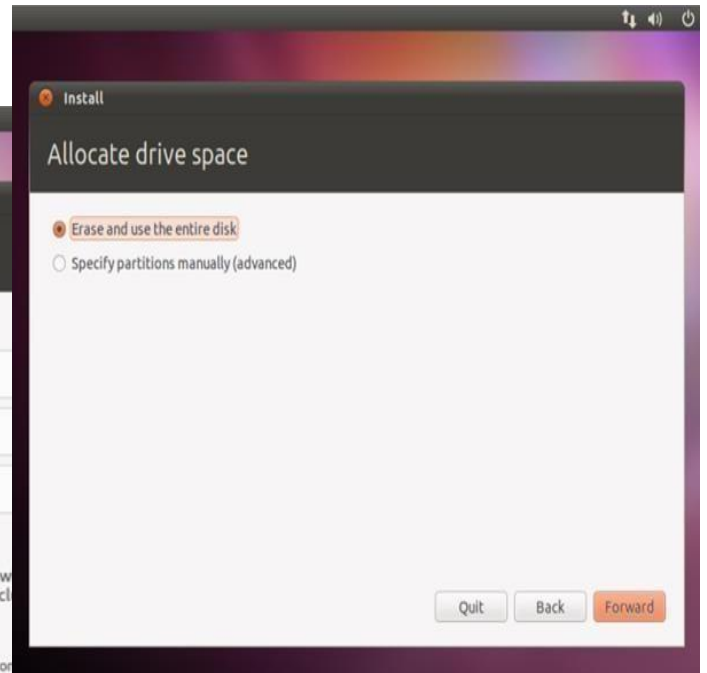


**Step 3 :** Ubuntu will start now initialize and after few minutes you can see the installation wizard.

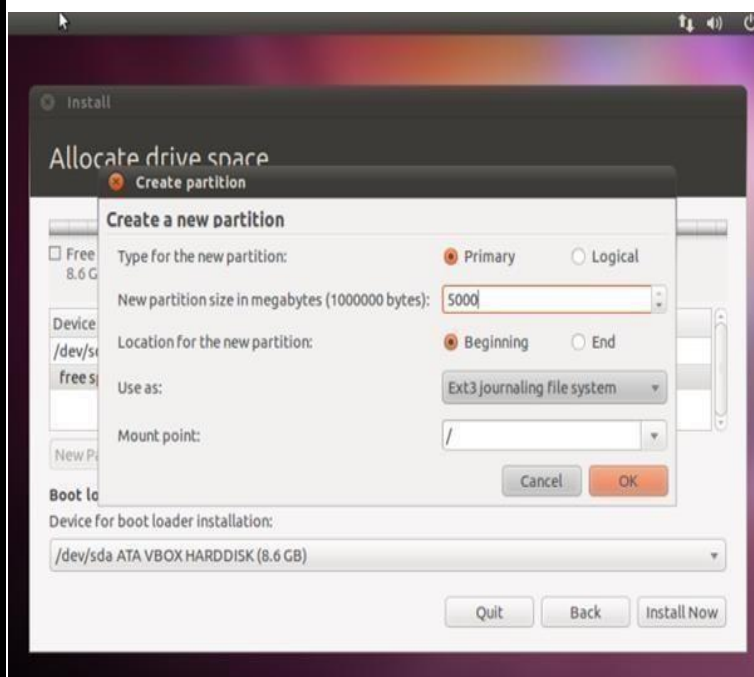
**Step 4 :** Click Forward and it will check the **minimum requirements for running ubuntu** on your PC. If everything is fine you can see green colored tick marks. You can also select to download updates while installing and install some third party software. After selecting the things you want click forward.



**manually** option. You can choose the 1st option if you just want linux to exist in your system. Else select second option. Now it will display the free space available for your pc.

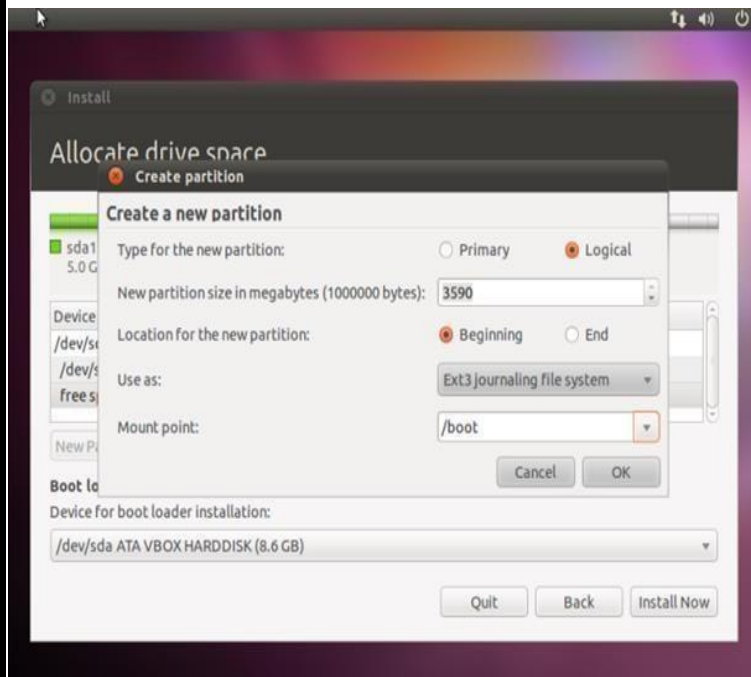


**Step 5 :** Now you can choose either erase and use entire disk option or **specify partitions**

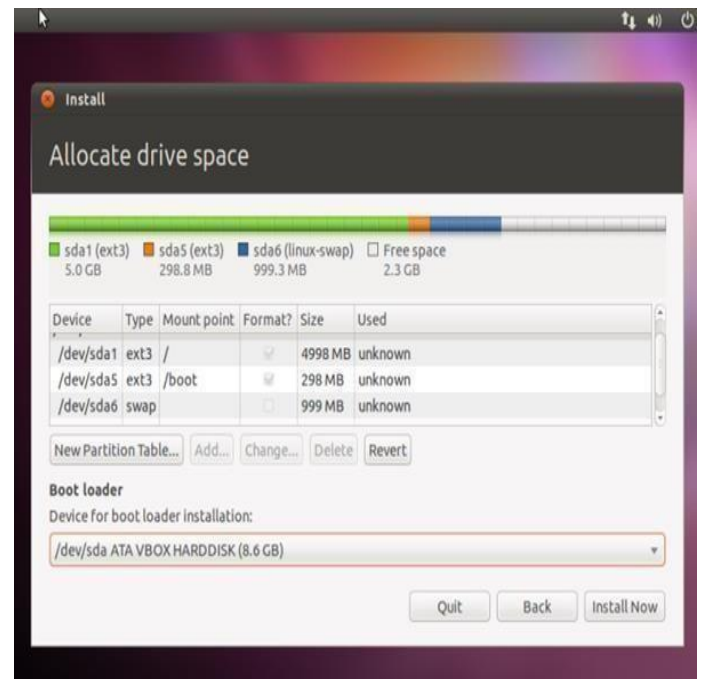
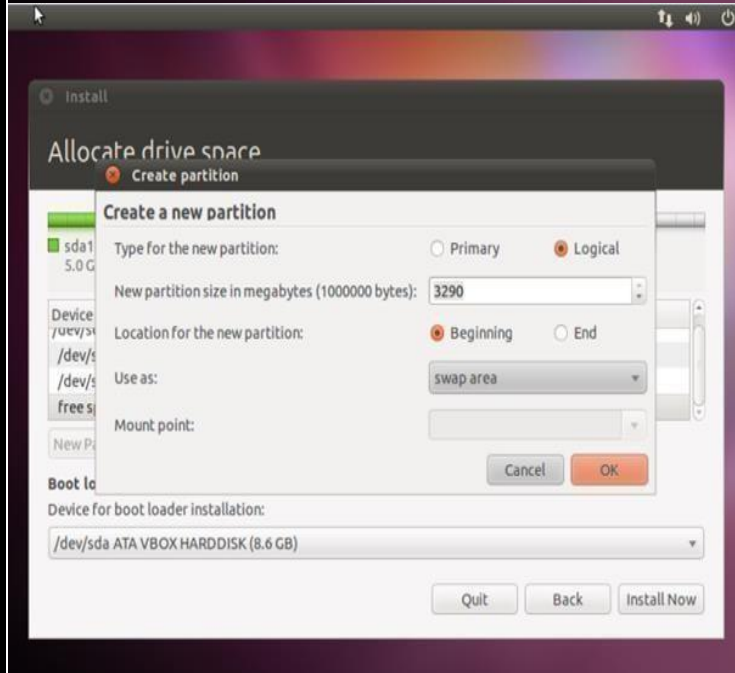


Select free space and click on Add option to create a new partition and choose partition type as primary, size around 70% of the free space available or choose anything like 10,000 or 20,000mb, use as ext3 journaling file system and select mount point as /.

Now again select free space from the table and click add option. Now select size to be around 300mb, use as ext3 journaling file system and select mount point as /boot.



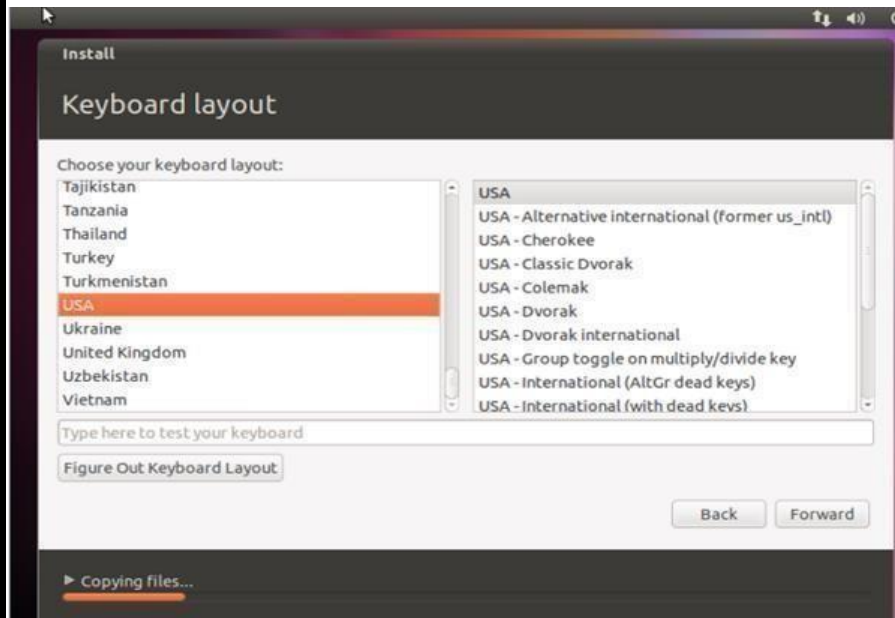
Now again select free space from the table and click add option. Now select size to be around twice the size of your ram that is around 1000 mb if your ram size is 512mb and select use as swap area and click ok.



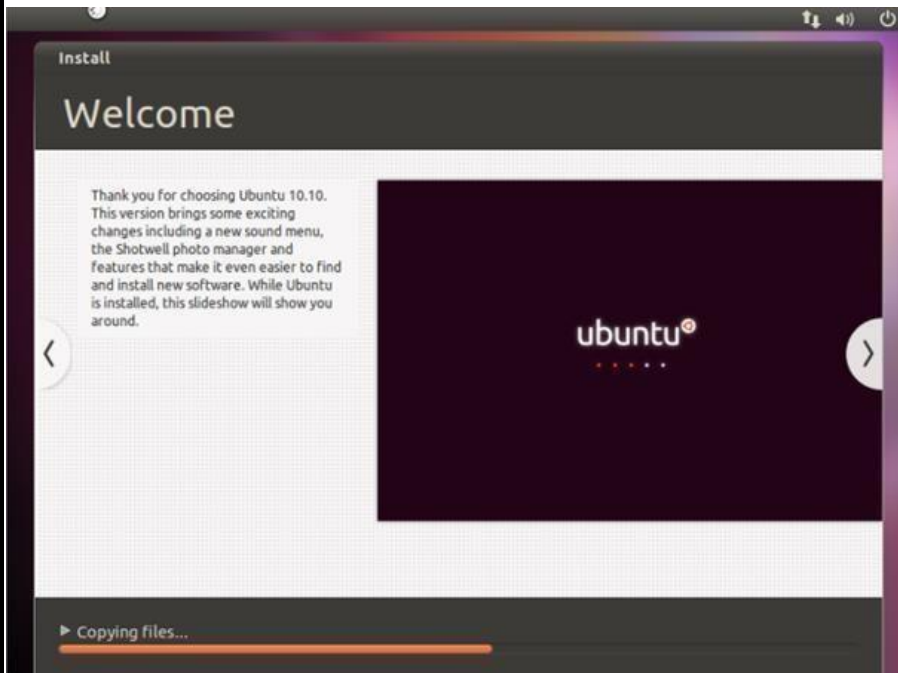
**Step 6 :** Click Install now button and then the wizard will ask you location. Select your location and click forward.



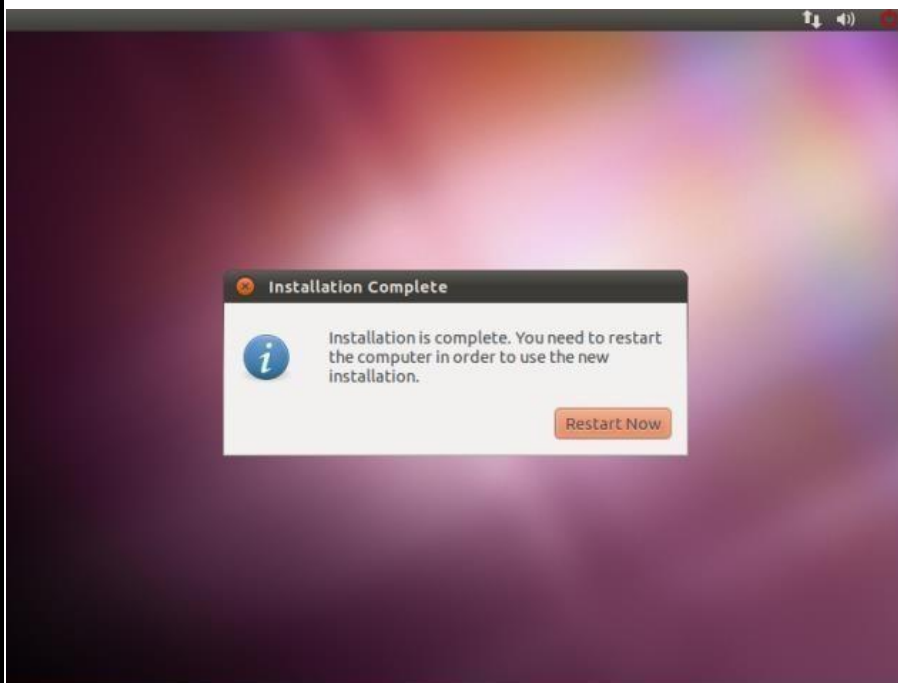
**Step 7 :** While you are selecting these options wizard will continue to copy files. Now select your desired keyboard layout and click forward.



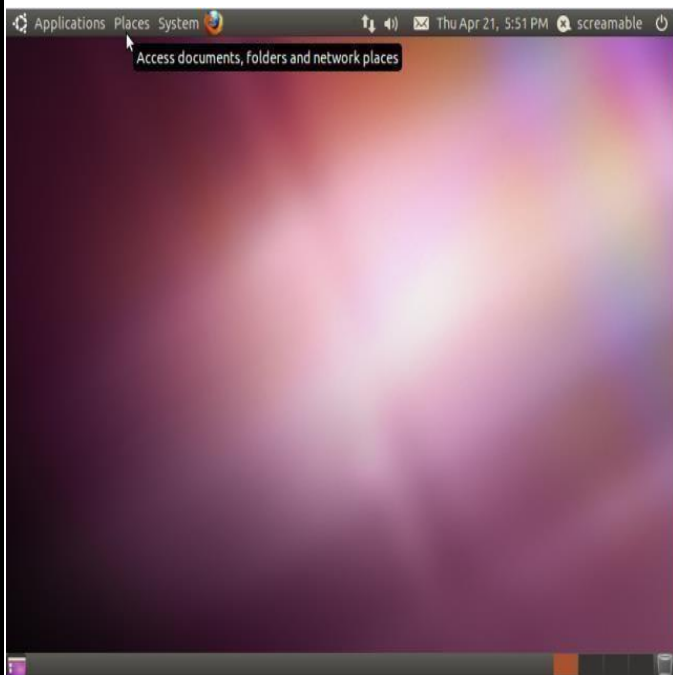
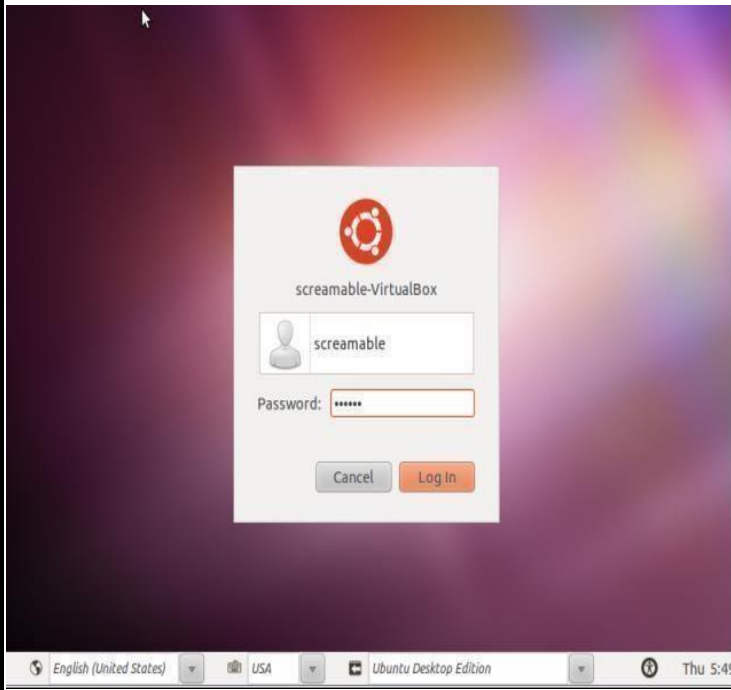
**Step 8 :** Now fill in the details about yourself. Fill your name, computer name, choose a username and create a password and click forward and let ubuntu copy all the essential files.



**Step 9 :** After all files have been copied and installed ubuntu will display a message saying that installation complete and click on restart button to restart your computer. Remove the cd from the cd drive.



**Step 10 :** After restarting your pc wait for the ubuntu to load and then it will display the login screen. Choose the user and enter password and click login.





# A-Z Index of the **Bash** command line for Linux

## a

**alias** Create an alias •  
**apropos** Search Help manual pages (man -k)  
**apt-get** Search for and install software packages (Debian/Ubuntu) **aptitude** Search for and install software packages (Debian/Ubuntu)  
**aspell** Spell Checker  
**awk** Find and Replace text, database sort/validate/index

## b

**basename** Strip directory and suffix from filenames  
**bash** GNU Bourne-Again SHell  
**bc** Arbitrary precision calculator language  
**bg** Send to background  
**bind** Set or display readline key and function bindings • **break** Exit from a loop •  
**builtin** Run a shell builtin  
**bzip2** Compress or decompress named file(s)

## c

**cal** Display a calendar  
**case** Conditionally perform a command  
**cat** Concatenate and print (display) the content of files  
**cd** Change Directory  
**cfdisk** Partition table manipulator for Linux  
**chattr** Change file attributes on a Linux filesystem  
**chgrp** Change group ownership  
**chmod** Change access permissions  
**chown** Change file owner and group  
**chroot** Run a command with a different root directory  
**chkconfig** System services (runlevel)  
**cksum** Print CRC checksum and byte counts  
**clear** Clear terminal screen  
**cmp** Compare two files  
**comm** Compare two sorted files line by line  
**command** Run a command - ignoring shell functions • **continue** Resume the next iteration of a loop •  
**cp** Copy one or more files to another location  
**cron** Daemon to execute scheduled commands  
**crontab** Schedule a command to run at a later time  
**csplit** Split a file into context-determined pieces  
**curl** Transfer data from or to a server  
**cut** Divide a file into several parts

## d

**date** Display or change the date & time  
**dc** Desk Calculator  
**dd** Convert and copy a file, write disk headers, boot records

**ddrescue** Data recovery tool

**declare** Declare variables and give them attributes • **df** Display free disk space

**diff** Display the differences between two files • **diff3** Show differences among three files

**dig** DNS lookup

**dir** Briefly list directory contents • **dircolors** Colour

setup for `ls`

**dirname** Convert a full pathname to just a path • **dirs** Display list of remembered directories • **dmesg** Print kernel & driver messages

**du** Estimate file space usage

e

**echo** Display message on screen •

**egrep** Search file(s) for lines that match an extended expression • **eject** Eject removable media

**enable** Enable and disable builtin shell commands • **env** Environment variables

**eval** Evaluate several commands/arguments • **exec** Execute a command

**exit** Exit the shell

**expect** Automate arbitrary applications accessed over a terminal • **expand** Convert tabs to spaces

**export** Set an environment variable • **expr** Evaluate expressions

f

**false** Do nothing, unsuccessfully • **fdformat** Low-level format a floppy disk

**fdisk** Partition table manipulator for Linux • **fg** Send job to foreground

**fgrep** Search file(s) for lines that match a fixed string • **file** Determine file type

**find** Search for files that meet a desired criteria • **fmt** Reformat paragraph text

**fold** Wrap text to fit a specified width. • **for** Expand words, and execute *commands* • **format** Format disks or tapes

**free** Display memory usage

**fsck** File system consistency check and repair • **ftp** File Transfer Protocol

**function** Define Function Macros

**fuser** Identify/kill the process that is accessing a file

g

**gawk** Find and Replace text within file(s) • **getopts** Parse positional parameters

**grep** Search file(s) for lines that match a given pattern • **groupadd** Add a user security group

**groupdel** Delete a group



**groupmod** Modify a group  
**groups** Print group names a user is in  
**gzip** Compress or decompress named file(s)  
**hash** Remember the full pathname of a nameargument  
**head** Output the first part of file(s)  
**help** Display help for a built-in command • **history**  
 Command History  
**hostname** Print or set system name

**iconv** Convert the character set of a file  
**id** Print user and group id's  
**if** Conditionally perform a command  
**ifconfig** Configure a network interface  
**install** Copy files and set attributes  
**ip** Routing, devices and tunnels

**jobs** List active jobs •  
**join** Join lines on a common field

**kill** Kill a process by specifying its PID  
**killall** Kill processes by name

**let** Perform arithmetic on shell variables • **link** Create a link to a file

**ln** Create a symbolic link to a file  
**local** Create a function variable • **locate** Find files

**logname** Print current login name  
**logout** Exit a login shell •

**look** Display lines beginning with a given string  
**lpc** Line printer control program

**lpr** Off line print  
**lprint** Print a file  
**lprintd** Abort a print job  
**lprintq** List the print queue

**ls** List information about file(s)  
**lsof** List open files

**make** Recompile a group of programs  
**man** Help manual

**mkdir** Create new folder(s)  
**mkfifo** Make FIFOs (named pipes)  
**mkfile** Make a file

**mktemp** Make a temporary file

**more** Display output one screen at a time  
**most** Browse or page through a text file

**mount** Mount a file system **mttools**  
Manipulate MS-DOS files

**mtr** Network diagnostics(traceroute/ping)**mv** Move or  
rename files or directories **mmv** Mass Move and rename (files)

**nc** Netcat, read and write data across networks**netstat** Networking  
connections/stats

**nice** Set the priority of a command or job**nl** Number lines  
and write files

**nohup** Run a command immune to hangups**notify-**  
**send** Send desktop notifications

**nslookup** Query Internet name servers interactively

**open** Open a file in its default application**op** Operator access

**passwd** Modify a user password **paste**  
Merge lines of files **ping** Test  
a network connection**pgrep** List processes by  
name **pskill** Kill processes by name

**popd** Restore the previous value of the current directory**pr** Prepare files for  
printing

**printcap** Printer capability database **printenv** Print  
environment variables**printf** Format and print data •

**ps** Process status

**pushd** Save and then change the current directory **pv** Monitor the  
progress of data through a pipe **pwd** Print Working Directory

**quota** Display disk usage and limits **quotacheck** Scan a  
file system for disk usage

**ram** ram disk device

**rar** Archive files with compression **rcp** Copy files  
between two machines **read** Read a line from standard input  
•

**readarray** Read from stdin into an array variable•**readonly** Mark  
variables/functions as readonly **reboot** Reboot the system

**rename** Rename files

**renice** Alter priority of running processes**remsync**  
Synchronize remote files via email **return** Exit a shell function

**rev** Reverse lines of a file**rm**  
Remove files

**rmdir** Remove folder(s)

s

**screen** Multiplex terminal, run remote shells via ssh  
**scp** Secure copy (remote file copy)

**sdiff** Merge two files interactively  
**sed** Stream Editor

**select** Accept keyboard input  
**seq** Print numeric sequences

**set** Manipulate shell variables and functions  
**ssftp** Secure File Transfer Program

**shift** Shift positional parameters  
**shopt** Shell Options

**shutdown** Shutdown or restart linux  
**sleep** Delay for a specified time  
**locate** Find files

**sort** Sort text files

**source** Run commands from a file '.'

**split** Split a file into fixed-size pieces  
**ss** Socket Statistics

**ssh** Secure Shell client (remote login program)  
**su** Substitute user identity

**sudo** Execute a command as another user  
**sum** Print a checksum for a file  
**suspend** Suspend execution of this shell •

t

**tail** Output the last part of file

**tar** Store, list or extract files in an archive  
**tee** Redirect output to multiple files

**test** Evaluate a conditional expression  
**time** Measure Program running time  
**timeout** Run a command with a time limit

**times** User and system times

**touch** Change file timestamps

**top** List processes running on the system

**tput** Set terminal-dependent capabilities, color, position  
**traceroute** Trace Route to Host

**trap** Execute a command when the shell receives a signal •  
**tr** Translate, squeeze, and/or delete characters

**true** Do nothing, successfully  
**tsort** Topological sort

**tty** Print filename of terminal on stdin  
**type** Describe a command •

u

**ulimit** Limit user resources •  
**umask** Users file creation mask  
**umount** Unmount a device

**unalias** Remove an alias •  
**uniq** Uniquify files  
**units** Convert units from one scale to another  
**until** Execute commands (until error)  
**uptime** Show uptime  
**useradd** Create new user account  
**userdel** Delete a user account  
**usermod** Modify user account  
**users** List users currently logged in

**V**  
**V** Verbosely list directory contents (`ls -l -b')  
**vdir** Verbosely list directory contents (`ls -l -b')Text Editor  
**vi** Report virtual memory statistics  
**vmstat**

**W**  
**w wait** Show who is logged on and what they are doingWait for a process to  
**watch** complete • Execute/display a program periodically  
**wc** Print byte, word, and line counts  
**whereis** Search the user's \$path, man pages and source files

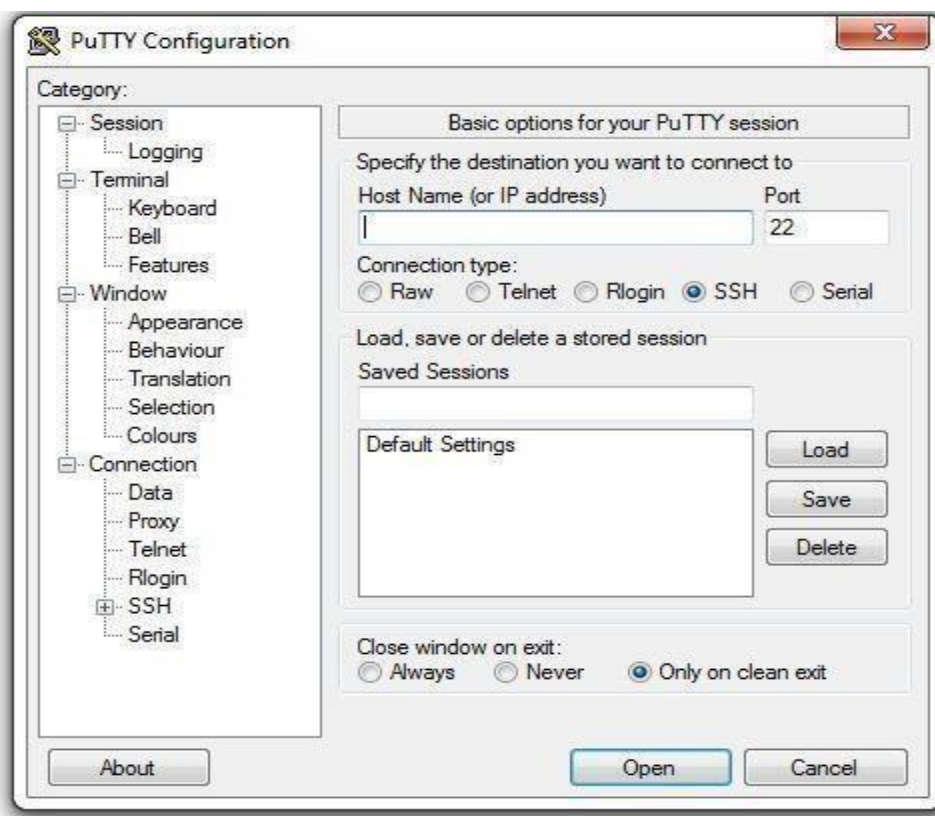
for a program

**which** Search the user's \$path for a program file**while** Execute  
 commands  
**who** Print all usernames currently logged in **whoami** Print the  
 current user id and name (`id-un')  
**wget** Retrieve web pages or files via HTTP, HTTPS orFTP**write** Send a  
 message to another user

**x**  
**xargs** Execute utility, passing constructed argument list(s) **xdg-open** Open a file or  
 URL in the user's preferredapplication.**xz** Compress or decompress .xz and .lzma files  
**yes** Print a string until interrupted  
**zip** Package and compress (archive) files.  
**.** Run a command script in the current shell  
**!!** Run the last command again  
**###** Comment / Remark

## Procedure to connect to LINUX(putty)

Step 1:click on putty icon available on desk top. A window is opened



Step 2:fill in ip address of linux server and click open



Step 3: provide login and password (nothing is displayed on screen while typing password)

Step 4: change the default password at your first login

### EXPERIMENT NO: 1. b)

**Aim: Write a Shell Script that accepts a file name, starting and ending line numbers as Arguments and displays all lines between the given line numbers.**

#### ALGORITHM:

Step 1: Create a file with 5-6 lines of data

File can be created by vi sample.dat or cat sample.dat

Step 2: Now write a shell script with

vi 1.sh

step3: Check the no of arguments for shell script

if 0 arguments then print no arguments

else if 1 argument then print 1 argument

else if 2 arguments then print 2 arguments

else check for file is there or not (if file is not there print file does not exist)

1 else sed -ne "\$2','\$3' p' \$1

sed is one of powerful filter (stream editor)

-e default option (script on command line)

-n suppresses automatic output

\$2 first line number passed \$3 2nd line number passed

p is a print command (prints current content to the pattern space).

\$1 is name of file from which we are printing data between the line numbers.

Step 4: top

#### Script Name: 1sh

```
#!/bin/bash
```

```
if [ $# -lt 3 ]
```

```
then
```

```
    echo "To execute you have to enter 3 arguments in command line in following order..."
```

```
    echo " File Name ,starting line number and ending line number..."
```

```
else
```

```
    sed -n $2,$3p $1
```

```
fi
```

#### Commands used in the script:

##### Sed command:

stream editor for filtering and transforming text

##### 1. Replacing or substituting string

Sed command is mostly used to replace the text in a file. The below simple sed command replaces the word "unix" with "linux" in the file.

```
$sed 's/unix/linux/' file.txt
```

##### 2. Replacing the nth occurrence of a pattern in a line

```
$sed 's/unix/linux/2' file.txt
```

Replaces 2<sup>nd</sup> occurrence

##### 3. printing lines for a given range

```
$sed -n 1,5p hello.txt Prints first 5 lines in the file hello.txt
```

**nl command:**

The nl utility in Linux is used to give number lines of a file on console.

Example:

```
$ nl sort.txt
1    UK
2    Australia
3    Newzealand
4    Brazil
5    America
```

**Execution:**

check how many lines of data in the input file

```
root@localhost sh)# cat hello.txt | nl
```

```
1 abc
2 def
3 ghi
4 abc
5 abc
6 cccc
```

Executing Shell script:

run1:

```
[root@localhost sh)# sh 1.sh abc1.txt 2 4
```

```
def
ghi
abc
```

compare with the data in the file and output

**Viva Questions**

1. What is a shell script?
2. How to find current shell name
3. How to switch to another shell
4. How to execute shell Script

**Exercises:**

S.No.	Task
1	Write a shell script to count no of character in a file ,prompt for inputfile
2	Write a shell script to count no of character in a file name given in command prompt
3	Write a shell script to perform arithmetic operation using case statement

## EXPERIMENT NO: 1. c)

**AIM: Write a shell script that deletes all lines containing the specified word in one or more files Supplied as arguments to it.**

**ALGORITHM:**

Step 1: Create a file with 5-6 lines of data  
Create the 2 file f1 and f2 as vi s1 and vi s2  
Step2: Now write a shell script with  
vi 2.sh  
step3: Check the no of arguments for shell script  
if 0 arguments then print no arguments  
else pattern=\$1 (word will be stored in pattern)  
for fname in \$\*  
do  
for every filename in given files  
if it is a file if [ -f \$fname ] then  
print DELETING \$pattern FROM  
\$fname sed '/\$pattern/d' \$fname  
sed acts as filter if word is a file in any line that will be deleted  
'/' is used to represent regular expressions  
'/d' is a delete command in sed  
else print file NOT FOUND

**Script name: 2.sh**

```
#!/bin/bash
if [ $# -lt 2 ] then
    echo "Enter atleast two files as input in command line"
else
    printf "enter a word to find:"
    read word
    for f in $*
    do
        printf "\n In File $f:\n"
        sed /$word/d $f
    done
fi
```

**Execution:**

run1:  
check data in input files  
[root@localhost sh]# cat abc1.txt  
abc  
def  
ghi  
abc  
abc  
cccc  
[root@localhost sh]# cat abc2.txt  
abc



```
def
ghi
abc
abc
cccc
Executing shell script
[root@localhost sh]# sh 2.sh abc1.txt abc2.txt
enter a word to find:abc
In File abc1.txt:
def
ghi
cccc
In File abc2.txt:
def
ghi
cccc
```

**Expected output:**

Displays lines from files s1 s2 after deleting the word hi

**Viva Questions**

- 1.Explain various loops in shell script
- 2.Explain grep
- 3.Explain egep
- 4.Explain fgep
5. .Explain sed

**Exercises:**

S.No.	Task
1	Write a shell script to count occurrence of a word in a file
2	Write a shell script to print line numbers in which a particular word has occurred where word is provides as input.

## EXPERIMENT NO: 1. d)

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

### ALGORITHM:

Step1: selects list of files from present working directory  
Step 2: check for each file whether it has read, write and execute permissions if true goto step 3  
Step 3: print file  
Step 4 :stop

### Script name: 3.sh

```
#!/bin/bash
echo "List of Files which have Read, Write and Execute Permissions in Current Directory are..."
for file in *
do
    if [ -r $file -a -w $file -a -x $file ]
    then
        echo $file
    fi
done
```

Execution:

\$sh 3.sh

Expected output:

by executing above shell script you will get all files which have read, write and execute permissions in current working directory

### sample output

```
[root@localhost sh]# sh 3.sh
List of Files which have Read, Write and Execute Permissions in Current Directory are...
5.sh
a.out
```

### Viva Questions:

1. Display all files in a directory
2. how to use chmod
3. How to change file permissions

### Exercises:

S.No.	Task
1	Write a shell script to display all files with read or write or execute permissions provide a selection menu
2	Write a comparison report for using chmod using symbolic representation or octal number representation
3	Write a shell script to count no of files in current directory with full permissions

### EXPERIMENT NO: 2.a)

**Aim:-**Write a shell script that receives any number of file names as arguments checks if every argument supplied is a file or directory and reports accordingly. Whenever the argument is a file it reports no of lines present in it

#### ALGORITHM:

step 1: if arguments are less than 1 print Enter at least one input file name and goto step 9  
Step 2: selects list a file from list of arguments provided in command line  
Step 3: check for whether it is directory if yes print is directory and goto step 9  
step 4: check for whether it is a regular file if yes goto step 5 else goto step 8  
Step 5: print given name is regular file  
step 6: print No of lines in file  
step 7: goto step  
step 8: print not a file or adirectory  
step 9: stop

#### Script name: 4.sh

```
#!/bin/bash
if [ $# -lt 1 ]
then
    echo "Enter at least one input file name"
else
    for i in $*
    do
        if [ -d $i ]
        then
            echo " given name is directory"
        elif [ -f $i ]
        then
            echo " given name is file: $i"
            echo " No of lines in file are : `wc -l $i`"
        else
            echo "given name is not a file or a directory"
        fi
    done
fi
```

#### Execution:

provide two file names as input one a regular file and other directory  
for example abc1.txt a text file as first argument and vazralu a directory as second argument

#### Run1:

```
[root@localhost sh]# sh 4.sh abc1.txt vazralu
given name is file: abc1.txt
No of lines in file are : 7 abc1.txt
```

vazralu is directory

**run 2:**[root@localhost sh]# sh 4.sh abc1.txt abc2.txt  
given name is file: abc1.txt  
No of lines in file are : 7 abc1.txt  
given name is file: abc2.txt  
No of lines in file are : 7 abc2.txt

### Viva Questions:

1. What is an internal command in Linux?

Internal commands are also called shell built-in commands. Example: cd,fg. Since these are shell built-in, no process is created while executing these commands, and hence are considered to be much faster.

2. x and y are two variables containing numbers? How to add these 2 numbers?

\$ expr \$x + \$y

3. How to add a header record to a file in Linux?

\$ sed -i '1i HEADER' file

4. How to find the list of files modified in the last 30 mins in Linux?

\$ find . -mmin -30

5. How to find the list of files modified in the last 20 days?

\$ find . -mtime -20

### Exercises:

S.No.	Task
1	Write a shell script to count no of regular files in the current workingdirectory
2	Write a shell script to display list of currently logged users

## EXPERIMENT NO: 2. b)

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

**ALGORITHM:**

step1: Check the no of arguments for shell script  
if 0 arguments then print no arguments  
step2:else translate each word in the first file is to be on separate line  
which will be stored in temp file  
step3: for i in \$\*  
for every filename in given files  
step 4: translate each word in the file is to be on separate line  
which will be stored in temp1 file  
step5: count no of lines in temp file assign it to j  
step6: initialize j=1  
step 7: while i< j  
extract the line that are common in both the file by using  
head and tail commands  
then apply the filter grep to count and print the lines  
which are common to files  
increment j  
step 8: stop

**Script name:5.sh**

```
#!/bin/bash
echo "no of arguments $#"
```

```
if [ $# -le 2 ]
then
    echo "Error : Invalid number of arguments."
    exit
fi
str=`cat $1 | tr '\n' ' '`
for a in $str
do
    echo "in file $a"
    echo "Word = $a, Count = `grep -c "$a" $2`"
done
```

Execution and output:

check data in abc1.txt file

```
[root@localhost sh]# cat abc1.txt
```

abc

def

ghi

abc

abc

```

cccc
check data in abc1.txt file
[root@localhost sh]# cat abc2.txt
abc
def
ghi
abc
abc
cccc

```

### executing script

```

[root@localhost sh]# sh 5.sh abc1.txt abc2.txt
Word = abc, Count = 3
Word = def, Count = 1
Word = ghi, Count = 1
Word = abc, Count = 3
Word = abc, Count = 3
Word = cccc, Count = 1

```

### Viva Questions

#### 1. What is Shell Scripting ?

Shell scripting, in Linux or Unix, is programming with the shell using which you can automate your tasks. A shell is the command interpreter which is the interface between the User and the kernel. A shell script allows you to submit a set of commands to the kernel in a batch. In addition, the shell itself is very powerful with many properties on its own, be it for string manipulation or some basic programming stuff.

#### 2. The command "cat file" gives error message "--bash: cat: Command not found". Why?

It is because the PATH variable is corrupt or not set appropriately. And hence the error because the cat command is not available in the directories present PATH variable.

#### 3. How to find the length of a string in Linux?

```
$ x="welcome" $ echo ${#x} 7
```

#### 4. What are the different timestamps associated with a file?

Modification time:- Refers to the time when the file is last modified.

Access time :- The time when the file is last accessed.

Changed time :- The time when the attributes of the file are last changed.

#### 5. How to get the list of files alone in a directory in Linux?

```
$ ls -lrt | grep ^-
```

### Exercises:

S.No.	Task
1	Write a shell script to print prime numbers
2	Write a shell script to print Fibonacci numbers

### EXPERIMENT NO: 3. a)

**Aim:-Write a shell script to list all of the directory files in a directory.**

Algorithm:

Step1: enter the name of the directory

Read dir

Step2: if it is a directory

Then list the files present in that directory

By using ls command with -p option to list all directory files in a given directory

Step 3: else enter the directory name

Step 4: stop

**Script name: 6.sh**

```
#!/bin/bash
echo " Enter dir name: "
read dir
if [ -d $dir ]
then
    printf " Files in Directory $dir are...\n`ls $dir`"
else
    echo " Dir does not exist"
fi
```

Execution and output:

```
[root@localhost sh]# sh 6.sh
```

Enter dir name:

japs

Files in Directory japs are...

abc1.txt

abc2.txt

ls-l.c

prg5

sl

Viva Questions

1. A string contains a absolute path of a file. How to extract the filename alone from the absolute path in Linux?

```
$ x="/home/guru/temp/fl.txt"
```

```
$ echo $x | sed 's^.*/^'
```

2. How to find all the files created after a pre-defined date time, say after 10th April 10AM?

This can be achieved in 2 steps:

1. Create a dummy file with the time stamp, 10th April 10AM.

2.Find all the files created after this dummy file.

```
$ touch -t 1004101000 file
```

```
$ find . -newer file
```

3. The word "Unix" is present in many .txt files which is present across many files and also files present in sub directories. How to get the total count of the word "Unix" from all the .txt files?

```
$ find . -name *.txt -exec grep -c Unix '{}' \; | awk '{x+=$0;}END{print x}'
```

### Exercises:

S.No.	Task
1	How to find the files modified exactly before 30minutes? \$ find . -mmin 30
2	How to print the contents of a file line by line in Linux?



### EXPERIMENT NO: 3. b)

**Aim:-Write a shell script to find factorial of a given number.**

#### ALGORITHM

Step 1: read any number to find factorial

Step 2: initialize fact=1 and i=1

Step 3: while i less than

do

fact=fact\* i

i=i+1

done

step 4:print fact

step 5:stop.

#### Script Name:7.sh

```
#!/bin/bash
```

```
echo "Factorial Calculation Script. .."
```

```
echo "Enter a number: "
```

```
read f
```

```
fact=1
```

```
factorial=1
```

```
while [ $fact -le $f ]
```

```
do
```

```
    factorial=`expr $factorial \* $fact`
```

```
    fact=`expr $fact + 1`
```

```
done
```

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

Execution and Output:

```
[root@localhost sh]# sh 7.sh
```

```
Factorial Calculation Script....
```

```
Enter a number: 4
```

```
Factorial of 4 = 24
```

#### Exercises:

S.No.	Task
1	Write a shell script to find sum of first n natural numbers
2	Write a shell script to find largest of given three numbers

## EXPERIMENT NO: 4. a)

**Aim:-write an awk script to count number of lines in a file that does not contain vowels**

### ALGORITHM

Step 1: create a file with 5-10 lines of data

Step 2: write an awk script by using grep command to filter the lines that do not contain vowels

```
awk ' $0 !~/aeiou/ {print $0}' file1
```

step3: count=count+1

step4:print count

step5:stop

**Awk script name:**nm.awk

```
BEGIN{ }
{
  If($0 !~/[aeiou AEIOU]/)
    wordcount+=NF
}
END
{
  print "Number of Lines are", wordcount
}
```

**input file for awk script:**data.dat

```
bcd fghj
abcd fghj
bcd fghj
ebcd fghj
bcd fghj
ibcd fghj
bcd fghj
obcd fghj
bcd fghj
ubcd fghj
```

Executing the script:

```
[root@localhost awk]# awk -f nm.awk data.dat
```

```
bcd fghj
bcd fghj
bcd fghj
bcd fghj
bcd fghj
```

Number of lines are 5

### Exercises:

S.No.	Task
1	Write an awk script to find square root of a given number
2	Write an awk script to find maximum of two numbers , read input from keyboard

## EXPERIMENT NO: 4. b)

**Aim:-write an awk script to find the no of characters ,words and lines in a file**

### ALGORITHM

Step 1: create a file with 5 to10 lines of data

Step 2: write an awk script  
find the length of file  
store it in chrnt

step3: count the no of fields (NF), store it in wordcount

step4: count the no of records (NR), store it in NR

step5: print chrnt,NRwordcount

step6: stop

**Awk script name:nc.awk**

```
BEGIN{ }  
{  
    print len=length($0),"\t",$0  
    wordcount+=NF  
    chrnt+=len  
}  
END {  
    print "total characters",chrnt  
    print "Number of Lines are",NR  
    print "No of Words count:",wordcount  
}
```

**input data file name:data.dat**

```
bcdfghj  
abcdfghj  
bcdfghj  
ebcdfghj  
bcdfghj  
ibcdfghj  
bcdfghj  
obcdfghj  
bcdfghj  
ubcdfghj
```

Executing the script:

[root@localhost awk]# **awk -f nc.awk data.dat**

```
7 bcdfghj  
8 abcdfghj  
7 bcdfghj  
8 ebcdfghj  
7 bcdfghj
```

```
8 ibcdfghj
7 bcdfghj
8 obcdfghj
7 bcdfghj
8 ubcdfghj
total characters 75
Number of Lines are 10
No of Words count: 10
```

#### VIVA QUESTIONS:

1. How to find the last modified file or the newest file in a directory?

```
$ ls -lrt | grep ^- | awk 'END{print $NF}'
```

2. How to access the 10th command line argument in a shell script in Linux?

\$1 for 1st argument, \$2 for 2nd, etc... For 10th argument, \${10}, for 11th, \${11} and so on.

3. How to find the sum of all numbers in a file in Linux?

```
$ awk '{x+=$0}END{print x}' file
```

4. How to delete a file which has some hidden characters in the file name?

Since the rm command may not be able to delete it, the easiest way to delete a file with some hidden characters in its name is to delete it with the find command using the inode number of the file.

```
$ ls -li
```

```
total 32
```

```
9962571 -rw-r--r-- 1 guru users 0 Apr 23 11:35
```

```
$ find . -inum 9962571 -exec rm '{}' \;
```

5. Using the grep command, how can you display or print the entire file contents?

```
$ grep '.*' file
```

6. What is the difference between a local variable and environment variable in Linux?

A local variable is the one in which the scope of the variable is only in the shell in which it is defined. An environment variable has scope in all the shells invoked by the shell in which it is defined.

## EXPERIMENT NO: 5

**Aim: Implement in c language the following Unix commands using systemcalls a)cat b)ls c)mv**

a) AIM:- Write a c program to implement **cat command** using system calls

**Description:**

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

**SYNTAX:**

cat [OPTIONS] [FILE]...

**OPTIONS:**

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

**Operations With cat Command:**

1. To Create a new file:

**\$cat > file1.txt**

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

2. To Append data into the file:

**\$cat >> file1.txt**

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

3. To display a file:

**\$cat file1.txt**

This command displays the data in the file.

4. To concatenate several files and display:

**\$cat file1.txt file2.txt**

The above cat command will concatenate the two files (file1.txt and file2.txt) and it will display the output in the screen. Some times the output may not fit the monitor screen. In such situation you can print those files in a new file or display the file using less command.

cat file1.txt file2.txt | less

5. To concatenate several files and to transfer the output to another file.

**\$cat file1.txt file2.txt > file3.txt**

In the above example the output is redirected to new file file3.txt. The cat command will create new file file3.txt and store the concatenated output into file3.txt.

**Algorithm:**

Step 1:Start  
Step 2:read arguments from keyboard at command line  
Step 3:if no of arguments are less than two print ENTER CORRECT ARGUMENTS  
Else goto step 4  
Step4:read the date from specified file and write it to destination file  
Step 5 :stop

**Program file name:catdemo.c**

```
#include<stdio.h>
#include<sys/types.h>
#include<stdlib.h>
#include<fcntl.h>
#include<sys/stat.h>
int main(int argc,char *argv[])
{
int fd,n;
char buff[512];
if(argc!=2)
printf("ENTER CORRECT ARGUMENTS :");
if((fd=open(argv[1],4))<0)
{
printf("ERROR");
return 0;
}
while(n=read(fd,buff,sizeof(buff))>0)
write(1,buff,n);
}
```

b) **AIM:-** Write a c program to implement **ls command** using systemcalls

Description:

**ls command** is used to list the files present in a directory

**Algorithm:**

Step 1. Start.  
Step 2. open directory using opendir( ) system call.  
Step 3. read the directory using readdir( ) system call.  
Step 4. print dp.name and dp.inode .  
Step 5. repeat above step until end of directory.  
Step 6: Stop.

**Program name: lsdemo.c**

```
#include<stdio.h>
#include<dirent.h>
void quit(char*,int);
int main(int argc,char **argv )
{
```

```

    DIR *dirop;
    struct dirent *dired;
    if(argc!=2)
    {
        printf("Invalid number of arguments\n");
    }
    if((dirop=opendir(argv[1]))==NULL)
        printf("Cannot open directory\n");
    while((dired=readdir(dirop))!=NULL)
        printf("%10d %s\n",dired->d_ino,dired->d_name);
    closedir(dirop);
}

```

c) **AIM:** write a c program that simulates **mv command** (using system calls)

**Description:**

**mv command** is used to move or rename a file

synatax:

mv file1 file2

here file1 is renamed as file2

**Algorithm:**

Step 1: Start

Step 2: open an existed file and one new open file using open() system call

Step 3: read the contents from existed file using read( ) system call

Step 4: write these contents into new file using write system call using write( ) system call

Step 5: repeat above 2 steps until eof

Step 6: close 2 file using fclose( ) system call

Step 7: delete existed file using unlink( ) system

Step 8: Stop.

**Program File name: mvdemo.c**

```

#include<stdio.h>
#include<string.h>
int main(int argc ,char *argv[])
{
    int r,i;
    char p[20],q[20];
    if(argc<3)
        printf("improper arguments\n file names required\n");
    else
        if( argc==3)
        {
            printf("\n%s\n",argv[1],argv[2]);
            r=link(argv[1],argv[2]);
            printf("%d\n",r);
            unlink(argv[1]);
        }
    }

```

```
    }  
    else  
    {  
        for(i=1;i<argc-1;i++)  
        {  
            strcpy(p,argv[argc-1]);  
            strcat(p,"/");  
            strcat(p,argv[i]);  
            printf("%s%s\n",argv[i],p);  
            link(argv[i],p);  
            unlink(argv[i]);  
        }  
    }  
}
```



## EXPERIMENT NO: 6

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

A) File Type

B) Number Of Links

c) Time of last Access

D) Read, write and execute permissions

Algorithm:

Step 1: start

Step 2: Declare struct stat a

Step 3: read arguments at command line

Step 4: set the status of the argument using stat(argv[i], &a);

Step 5: Check whether the given file is Directory file by using S\_ISDIR(a.st\_mode)  
if it is a directory file print Directory file

Else

print is Regular file

Step 6: print number of links

Step 7: print last time access

Step 8: Print Read, write and execute permissions

Step 9: stop

**Program File name: 13.c**

```
#include<stdio.h>
#include<sys/stat.h>
#include<time.h>
int main(int argc, char *argv[])
{
    int i, j;
    struct stat a;
    for (i=1; i<argc; i++)
    {
        printf("%s : ", argv[i]);
        stat(argv[i], &a);
        if(S_ISDIR(a.st_mode))
        {
            printf("is a Directory file\n");
        }
        else
        {
            printf("is Regular file\n");
        }

        printf("*****File Properties*****\n");
        printf("Inode Number: %d\n", a.st_ino);
        printf("UID: %o\n", a.st_uid);
        printf("GID: %o\n", a.st_gid);
        printf("No of Links: %d\n", a.st_nlink);
        printf("Last Access time: %s", asctime(localtime(&a.st_atime)));
    }
}
```

```

printf("Permission flag:%o\n",a.st_mode%512);
printf("size in bytes:%d\n",a.st_size);
printf("Blocks Allocated:%d\n",a.st_blocks);
printf("Last modification time %s\n",ctime(&a.st_atime));
}
}

```

**Exercises:**

S.No.	Task
1	write a c program that simulates mkdir command using systemcalls
2	write a c program that simulates rmdir command using systemcalls

## EXPERIMENT NO: 7

**Aim:** Write a C program to list every file in directory, its inode number and file name

**Algorithm:**

- Step 1: Start
- Step 2: Read Directory name
- Step 3: open the directory
- Step 4: print file name and Inode number of each file in the directory
- Step 5: Stop

**Program file name:** inode.c

```
#include<fcntl.h>
#include<stdio.h>
#include<dirent.h>
#include<sys/stat.h>
int main(int argc,char*argv[])
{
    DIR *dirop;
    struct dirent *dired;
    if(argc!=2)
    {
        printf("Invalid number of arguments\n");
    }
    else if((dirop=opendir(argv[1]))==NULL)

        printf("Cannot open Directory\n");
    else
    {
        printf("%10s %s \n","Inode","File Name");
        while((dired=readdir(dirop))!=NULL)
            printf("%10d %s\n",dired->d_ino,dired->d_name);
        closedir(dirop);
    }
    return 0;
}
```

**Exercises:**

S.No.	Task
1	Write a c program to test whether the given file is seekable or not
2	Write a c program for requesting and releasing lock

## EXPERIMENT NO: 8 a)

**Aim:** Write a C program to create child process and allow parent process to display “parent” and the child to display “child” on the screen

**Algorithm:**

- Step 1: start
- Step2: call the fork() function to create a child process
  - fork function returns 2 values
- step 3: which returns 0 to child process
- step 4: which returns process id to the parent process
- step 5: stop

### Program file name: 16.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
    int pid,pid1,pid2;
    pid=fork();
    if(pid==-1)
    {
        printf("ERROR IN PROCESS CREATION \n");
        exit(0);
    }
    if(pid!=0)
    {
        pid1=getpid();
        printf("\n the parent process ID is %d", pid1);
    }
    else
    {
        pid2=getpid();
        printf("\n the child process ID is %d\n", pid2);
    }
}
```

Execution:

```
[root@dba ~]# cc -o 16 16.c
[root@dba ~]# ./16
```

```
the child process ID is 4485
the parent process ID is 4484
```

## EXPERIMENT NO: 8 b)

**Aim:** Write a C program to create zombie process

**Algorithm:** Step 1: call fork function to create a child process

Step 2: if fork() > 0

Then creation of Zombie

By applying sleep function for 10 seconds

Step 3: now terminate the child process

Step 4: exit status child process not reported to parent

Step 5: status any process which is zombie can known by

Applying ps(1) command

Step 6: stop

**Program file name:** 17.c

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

```
#include <stdlib.h>
```

```
int main()
```

```
{
```

```
int pid;
```

```
pid=fork();
```

```
if(pid == 0)
```

```
{ printf("I am child my pid is %d\n",getpid());
```

```
printf("My parent pid is:%d\n",getppid());
```

```
exit(0);
```

```
}
```

```
else
```

```
{ printf("I am parent, my pid is %d\n",getpid());
```

```
sleep(100);
```

```
exit(0);
```

```
}
```

```
}
```

**Execution:**

To see zombie process, after running the program, open a new terminal Give this command \$ps -el|grep a.out

First terminal

Compilation:

```
[root@dba ~]# cc 17.c
```

Executing binary

```
[root@dba ~]# ./a.out
```

```
I am child my pid is 4732
```

```
My parent pid is:4731
```

```
I am parent, my pid is 4731
```

Checking for zombie process. Z means zombie process

Second terminal

```
[root@dba ~]# ps -el|grep a.out
```

```
0 S  0 4731 4585 0 77  0 - 384 - pts/3  00:00:00 a.out
```

```
1 Z  0 4732 4731 0 77  0 - 0 exit pts/3  00:00:00 a.out <defunct>
```

**Exercises:**

S.No.	Task
1	Write a program to create zombie process and then call system functions to execute ps(1) command to verify process is zombie

## EXPERIMENT NO: 8 c)

**Aim:-Write a C program to illustrate how an orphan process is created**

**Algorithm:**

- Step 1: call the fork function to create the child process
- Step 2:if (pid==0)
  - Then print child id and parent id
  - else goto step 4
- Step 3:Then sleep(10)
  - Print child id and parent id
- Step 4: Print child id and parent id
- Step 5:which gives the information of orphan process
- Step 6:stop

**Program file name:18.c**

```
#include <stdio.h>
#include<stdlib.h>
int main()
{
    int pid;
    printf("I am the original process with PID %d and PPID %d\n",getpid(),getppid());
    pid=fork();
    if(pid == 0)
    {
        printf("I am child, my pid is %d ",getpid());
        printf("My Parent pid is:%d\n",getppid());
        sleep(10);
        printf("Now my pid is %d ",getpid());
        printf("My parent pid is:%d\n",getppid());
        exit(0);
    }
    else
    {
        sleep(10);
        printf("I am parent, my pid is %d\n",getpid());
        //printf("I am going to die\n");
    }
    printf("PID:%d terminates...\n",getpid());
}
```

**Execution:**

**Compilation :** [root@dba ~]# cc -o 18 18-1.c

**Executing Binary:**

```
[root@dba ~]# ./18
I am the original process with PID 5960 and PPID 5778
I am child, my pid is 5961 My Parent pid is:5960
I am parent, my pid is 5960
PID:5960 terminates...
[root@dba ~]# Now my pid is 5961 My parent pid is:1
```

**Exercises:**

S.No.	Task
1	Write a program to illustrate Vfork();
2	Write a program to illustrate fork();

## EXPERIMENT NO: 9 a)

**Aim:- Write a C program that illustrate communication between two unrelated process using named pipes**

### Algorithm for server :

step 1:Start  
step 2:Create a first named pipe by using mkfifo system call  
Pipe1=mkfifo(NP1,0666).  
step 3:if mkfifo returns -1 then  
print a message that error in creating the pipe.  
step 4:Create a second named pipe by using mkfifo system call  
Pipe2=mkfifo(NP2,0666).  
step 5:if mkfifo returns -1 then  
print a message that error in creating the pipe.  
step 6:Open the first pipe for reading by open system call by setting  
O\_RDONLY Fd=open(NP1,O\_RDONLY)  
step 7: Open the second pipe for writing by open system call by setting  
O\_WRONLY Fd=open(NP2,O\_WRONLY)  
step 8:read the data from the first pipe by using read system call  
numread=Read(fd,buf,MAX\_BUF\_SIZE) buf\*numread+='\0'  
step 9:print the data that we have read from pipe  
step 10:convert the data to the upper case.  
step 11:write the converted string back to second pipe by write(fd,buf, strlen(buf))  
step 12:stop.

### Algorithm for client :

Step 1:start  
Step 2:check whether the no of arguments specified were correct or not  
Step 3:if no of arguments are less then print error message  
Step 4:Open the first named pipe for writing by open system call by setting  
O\_WRONLY Fd=open(NP1,O\_WRONLY)  
Step 5: .Open the second named pipe for reading by open system call by setting  
O\_RDONLY Fd=open(NP2,O\_RDONLY)  
Step 6: write the data to the pipe by using write system call  
write(fd,argv[1],strlen(argv[1]))  
Step 7: read the data from the first pipe by using read system call  
numread=Read(fd,buf,MAX\_BUF\_SIZE) buf\*numread+='\0'  
Step 8: print the data that we have read from pipe  
Step 9:stop

**Program file name:named\_pipe.c**

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<string.h>
#include<fcntl.h>
void server(int,int);
void client(int,int);
int main()
{
int p1[2],p2[2],pid;
    pipe(p1);
    pipe(p2);
    pid=fork();
    if(pid==0)
    {
        close(p1[1]);
        close(p2[0]);
        server(p1[0],p2[1]);
        return 0;
    }
    close(p1[0]);
    close(p2[1]);
    client(p1[1],p2[0]);
    wait();
return 0;
}

void client(int wfd,int rfd)
{
int i,j,n;
char fname[2000];
char buff[2000];
printf("ENTER THE FILE NAME :");
scanf("%s",fname);
printf("CLIENT SENDING THE REQUEST .....PLEASE WAIT\n");

sleep(10);
write(wfd,fname,2000);
n=read(rfd,buff,2000);
buff[n]='\0';
printf("THE RESULTS OF CLIENTS ARE.....\n");
write(1,buff,n);
}

void server(int rfd,int wfd)
{
int i,j,n; char fname[2000];
char buff[2000];
```



```

n=read(rfd,fname,2000);
fname[n]='\0';
int fd=open(fname,O_RDONLY);
sleep(10);
if(fd<0)
    write(wfd,"can't open",9);
else
    n=read(fd,buff,2000);
write(wfd,buff,n);
}

```

**Exercises:**

S.No.	Task
1	Write a program to demonstrate the function of a pipe
2	Write a program to demonstrate the pipe function using dup() system call

## EXPERIMENT NO: 9 b)

**Aim:-Write a C program that receives a message from message queue and display them**

### Algorithm:

Step 1:Start

Step 2:Declare a message queue structure

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

Mtype = 0 Retrieve the next message on the queue, regardless of its mtype.

Positive Get the next message with an mtype equal to the specified msgtyp.

Negative Retrieve the first message on the queue whose mtype field is less than or equal to the absolute value of the msgtyp argument.

Usually mtype is set to 1

mtext is the data this will be added to the queue.

Step 3: Get the message queue id for the "name" 1234, which was created by the server  
key = 1234

Step 4 : if ((msqid = msgget(key, 0666) < 0) Then print error

The msgget() function shall return the message queue identifier associated with the argument key.

Step 5: Receive message from message queue by using msgrcv function

```
int msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg);
```

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

```
(msgrcv(msqid, &rbuf, MSGSZ, 1, 0)
```

msqid: message queue id

&rbuf: pointer to user defined structure MSGSZ: message size

Message type: 1

Message flag: The msgflg argument is a bit mask constructed by ORing together zero or more of the following flags: IPC\_NOWAIT or MSG\_EXCEPT or MSG\_NOERROR

Step 6: if msgrcv < 0 return error

Step 7: otherwise print message sent is sbuf.mext

Step 8: stop

### Exercises:

S.No.	Task
1	Write a program to demonstrate a single process create a message queue and sends itself a "welcome" message via the queue
2	Write a program to demonstrate how we can print the status information about the queue

### EXPERIMENT NO: 10 a)

**Aim:-Write a C program to allow cooperating process to lock a resource for exclusive use using, a) Semaphore**

```
#include<stdio.h>
#include<stdlib.h>
#include<error.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>
int main(void)
{
    key_t key;
    int semid;
    union semun arg;
    if((key==ftok("sem demo.c","j"))== -1)
    {
        perror("ftok");
        exit(1);
    }
    if(semid=semget(key,1,0666|IPC_CREAT))== -1)
    {
        perror("semget");
        exit(1);
    }
    arg.val=1;
    if(semctl(semid,0,SETVAL,arg)== -1)
    {
        perror("smctl");
        exit(1);
    }
    return 0;
}
```

#### Exercises:

S.No.	Task
1	Write a program using the simpler semaphore operation
2	Write a program to create a semaphore

## EXPERIMENT NO: 10 b)

**Aim:-**Write a C program that illustrate the suspending and resuming process using signal

### Algorithm:

- Step 1: call the signal function to generate the signal
- Step 2: execution of process will be started
- Step 3: call alarm function to suspend the execution of current process
- Step 4: then it will execute the signal function
- Step 5: again the process will be resumed
- Step 6: stop

### Program

```
#include<stdio.h>
int main()
{
    int n;
    if(signal(SIGALRM,sig_alarm)==SIG_ERR)
        printf(,Signal error');
    alarm(5);
    for(n=0;n<=15;n++)
        printf(,from for loop n=%d',n);
    printf(,main program terminated');
}

void sig_alarm(int signo)
{
    printf(,from sigalarm function');
}
```

### Exercises:

S.No.	Task
1	Write a program using kill and rise functions
2	Write a program using abort()

## EXPERIMENT NO: 10 c)

**Aim:-Write a C program that implements producer –consumer system with two processes using semaphores**

### Algorithm for producer :

- step 1:Start
- step 2:Create a named pipe by using mkfifo system call Pipe1=mkfifo(NP1,0666)
- step 3:if mkfifo returns -1 then print a message that error in creating the pipe
- step 4:Open the pipe for reading by open system call by setting O\_RDONLY Fd=open(NP1,O-  
RDONLY)
- step 5:read the data from the pipe by using read system call  
numread=Read(fd,buf,MAX-BUF-SIZE)
- step 6:print the data that we have read from pipe
- step 7:convert the data to the upper case.
- step 8:print the converted data
- step 9:stop.

### Algorithm for consumer:

- Step 1:start
- step 2:check whether the no of arguments specified were correct or not
- step3:if no of arguments are less then print error message
- step 4:Open the pipe for writing by open system call by setting O\_WRONLY  
Fd= open (NP1, O\_WRONLY )
- step 5: write the data to the pipe by using write system call write(fd,argv[1],strlen(argv[1]))
- step 6:stop

### Consumer:

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#define MAXSIZE 10
#define FIFO_NAME "myfifo"
int main()
{
    int fifoid; int fd, n; char *r;
    system("clear");
    r=(char *)malloc(sizeof(char)*MAXSIZE); int open_mode=O_RDONLY;
    if( (fd=open(FIFO_NAME, open_mode)) < 0 )
    {
        printf("\nError: Named pipe cannot be opened\n"); exit(0);
    }
    while(1)
    {
        n=read(fd, r, MAXSIZE); if(n > 0)
        printf("\nConsumer read: %s", r);
    }
} /*main close*/
```

### Producer program:

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#define MAXSIZE 10

#define FIFO_NAME "myfifo"
```

```

int main()
{
int fifoid; int fd, n; char *w;
int open_mode;
    system("clear");
    w=(char *)malloc(sizeof(char)*MAXSIZE);
    open_mode=O_WRONLY;
    fifoid=mkfifo(FIFO_NAME, 0755);
    if(fifoid==-1)
    {
        printf("\nError: Named pipe cannot be Created\n"); exit(0);
    }
    if( (fd=open(FIFO_NAME, open_mode)) < 0 )
    {
        printf("\nError: Named pipe cannot be opened\n");
        exit(0);
    }
    while(1)
    {
        printf("\nProducer :"); fflush(stdin);
        read(0, w, MAXSIZE);
        n=write(fd, w, MAXSIZE);
        if(n > 0)
            printf("\nProducer sent: %s", w);

    }

} /*main close*/

```

Output:

```

$ cc -o producer producer.c           #first window
$cc -o consumer consumer.c           # second window
$ ./producer      #first window
$./consumer      # second window
Producer:
Producer sent:  hai           #first window
Consumer read: hai           # second window
Producer sent:  good morning  #first window
Consumer read: good morning # second window
Producer sent:  welcome       #first window
Consumer read: welcome        # second window

```

## EXPERIMENT NO: 11

**Aim:-Write client server programs using c for interaction between server and clientprocess using Unix Domain sockets**

### Algorithm:-

Sample UNIX server

Step 1:define NAME "socket"

Step 2: sock = socket(AF\_UNIX, SOCK\_STREAM, 0);

Step 3:if (sock < 0) perror("opening stream socket");exit(1);

step4: server.sun\_family = AF\_UNIX;

strcpy(server.sun\_path, NAME);

if (bind(sock, (struct sockaddr \*) &server, sizeof(struct sockaddr\_un)))

{  
perror("binding stream socket"); exit(1);  
}

step 5: print ("Socket has name %s\n", server.sun\_path);

listen(sock, 5);

step 6: for (;;)

{  
msgsock = accept(sock, 0, 0);  
if (msgsock == -1)  
perror("accept");  
else  
do { bzero(buf, sizeof(buf));  
if ((rval = read(msgsock, buf, 1024)) < 0)

perror("reading stream message");

else if (rval == 0)

else print ("-->%s\n", buf);

} while (rval > 0);

close(msgsock);

}

close(sock);

unlink(NAME);

}

Step 7:stop

### Programs:

#### Server.c

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

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

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

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

```
#include <unistd.h>
```

```
#include <string.h>
```

```

int connection_handler(int connection_fd)
{
    int nbytes;
    char buffer[256];
    nbytes = read(connection_fd, buffer, 256);
    buffer[nbytes] = 0;
    printf("MESSAGE FROM CLIENT: %s\n", buffer);
    nbytes = snprintf(buffer, 256, "hello from the server");
    write(connection_fd, buffer, nbytes);
    close(connection_fd);
    return 0;
}

int main(void)
{
    struct sockaddr_un address;
    int socket_fd, connection_fd;
    socklen_t address_length;
    pid_t child;
    socket_fd = socket(PF_UNIX, SOCK_STREAM, 0);
    if(socket_fd < 0)
    {
        printf("socket() failed\n");
        return 1;
    }

    unlink("./demo_socket");

    /* start with a clean address structure */
    memset(&address, 0, sizeof(struct sockaddr_un));

    address.sun_family = AF_UNIX;
    snprintf(address.sun_path, UNIX_PATH_MAX, "./demo_socket");

    if(bind(socket_fd,
            (struct sockaddr *) &address,
            sizeof(struct sockaddr_un)) != 0)
    {
        printf("bind() failed\n");
        return 1;
    }
}

```



```

if(listen(socket_fd, 5) != 0)
{
    printf("listen() failed\n");
    return 1;
}

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

```

### **Client.c**

```

#include <stdio.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <string.h>

int main(void)
{
    struct sockaddr_un address;
    int socket_fd, nbytes;
    char buffer[256];

    socket_fd = socket(PF_UNIX, SOCK_STREAM, 0);
    if(socket_fd < 0)
    {
        printf("socket() failed\n");
        return 1;
    }

```

```

/* start with a clean address structure */
memset(&address, 0, sizeof(struct sockaddr_un));

address.sun_family = AF_UNIX;
snprintf(address.sun_path, UNIX_PATH_MAX, "./demo_socket");

if(connect(socket_fd,
          (struct sockaddr *) &address,
          sizeof(struct sockaddr_un)) != 0)
{
    printf("connect() failed\n");
    return 1;
}
nbytes = snprintf(buffer, 256, "hello from a client");
write(socket_fd, buffer, nbytes);

nbytes = read(socket_fd, buffer, 256);
buffer[nbytes] = 0;

printf("MESSAGE FROM SERVER: %s\n", buffer);

close(socket_fd);
return 0;
}

```

### Exercises:

S.No.	Task
1	Write a program to demonstrate getting and setting the socket optionsthrough socket related system call
2	Write a program to demonstrate bind system call.

## EXPERIMENT NO: 12

**Aim:-**Write a C program that illustrates two processes communicating using Shared memory

**Algorithm:-**

step1.Start

step 2.Include header files required for the program are

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
```

step 3.Declare the variable which are required as

```
pid_t pid
int *shared /* pointer to the shm */
int shmid
```

step 4.Use shmget function to create shared memory

```
#include <sys/shm.h>
int shmget(key_t key, size_t size, int shmflg)
```

The shmget() function shall return the shared memory identifier associated with key The argument key is equal to IPC\_PRIVATE. so that the operating system selects the next available key for a newly created shared block of memory. Size represents size of shared memory block Shmflg shared memory permissions which are represented by octalinteger

```
shmid = shmget (IPC_PRIVATE, sizeof(int), IPC_CREAT | 0666);
print the shared memory id
```

step 5.if fork()==0 Then

```
begin
    shared = shmat(shmid, (void *) 0, 0)
    print the shared variable(shared) *shared=2
    print *shared sleep(2)
    print *shared
end
```

step 6.else

```
begin
    shared = shmat(shmid, (void *) 0, 0)
    print the shared variable(shared)
    print *shared sleep(1) *shared=30
    printf("Parent value=%d\n", *shared);
    sleep(5)
    shmctl(shmid, IPC_RMID, 0)
```

end

step 7.stop.

**Program:**

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
```

```

int main(void) {
    pid_t pid;
    int *shared; /* pointer to the shm */ int shmid;
    shmid = shmget(IPC_PRIVATE, sizeof(int), IPC_CREAT | 0666); printf("Shared Memory
    ID=%u",shmid);
    if (fork() == 0) { /* Child */

        /* Attach to shared memory and print the pointer */ shared = shmat(shmid, (void *) 0, 0);
        printf("Child pointer %u\n", shared); *shared=1;
        printf("Child value=%d\n", *shared); sleep(2);
        printf("Child value=%d\n", *shared); } else { /* Parent */
        /* Attach to shared memory and print the pointer */ shared = shmat(shmid, (void *) 0, 0);
        printf("Parent pointer %u\n", shared); printf("Parent value=%d\n", *shared); sleep(1);
        *shared=42;
        printf("Parent value=%d\n", *shared); sleep(5);
        shmctl(shmid, IPC_RMID, 0);
    }

}

```

```

sampath@localhost ipc]$cc shared_mem.c
[sampath@localhost ipc]$ ./a.out
Shared Memory ID=65537Child pointer 3086680064 Child value=1
Shared Memory ID=65537Parent pointer 3086680064 Parent value=1
Parent value=42 Child value=42

```

### Viva questions

- 1.define shared memory
- 2.what are file locking functions.
- 3.what are shared memory systemcalls.
- 4.define internet domain sockets
- 5.Difference between internet and unix domain sockets.

### Exercises:

S.No.	Task
1	Write a program to demonstrate communication of two different process via shared memory
2	Write a program to demonstrate that the shared memory created will be available even after the process which created is exited.