

---

**Instruction Hours / week: L: 4 T: 0 P: 0    Marks: Internal: 40    External: 60    Total: 100**  
**End Semester Exam: 3 Hours**

**Course Objective**

- To understand fundamental operating system abstractions such as processes, threads, files, semaphores, IPC abstractions, shared memory regions, etc.
- To understand how the operating system abstractions can be used in the development of application programs, or to build higher level abstractions
- To Understand how the operating system abstractions can be implemented
- To Understand the principles of concurrency and synchronization, and apply them to write correct concurrent programs/software
- To understand basic resource management techniques (scheduling or time management, space management) and principles and how they can be implemented. These also include issues of performance and fairness objectives, avoiding deadlocks, as well as security and protection.

**Course Outcomes(COs)**

1. This course teaches the student the concepts and principles that underlie modern operating systems, and a practice component to relate theoretical principles with operating system implementation.
2. Learn about processes and processor management, concurrency and synchronization, memory management schemes, file system and secondary storage management, security and protection, etc.

**Unit I - INTRODUCTION**

What is Linux/Unix Operating systems, Difference between linux/unix and other operating systems , Features and Architecture, Various Distributions available in the market, Installation, Booting and shutdown process

**Unit II - SYSTEM PROCESSES (AN OVERVIEW)**

External and internal commands, Creation of partitions in OS, Processes and its creation phases – Fork, Exec, wait

**Unit III- USER MANAGEMENT AND THE FILE SYSTEM**

Types of Users, Creating users, Granting rights User management commands, File quota and various file systems available, File System Management and Layout, File permissions, Login process, Managing Disk Quotas, Links (hard links, symbolic links)

**Unit IV - SHELL INTRODUCTION AND SHELL SCRIPTING**

What is shell and various type of shell, various editors present in Linux Different modes of operation in vi editor, What is shell script, Writing and executing the shell script, Shell variable (user defined and system variables)

**Unit V- SYSTEM CALLS, USING SYSTEM CALLS**

Pipes and Filters, Decision making in Shell Scripts (If else, switch), Loops in shell Functions, Utility programs (cut, paste, join, tr, uniq utilities), Pattern matching utility (grep)

## **SUGGESTED READINGS**

1. Sumitabha, Das.(2006). Unix Concepts And Applications. New Delhi: Tata McGraw-Hill Education.
2. Michael Jang. (2011). RHCSA/ RHCE Red Hat Linux Certification: Exams (Ex200 & Ex300). Certification Press.
3. Nemeth Synder., & Hein.(2010). Linux Administration Handbook (2nd ed.). Pearson Education.
4. Richard Stevens. W. Bill Fenner., & Andrew, M. Rudoff. (2014). Unix Network Programming. The sockets Networking API.Vol. 1. (3rd ed.).

## **WEBSITE**

1. [www.sethi.org/tutorials/references\\_unix.html](http://www.sethi.org/tutorials/references_unix.html)
2. [www.penguintutor.com/linux/basic-shell-programming-reference](http://www.penguintutor.com/linux/basic-shell-programming-reference)
3. <https://swcarpentry.github.io/shell-novice/reference/>



**KarpagamAcademy of Higher Education**  
*(Established Under Section 3 of UGC Act 1956)*  
 Eachanari Post, Coimbatore – 641 021. INDIA  
 Phone : 0422-2611146, 2611082 Fax No : 0422 -2611043

## **17CSU603B UNIX/LINUX PROGRAMMING**

### **LECTURE PLAN**

<b>S.No</b>	<b>Lecturer Duration(Hrs)</b>	<b>Topics to be Covered</b>	<b>Support Materials</b>
<b>UNIT I</b>			
1	1	What is Linux/Unix Operating Systems	R1:12-15,W1
2	1	Difference between linux/unix and other operating systems	R1:16-20
3	1	Features and Architecture	R2:15-22
4	1	Various Distributions available in the market	R2:23-27
		Installation, Booting and shutdown process	R1:25-30
5	1	Recapitulation and discussion of Important Questions	
<b>Total No. of Hours planned for Unit-I : 5 Hours</b>			

<b>S.No</b>	<b>Lecturer Duration(Hrs)</b>	<b>Topics to be Covered</b>	<b>Support Materials</b>
-------------	-----------------------------------	-----------------------------	--------------------------

**UNIT II**

1	1	System processes (an overview)	R3:53-58
2	1	External and internal commands	R3:65-69
3	1	Creation of partitions in OS	R3:70-77,W2
4	1	Processes and its creation phases, Fork, Exec, wait	R3:78-86,W3
5	1	Recapitulation and discussion of Important Questions	
<b>Total No. of Hours planned for Unit II : 5 Hours</b>			

S.No	Lecturer Duration(Hrs)	Topics to be Covered	Support Materials
<b>UNIT III</b>			
1	1	Types of Users, Creating users	R4:63-71
2	1	Granting rights User management commands	R4:72-77
3	1	File quota and various file systems available	R4:78-87
4	1	File System Management and Layout, File permissions	R4:88-96,J1
5	1	Login process, Managing Disk Quotas, Links (hard links, symbolic links)	R4:103-112
6	1	Recapitulation and discussion of Important Questions	
<b>Total No. of Hours planned for Unit III : 6 Hours</b>			

### LECTURE PLAN

S.No	Lecturer Duration(Hrs)	Topics to be Covered	Support Materials
------	---------------------------	----------------------	-------------------

#### UNIT IV

1	1	<b>Shell introduction and Shell Scripting</b>	R4:129-138
2	1	What is shell and various type of shell	R4:139-143, W4
3	1	Various editors present in Linux Different modes of operation in vi editor	R4:143-148
4	1	What is shell script, Writing and executing the shell script	R4:149-157
5	1	Shell variable (user defined and system variables)	R4:158-166
6	1	Recapitulation and Discussion of important Questions	
<b>Total No. of Hours planned for Unit IV : 6 Hours</b>			

### LECTURE PLAN

S.No	Lecturer Duration(Hrs)	Topics to be Covered	Support Materials
<b>UNIT V</b>			
1	1	<b>System calls, Using system calls</b>	R4:167-173
2	1	Pipes and Filters, Decision making in Shell Scripts (If else, switch),	W5,R4:174-177
3	1	Loops in shell, Functions	R4:178-186
4	1	Utility programs (cut, paste, join, tr, uniq utilities), Pattern matching utility (grep)	R4:187-195
5	1	Recapitulation and Discussion of important Questions	
6	1	Discussion of Previous ESE papers	
7	1	Discussion of Previous ESE papers	
8	1	Discussion of Previous ESE papers	
		<b>Total No. of Hours planned for Unit V : 8 Hours</b>	
	<b>TOTAL PLANNED HOURS : 30</b>		

### REFERENCE BOOKS:

R1: Michael Jang, (2011). *RHCSA/ RHCE Red Hat Linux Certification: Exams (Ex200 & Ex300)*, Certification Press.

R2:Nemeth Synder & Hein,(2010). *Linux Administration Handbook*, (2<sup>nd</sup> ed. ) Pearson Education.

R3: Sumitabha, Das, (2006). *Unix Concepts And Applications*, Tata McGraw-Hill Education.

R4: Richard Stevens,W., Bill Fenner, Andrew M. Rudoff, (2014). *Unix Network Programming, The sockets Networking*, Vol. 1, 3<sup>rd</sup> ed. API.

## **WEBSITES**

W1:<http://en.wikipedia.org/wiki/Linux>

W2:[http://en.wikibooks.org/wiki/partitions\\_unix](http://en.wikibooks.org/wiki/partitions_unix)

W3:[www.tutorialspoint.com/fork](http://www.tutorialspoint.com/fork)

W4: [www.guru99.com/introduction\\_to\\_shell\\_scripting.html](http://www.guru99.com/introduction_to_shell_scripting.html)

W5: [www.tutorialspoint.com/unix/unix\\_decision\\_making.htm](http://www.tutorialspoint.com/unix/unix_decision_making.htm)

## **JOURNALS:**

J1: "Role of File System in Operating System", Int. Journal of Computer Science and innovation, Vol 3, 2016.



## UNIT-I

### SYLLABUS

What is Linux/Unix Operating systems, Difference between linux/unix and other operating systems , Features and Architecture, Various Distributions available in the market, Installation, Booting and shutdown process.

#### **Unix Operating Systems:**

The UNIX OS was born in the late 1960s. AT&T Bell Labs released an operating system called Unix written in C, which allows quicker modification, acceptance, and portability. The Unix OS works on CLI (Command Line Interface), but recently, there have been developments for GUI on Unix systems. Unix is an OS which is popular in companies, universities big enterprises, etc.

#### **What is LINUX?**

Linux is an operating system built by Linus Torvalds at the University of Helsinki in 1991. The name "Linux" comes from the Linux kernel. It is the software on a computer which enables applications and the users to access the devices on the computer to perform some specific function.

The Linux OS relays instructions from an application from the computer's processor and sends the results back to the application via the Linux OS. It can be installed on a different type of computers mobile phones, tablets video game consoles, etc.

The development of Linux is one of the most prominent examples of free and open source software collaboration. Today many companies and similar numbers of individuals have released their own version of OS based on the Linux Kernel.

#### **Features of Unix OS**

Multi-user, multitasking operating system

It can be used as the master control program in workstations and servers.

Hundreds of commercial applications are available

In its heydays, UNIX was rapidly adopted and became the standard OS in universities.

#### **Features of Linux Operating System**

Support multitasking

Programs consist of one or more processes, and each process have one or more threads

It can easily co-exists along with other Operating systems.

It can run multiple user programs

Individual accounts are protected because of appropriate authorization

Linux is a replica of UNIX but does not use its code.

### **Difference between Unix and Linux**

<b>Basis of Difference</b>	<b>Linux</b>	<b>Unix</b>
<b>Cost</b>	Linux is freely distributed, downloaded through magazines, Books, website, etc. There are paid versions also available for Linux.	Different flavors of Unix have different pricing depending upon the type of vendor.
<b>Development</b>	Linux is Open Source, and thousands of programmer collaborate online and contribute to its development.	Unix systems have different versions. These versions are primarily developed by AT&T as well as other commercial vendors.
<b>User</b>	Everyone. From home users to developers and computer enthusiasts alike.	The UNIX can be used in internet servers, workstations, and PCs.
<b>Text made interface</b>	BASH is the Linux default shell. It offers support for multiple command interpreters.	Originally made to work in Bourne Shell. However, it is now compatible with many others software.
<b>GUI</b>	Linux provides two GUIs,viz., KDE and Gnome. Though there are many alternatives such as Mate, LXDE, Xfce, etc.	Common Desktop Environment and also has Gnome.
<b>Viruses</b>	Linux has had about 60-100 viruses listed to date which are currently not spreading.	There are between 80 to 120 viruses reported till date in Unix.
<b>Threat detection</b>	Threat detection and solution is very fast because Linux is mainly community driven. So, if any Linux user posts any kind of threat, a team of qualified developers starts working to resolve this threat.	Unix users require longer wait time, to get the proper bug fixing patch.

Basis of Difference	Linux	Unix
Architectures	Initially developed for Intel's x86 hardware processors. It is available for over twenty different types of CPU which also includes an ARM.	It is available on PA-RISC and Itanium machines.
Usage	Linux OS can be installed on various types of devices like mobile, tablet computers.	The UNIX operating system is used for internet servers, workstations & PCs.
Best feature	Kernel update without reboot	Feta ZFS - next generation filesystem DTrace - dynamic Kernel Tracing
Versions	Different Versions of Linux are Redhat, Ubuntu, OpenSuse, Solaris, etc.	Different Versions of Unix are HP-UX, AIS, BSD, etc.
Supported file type	The Filesystems supported by file type like xfs, nfs, cramfs, ext 1 to 4, ufs, devpts, NTFS.	The Filesystems supported by file types are zfs, hfs, GPS, xfs, vxfs.
Portability	Linux is portable and is booted from a USB Stick	Unix is not portable
Source Code	The source is available to the general public	The source code is not available to anyone.

### Differences Between Linux and Other Operating Systems

It's important to understand the differences between Linux and other operating systems, such as Windows 95/98, Windows NT, OS/2, and other implementations of Unix for the personal computer. First of all, it should be made clear that Linux will coexist happily with other operating systems on the same machine: that is, you can run Windows NT and OS/2 along with Linux on the same system without problems. There are even ways to interact between the various operating systems, as you'll see.

### Why Use Linux?

Why use Linux instead of a commercial operating system? We could give you a thousand reasons. One of the most important, however, is that Linux is an excellent choice for personal Unix computing. If you're a Unix software developer, why use Windows at home? Linux will allow you to develop and test Unix software on your PC, including database and X applications. If you're a student, chances are that your university computing system runs Unix. With Linux, you can run your own Unix system and tailor it to your own needs.

Installing and running Linux is also an excellent way to learn Unix if you don't have access to other Unix machines.

But let's not lose perspective. Linux isn't just for personal Unix users. It's robust and complete enough to handle large tasks, as well as distributed computing needs. Many businesses are moving to Linux in lieu of other Unix-based workstation environments. Linux has an excellent price-performance ratio, is one of the most stable and powerful operating systems available, and because of its Open Source nature, is completely customizable for your needs. Universities are finding Linux to be perfect for teaching courses in operating systems design. Larger commercial software vendors are starting to realize the opportunities a free operating system can provide.

#### Linux Versus Windows 95 and 98

It's not uncommon to run both Linux and Windows 95/98 on the same system. Many Linux users rely on Windows for applications such as word processing and productivity tools. While Linux provides its own analogs for these applications (for example, TeX), and commercial software support for Linux is increasing, there are various reasons why a particular user would want to run Windows as well as Linux. If your entire dissertation is written using Microsoft Word, you may not be able to easily convert it to TeX or some other format (although the Star Office suite for Linux can probably do the trick). There are many commercial applications for Windows that aren't available for Linux, and there's no reason why you can't use both.

As you might know, Windows 95 and 98 do not fully utilize the functionality of the x86 processor. On the other hand, Linux runs completely in the processor's protected mode and exploits all of the features of the machine, including multiple processors.

We could debate the pros and cons of Windows and Linux for pages on end. However, suffice it to say that Linux and Windows are completely different entities. Windows is inexpensive (compared to other commercial operating systems) and has a strong foothold in the PC computing world. No other operating system for the PC has reached the level of popularity of Windows, largely because the cost of these other operating systems is unapproachable for most personal computer users. Very few PC users can imagine spending a thousand dollars or more on the operating system alone. Linux, however, is free, and you finally have the chance to decide.

We will allow you to make your own judgments of Linux and Windows based on your expectations and needs. Linux is not for everybody. But if you have always wanted to run a

complete Unix system at home, without the high cost of other Unix implementations for the PC, Linux may be what you're looking for.

There are tools available to allow you to interact between Linux and Windows. For example, it's easy to access Windows files from Linux. Development is proceeding on the Wine Windows emulator, which allows you to run many popular applications.

### Linux Versus Windows NT

A number of other advanced operating systems are on the rise in the PC world. Specifically, Microsoft's Windows NT is becoming very popular for server computing.

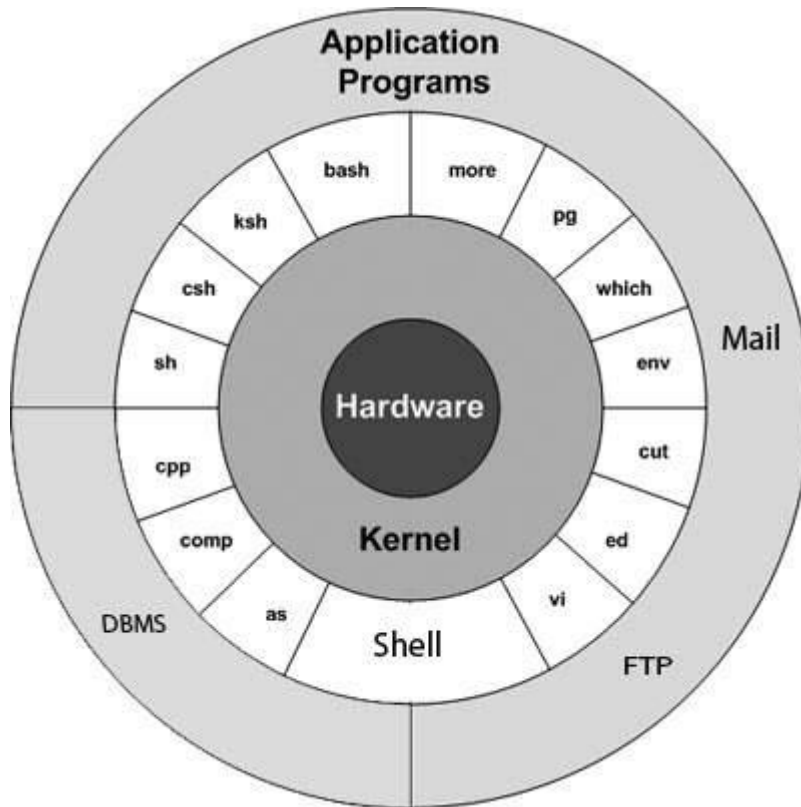
Windows NT, like Linux, is a full multitasking operating system, supporting multiprocessor machines, several CPU architectures, virtual memory, networking, security, and so on. However, the real difference between Linux and Windows NT is that Linux is a version of Unix and hence benefits from the contributions of the Unix community at large.

There are many implementations of Unix from many vendors. There is a large push in the Unix community for standardization in the form of open systems, but no single corporation controls this design. Hence, any vendor (or, as it turns out, any hacker) may implement these standards in an implementation of Unix.

Windows NT, on the other hand, is a proprietary system. The interface and design are controlled by a single corporation, Microsoft, and only that corporation may implement the design. (Don't expect to see a free version of Windows NT anytime in the near future.) In one sense, this kind of organization is beneficial: it sets a strict standard for the programming and user interface unlike that found even in the open systems community. NT is NT wherever you go.

It seems likely that in the coming years, Linux and Windows NT will be battling it out for their share of the server computing market. Windows NT has behind it the full force of the Microsoft marketing machine, while Linux has a community of thousands of developers helping to advance the system through the Open Source model. So far, benchmarks of Linux versus Windows NT have demonstrated that each system has its strengths and weaknesses; however, Linux wins hands-down in a number of areas, most notably networking performance. Linux is also much smaller than Windows NT, has a much better price-performance ratio, and is generally seen as more stable. (While Windows NT is known to crash quite often, Linux machines run continuously for months.) It might seem amazing that

“little” Linux gives Microsoft serious competition, but it’s not surprising when you realize how effective the Open Source development process really is.



The main concept that unites all the versions of Unix is the following four basics –

**Kernel** – The kernel is the heart of the operating system. It interacts with the hardware and most of the tasks like memory management, task scheduling and file management.

**Shell** – The shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want. The shell uses standard syntax for all commands. C Shell, Bourne Shell and Korn Shell are the most famous shells which are available with most of the Unix variants.

**Commands and Utilities** – There are various commands and utilities which you can make use of in your day to day activities. cp, mv, cat and grep, etc. are few examples of commands and utilities. There are over 250 standard commands plus numerous others provided through 3rd party software. All the commands come along with various options.

Files and Directories – All the data of Unix is organized into files. All files are then organized into directories. These directories are further organized into a tree-like structure called the filesystem.

Utility software usually focuses on how the computer infrastructure (including the computer hardware, operating system, application software and data storage) operates. Due to this focus, utilities are often rather technical and targeted at people with an advanced level of computer knowledge - in contrast to application software, which allows users to do things like creating text documents, playing video games, listening to music or viewing websites.

### vi Editor in UNIX

The default editor that comes with the UNIX operating system is called vi (visual editor). Using vi editor, we can edit an existing file or create a new file from scratch. we can also use this editor to just read a text file.

#### Syntax:

vi filename

ed is a line-oriented text editor. It is used to create, display, modify and otherwise manipulate text files.

Editing is done in two distinct modes: command and input. When first invoked, ed is in command mode. In this mode commands are read from the standard input and executed to manipulate the contents of the editor buffer. A typical command might look like:

,s/old/new/g

which replaces all occurrences of the string old with new.

When an input command, such as 'a' (append), 'i' (insert) or 'c' (change), is given, ed enters input mode. This is the primary means of adding text to a file.

#### Cut:

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.

### Various Distributions of Unix/Linux

Unix is not a single operating system. It offers many modern variants also referred to as flavors, types, distributions, or implementations, which branch from its origins in early 1970s mainframe computing. Although based on a core set of Unix commands, different



distributions have their own unique commands and features and are designed to work with different types of hardware.

Contemporary Unix implementations differ in whether they're open-source (i.e., free to download, use, or modify) or closed source (i.e., proprietary binary files not subject to user modification).

- **Minix** — a Unix-like open-source project, rarely used by home users.
- **Linux** — an open-source initiative to bring a Unix-like environment to both the desktop and server space. Linux is popular with home computer enthusiasts.
- **Mac OS X** — Mac operating system from Apple Computer Inc. having a Unix core; the latest version is the Mac OS X Panther
- **BSDs** (FreeBSD, DragonflyBSD, NetBSD, OpenBSD) — a branch from the earliest Unix specs, following the design principles of the Berkeley Software Distribution.
- **AIX** — a series of Unix-based operating environments developed by IBM for its servers.
- **IRIX** – the proprietary version of Unix from Silicon Graphics Inc.; the latest release of which is IRIX 6.5
- **Solaris** — a proprietary server operating system based on Unix and developed by Sun Microsystems.
- **OpenSolaris** — an open-source variant of Solaris.
- **HP-UX** — a series of Unix-based operating environments developed by HP for its servers.
- **Tru64 UNIX** – the Unix operating environment for HP AlphaServer systems; Tru64 UNIX v5.1B-1 is the latest version
- **OpenServer** — based on FreeBSD and is a closed source operating system. It is now owned by XinuOS. Previously known as SCO UNIX, it was developed by Santa Cruz Operation. SCO acquired the rights to the UnixWare operating system, portions of which became part of OpenServer.

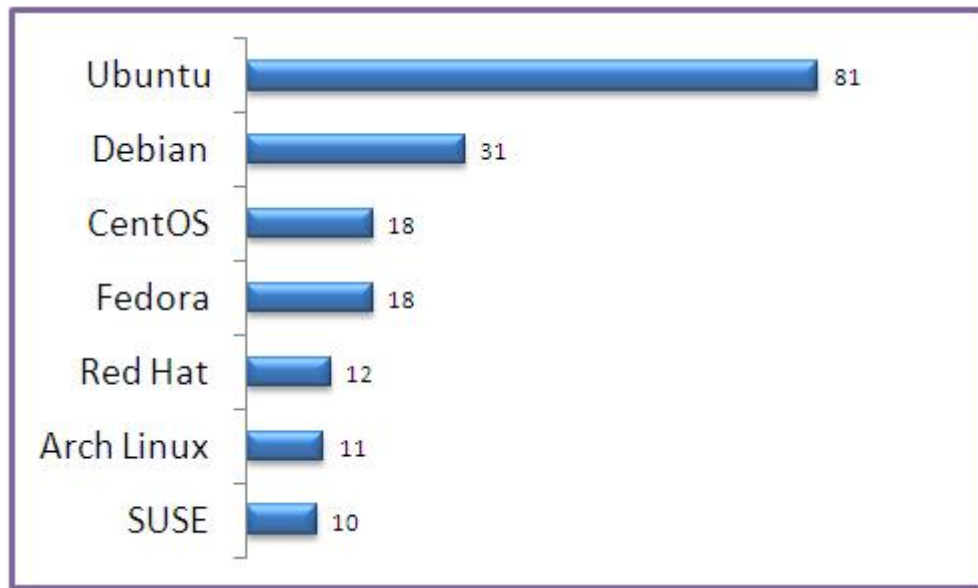
No one knows exactly how many Unix flavors are there, but it is safe to say that if including all those that are obscure and obsolete, the number of Unix flavors is at least in the hundreds.

Common Consumer Distributions


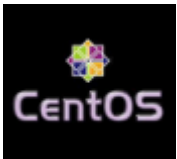







Over the years, different Linux flavors have enjoyed more or less popularity, but several stand out as being among the most commonly deployed on desktop computers. Some of the most commonly accessed distributions include:

- **Mint** — a version of Ubuntu with additional software drivers and minor customizations.
- **Debian** — a project that bills itself as a "universal operating system" and enjoys significant market share and a robust base of applications.
- **Manjaro** — based on the Arch Linux project and supports extensive configurability.
- **Ubuntu** — a significant player in the Linux market. Ubuntu's goal is to offer an easy-to-use distribution that's beautifully designed and accessible despite language and disability barriers.
- **Antergos** — also based on the Arch Linux project. This distribution offers its own custom installer program.
- **OpenSUSE** — a long-running German distribution that's the community version of the SUSE Linux commercial distribution.
- **Fedora** — a community project based on Red Hat Linux (an operating system that was discontinued in 2004).
- **Solus** — a built-from-scratch distribution from Ireland with a custom desktop environment called "Budgie" that looks like the old GNOME 2 desktop (GNOME is the default desktop environment on many major Linux distributions).
- **Zorin** — a distribution intended to mimic the look-and-feel of Windows to help new Linux users transition away from Microsoft's operating system.
- **Elementary** — based on Ubuntu and uses a custom desktop environment called Parthenon that resembles, in some ways, Mac OS.



#### Favorite Linux Distribution Voting Results

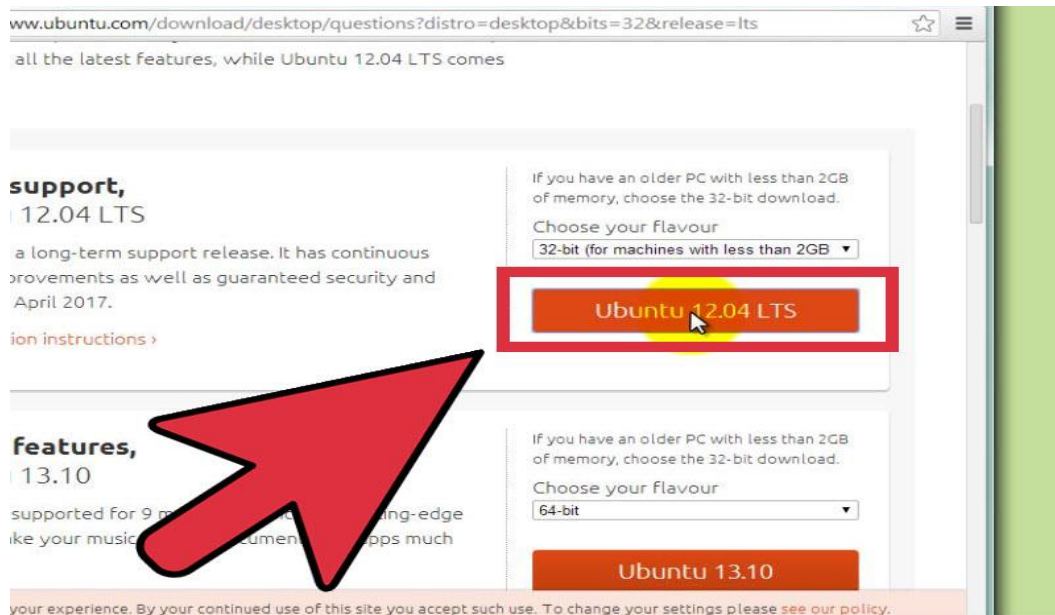
Linux Distribution	Name	Description
	Arch	This Linux Distro is popular amongst Developers. It is an independently developed system. It is designed for users who go for a do-it-yourself approach.
	CentOS	It is one of the most used Linux Distribution for enterprise and web servers. It is a free enterprise class Operating system and is based heavily on Red Hat enterprise Distro.
	Debian	Debian is a stable and popular non-commercial Linux distribution. It is widely used as a desktop Linux Distro and is user-oriented. It strictly acts within the Linux protocols.
	Fedora	Another Linux kernel based Distro, Fedora is supported by the Fedora project, an endeavor by Red Hat. It is popular among desktop users. Its versions are known for their short life cycle.
	Gentoo	It is a source based Distribution which means that you need to configure the code on your system before you can install it. It is not for Linux beginners, but it is sure fun for experienced users.

Linux Distribution	Name	Description
	<b>LinuxMint</b>	It is one of the most popular Desktop Distributions available out there. It launched in 2006 and is now considered to be the fourth most used Operating system in the computing world.
	<b>OpenSUSE</b>	It is an easy to use and a good alternative to MS Windows. It can be easily set up and can also run on small computers with obsolete configurations.
	<b>RedHat enterprise</b>	Another popular enterprise based Linux Distribution is Red Hat Enterprise. It has evolved from Red Hat Linux which was discontinued in 2004. It is a commercial Distro and very popular among its clientele.
	<b>Slackware</b>	Slackware is one of the oldest Linux kernel based OS's. It is another easy desktop Distribution. It aims at being a 'Unix like' OS with minimal changes to its kernel.
	<b>Ubuntu</b>	This is the third most popular desktop operating system after Microsoft Windows and Apple Mac OS. It is based on the Debian Linux Distribution, and it is known as its desktop environment.

### **Installation of Linux:**

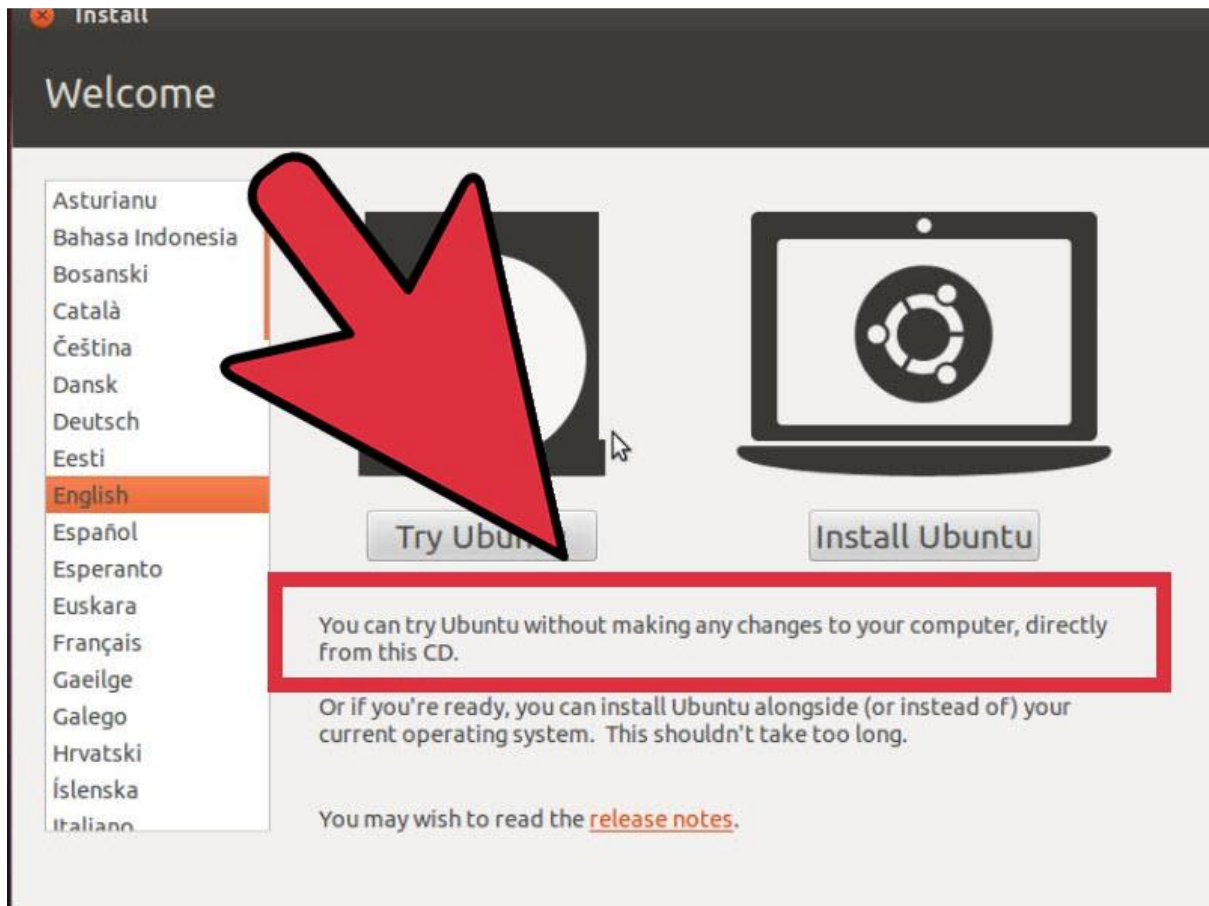
Linux is the foundation of thousands of open source operating systems designed to replace Windows and Mac OS. It is free to download and install on any computer. Because it is open source, there are a variety of different versions, or distributions, available developed by different groups.

#### **1. Installation of any Linux Distribution:**



**2. Download the Linux distribution of your choice.** If you're new to Linux, consider trying a lightweight and easy to use distribution, such as Ubuntu or Linux Mint. Linux distributions (known as "distros") are typically available for free to download in ISO format. You can find the ISO for the distribution of your choice at the distribution's website. This format needs to be burned to a CD or USB stick before you can use it to install Linux. This will create a Live CD or Live USB.

- A Live CD or Live USB is a disk that you can boot into, and often contains a preview version of the operating system that can be run directly from the CD or USB stick.
- Install an image burning program, or use your system's built-in burning tool if you are using Windows 7, 8, or Mac OS X. Pen Drive Linux and UNetBootin are two popular tools for burning ISO files to USB sticks.



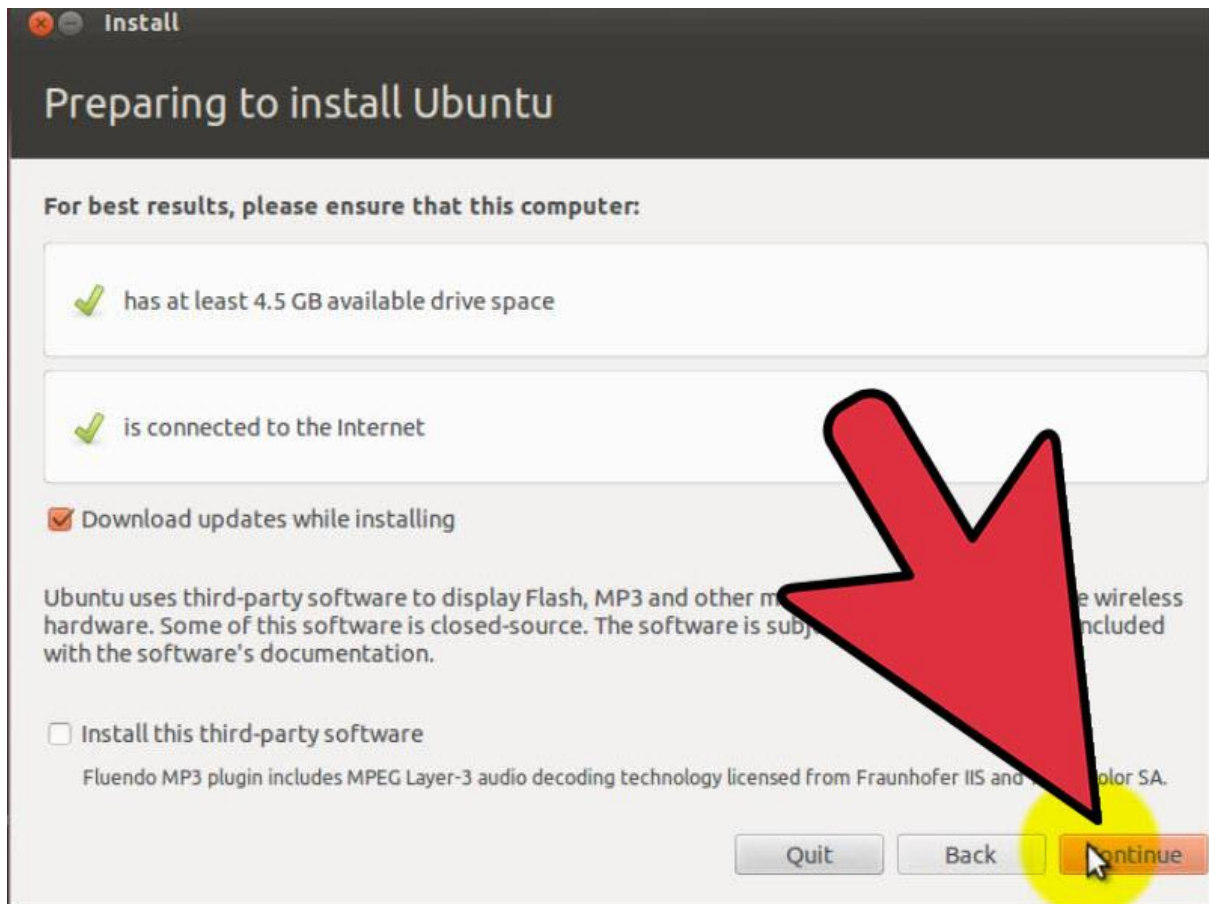
**3. Boot into the Live CD or Live USB.** Most computers are set to boot into the hard drive first, which means you will need to change some settings to boot from your newly-burned CD or USB. Start by rebooting the computer.

- Once the computer reboots, press the key used to enter the boot menu. The key for your system will be displayed on the same screen as the manufacturer's logo. Typical keys include F12, F2, or Del.
- For Windows 8 users, hold the Shift key and click restart. This will load the Advanced Startup Options, where you can boot from CD.
- For Windows 10 users, go to advanced boot in settings and click "Restart Now."
- If your computer doesn't give you direct access to the boot menu from the manufacturer's splash screen, it's most likely hidden in the BIOS menu. You can access the BIOS menu in the same way that you would get to the boot menu. At the manufacturer splash screen, the key should be listed in one of the bottom corners.
- Once you're in the boot menu, select your live CD or USB. Once you've changed the settings, save and exit the BIOS setup or boot menu. Your computer will continue with the boot process.



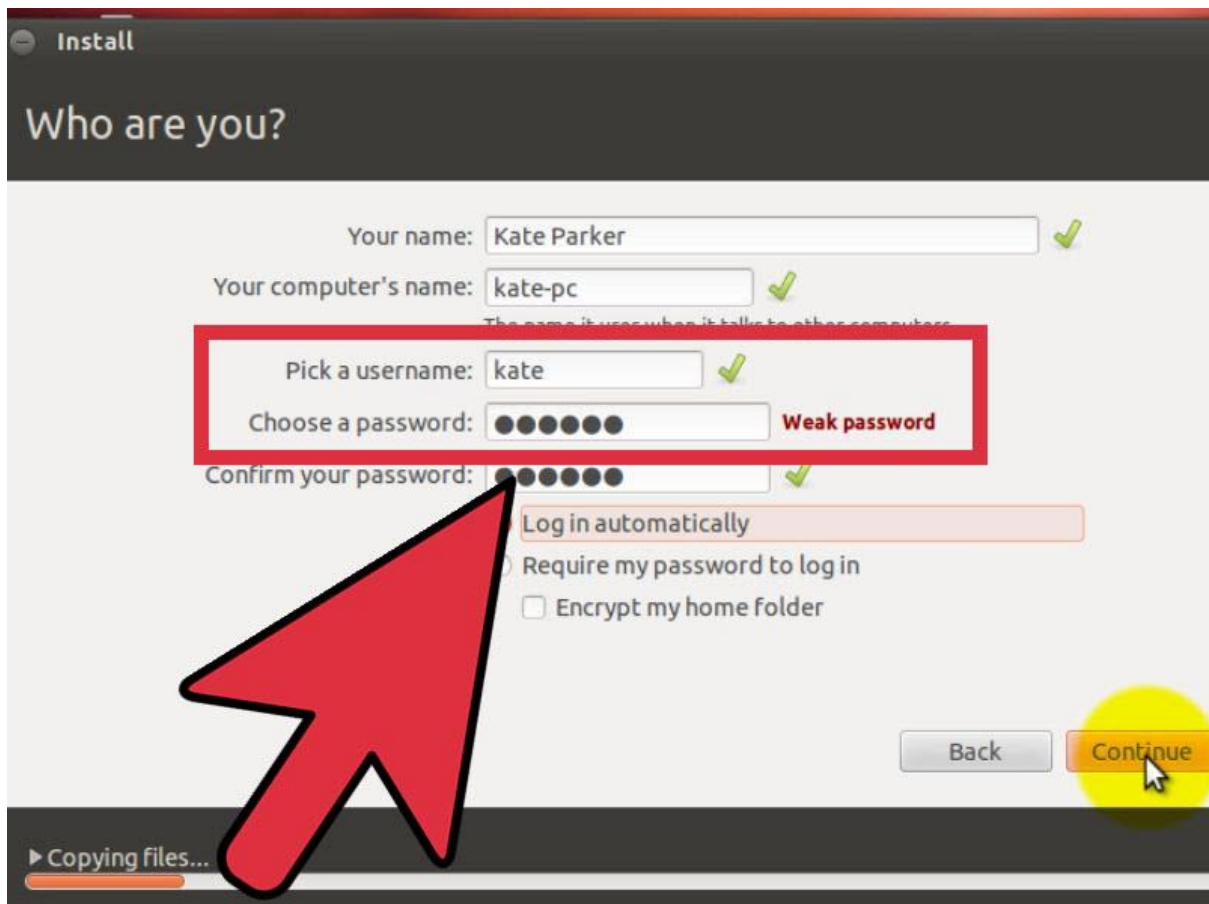
4. **Try out the Linux distribution before installing.** Most Live CDs and USBs can launch a "live environment", giving you the ability to test it out before making the switch. You won't be able to create files, but you can navigate around the interface and decide if it's right for you.





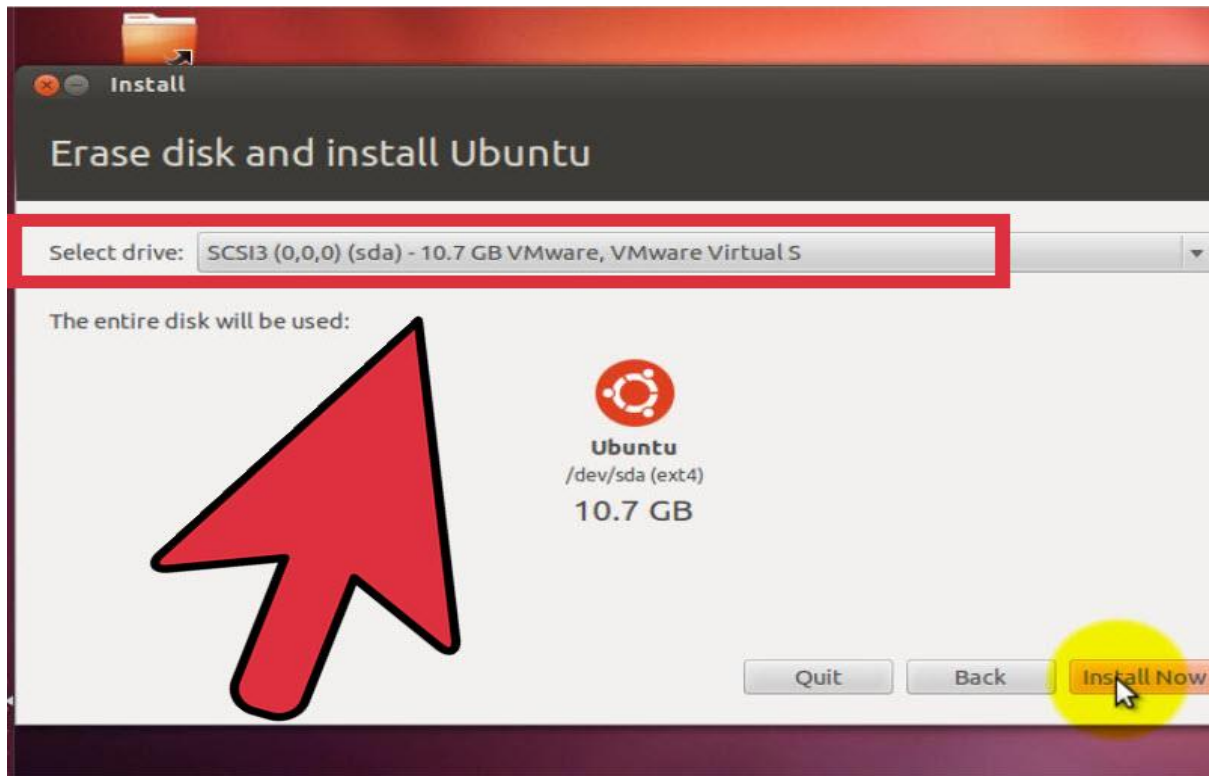
5. **Start the installation process.** If you're trying out the distro, you can launch the installation from the application on the desktop. If you decided not to try out the distribution, you can start the installation from the boot menu.

- You will be asked to configure some basic options, such as language, keyboard layout, and timezone.



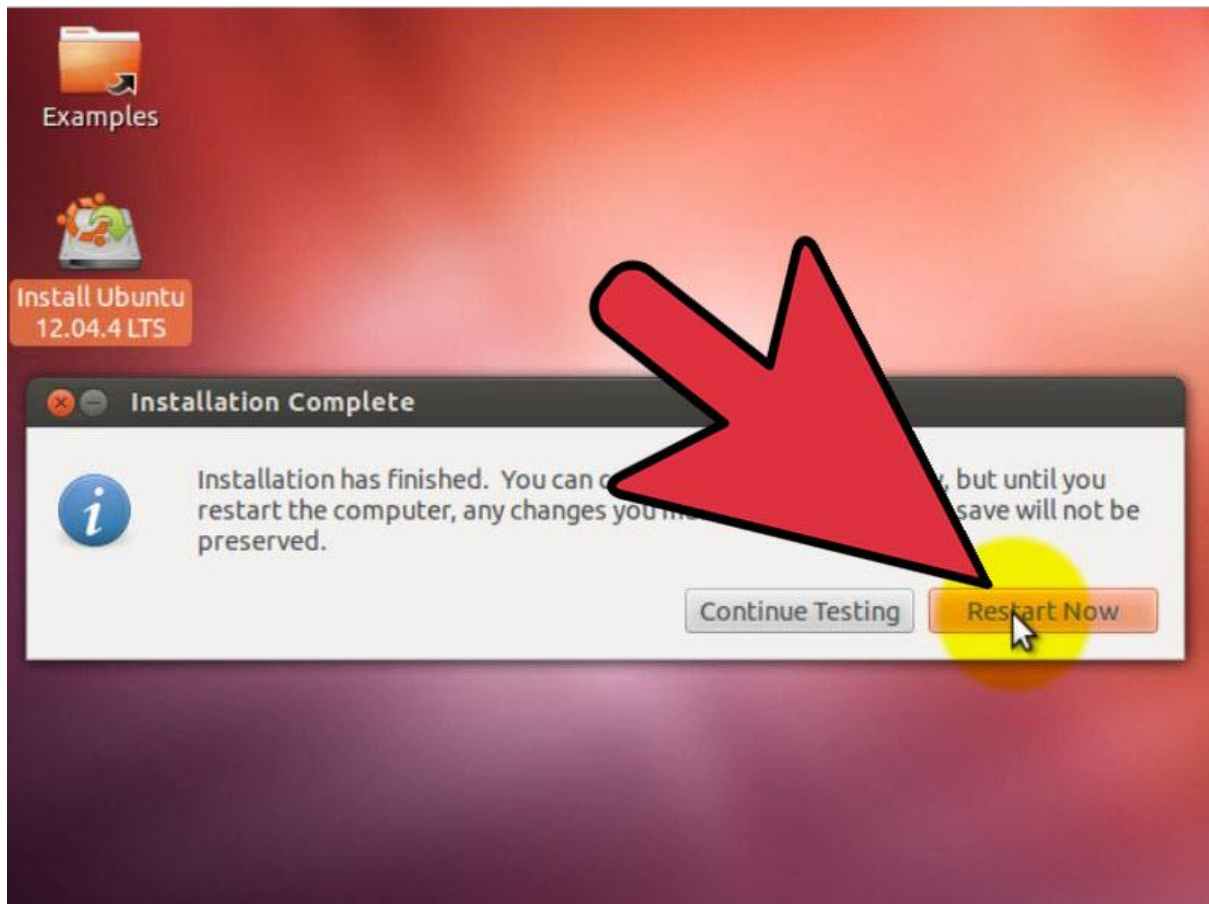
6. **Create a username and password.** You will need to create login information to install Linux. A password will be required to log into your account and perform administrative tasks.





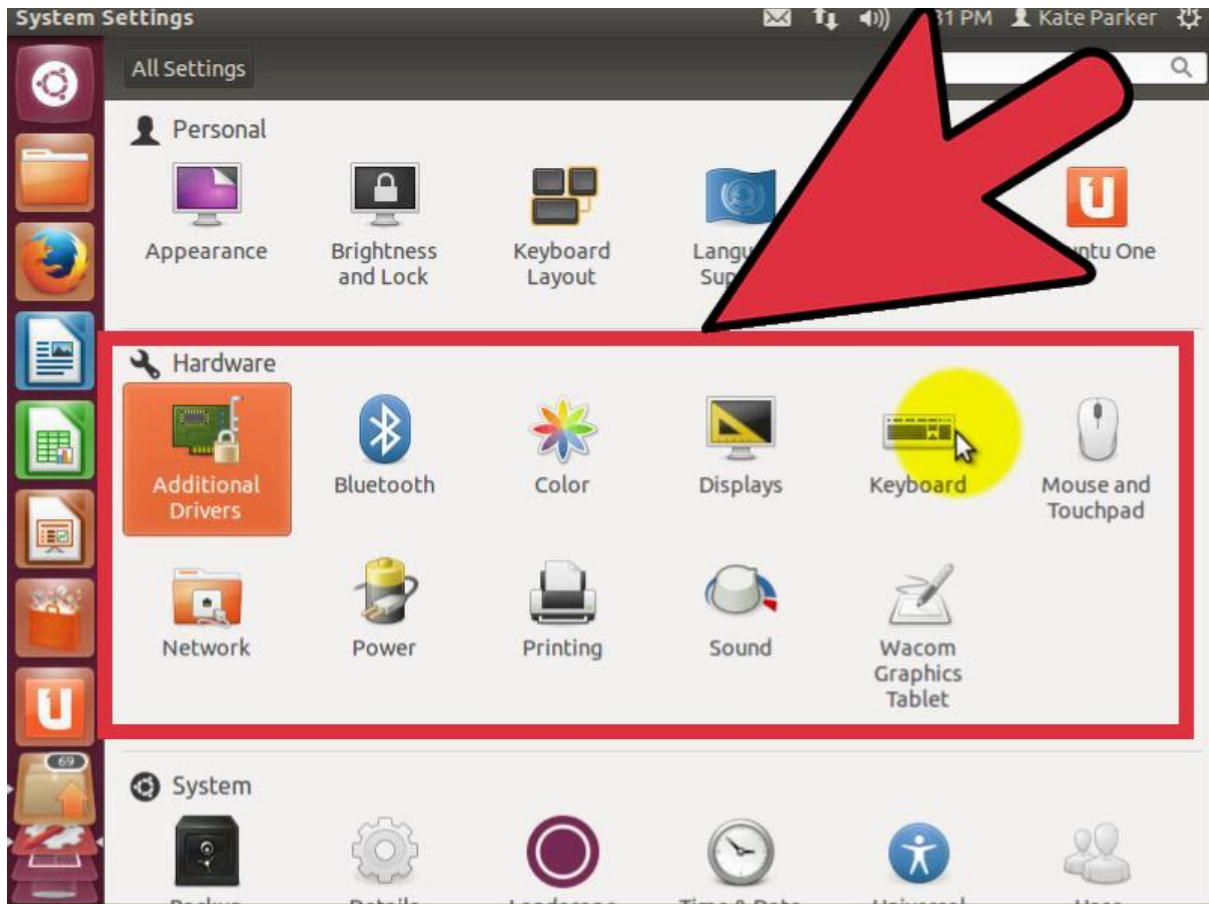
7. **Set up the partition.** Linux needs to be installed on a separate partition from any other operating systems on your computer if you intend dual booting Linux with another OS. A partition is a portion of the hard drive that is formatted specifically for that operating system. You can skip this step if you don't plan on dual booting.

- Distros such as Ubuntu will set a recommended partition automatically. You can then adjust this manually yourself. Most Linux installations require at least 20 GB, so be sure to set aside enough room for both the Linux operating system and any other programs you may install and files you may create.
- If the installation process does not give you automatic partitions, make sure that the partition you create is formatted as Ext4. If the copy of Linux you are installing is the only operating system on the computer, you will most likely have to manually set your partition size.



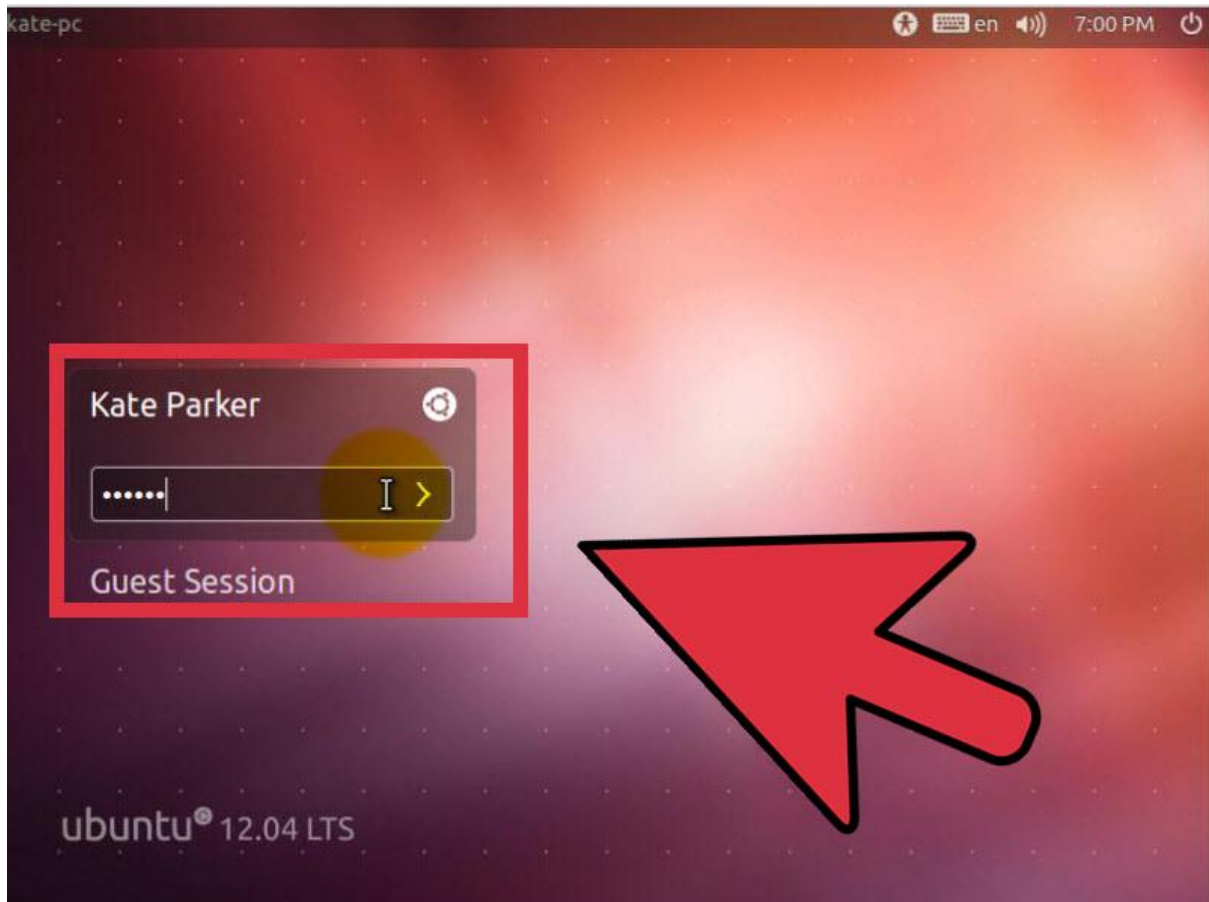
8. **Boot into Linux.** Once the installation is finished, your computer will reboot. You will see a new screen when your computer boots up called “GNU GRUB”. This is a boot loader that handles Linux installations. Pick your new Linux distro from the list. This screen may not show up if you only have one operating system on your computer. If this screen isn't being presented to you automatically, then you can get it back by hitting shift right after the manufacturer splash screen.

- If you install multiple distros on your computer, they will all be listed here.



9. **Check your hardware.** Most hardware should work out of the box with your Linux distro, though you may need to download some additional drivers to get everything working.

- Some hardware requires proprietary drivers to work correctly in Linux. This is most common with graphics cards. There is typically an open source driver that will work, but to get the most out of your graphics cards you will need to download the proprietary drivers from the manufacturer.
- In Ubuntu, you can download proprietary drivers through the System Settings menu. Select the Additional Drivers option, and then select the graphics driver from the list. Other distros have specific methods for obtaining extra drivers.
- You can find other drivers from this list as well, such as Wi-Fi drivers.



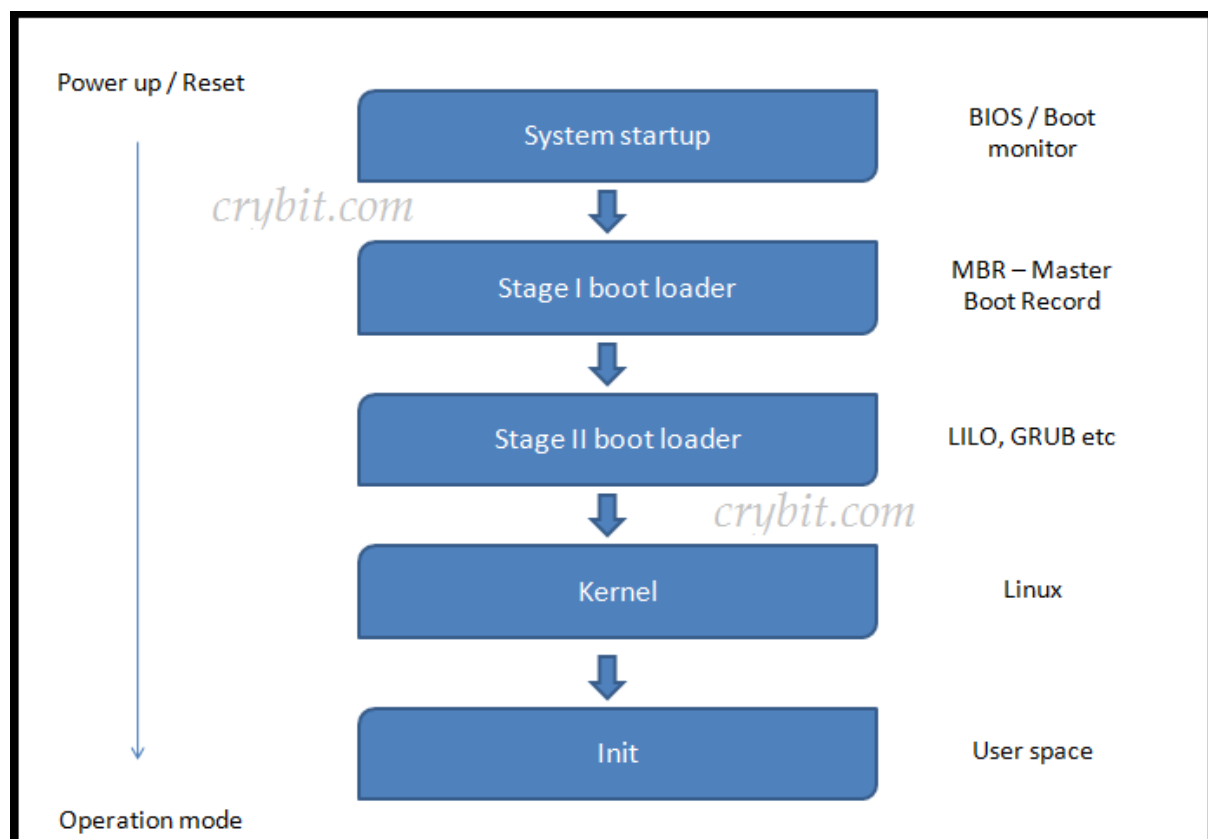
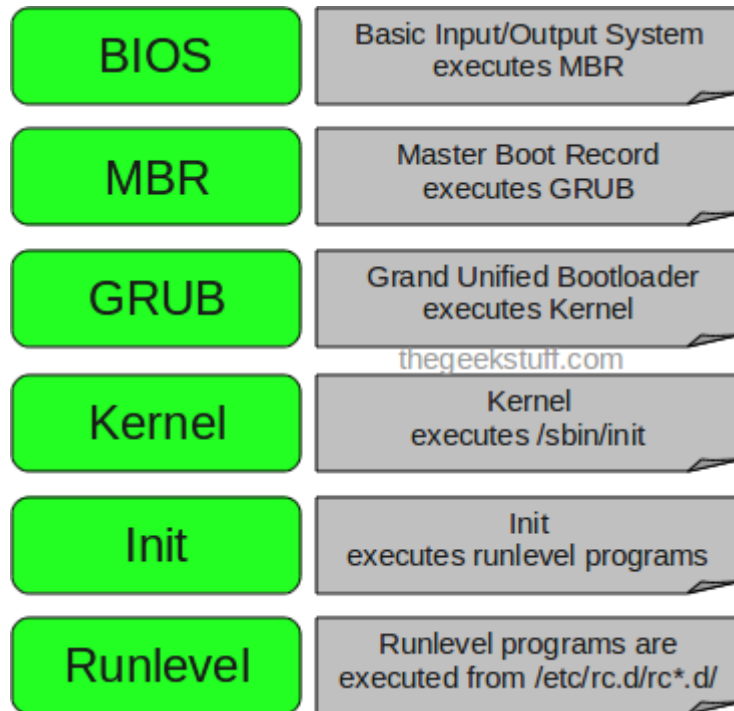
10. **Start using Linux.** Once your installation is complete and you've verified that your hardware is working, you're ready to start using Linux. Most distros come with several popular programs installed, and you can download many more from their respective file repositories.

### **BOOTING PROCESS IN LINUX**

Press the power button on your system, and after few moments you see the Linux login prompt.

Have you ever wondered what happens behind the scenes from the time you press the power button until the Linux login prompt appears?

The following are the 6 high level stages of a typical Linux boot process.



## 1. BIOS

- BIOS stands for Basic Input/Output System
- Performs some system integrity checks (System integrity checks determine the trustworthiness of a system as well as whether the data has been changed or damaged.)
- Searches, loads, and executes the boot loader program.
- It looks for boot loader in floppy, cd-rom, or hard drive. You can press a key (typically F12 or F2, but it depends on your system) during the BIOS startup to change the boot sequence.
- Once the boot loader program is detected and loaded into the memory, BIOS gives the control to it.
- So, in simple terms BIOS loads and executes the MBR boot loader.

## 2. MBR

- MBR stands for Master Boot Record.
- It is located in the 1st sector of the bootable disk. Typically /dev/hda, or /dev/sda (The first SCSI disk (SCSI ID address-wise) is named /dev/sda . ... The master disk on IDE primary controller is named /dev/hda . (Short for Small Computer System Interface)
- MBR is less than 512 bytes in size. This has three components 1) primary boot loader info in 1st 446 bytes 2) partition table info in next 64 bytes (A partition table is a 64-byte data structure that provides basic information for a computer's operating system about the division of the hard disk drive (HDD) into primary partitions.) 3) mbr validation check in last 2 bytes. (master boot record (MBR))
- It contains information about GRUB
- So, in simple terms MBR loads and executes the GRUB boot loader.

## 3. GRUB

- GRUB stands for Grand Unified Bootloader.
- If you have multiple kernel images installed on your system, you can choose which one to be executed.
- GRUB displays a splash screen, waits for few seconds, if you don't enter anything, it loads the default kernel image as specified in the grub configuration file.
- GRUB has the knowledge of the filesystem (the older Linux loader LILO didn't understand filesystem).
- Grub configuration file is /boot/grub/grub.conf (/etc/grub.conf is a link to this). The following is sample grub.conf of CentOS.

```
#boot=/dev/sda
```

```
default=0
```

```
timeout=5

splashimage=(hd0,0)/boot/grub/splash.xpm.gz

hiddenmenu

title CentOS (2.6.18-194.el5PAE)

    root (hd0,0)

    kernel /boot/vmlinuz-2.6.18-194.el5PAE ro root=LABEL=/

    initrd /boot/initrd-2.6.18-194.el5PAE.img
```

- As you notice from the above info, it contains kernel and initrd image. (Initial RAMDisk image is the image may be a file system image (optionally compressed), which is made available in a special block device (/dev/ram))
- So, in simple terms GRUB just loads and executes Kernel and initrd images.

#### 4. Kernel

- Mounts the root file system as specified in the “root=” in grub.conf
- Kernel executes the /sbin/init program
- Since init was the 1st program to be executed by Linux Kernel, it has the process id (PID) of 1. Do a ‘ps -ef| grep init’ and check the pid.
- initrd stands for Initial RAM Disk.
- initrd is used by kernel as temporary root file system until kernel is booted and the real root file system is mounted. It also contains necessary drivers compiled inside, which helps it to access the hard drive partitions, and other hardware.

#### 5. Init

- Looks at the /etc/inittab file to decide the Linux run level.
- Following are the available run levels
  - 0 – halt
  - 1 – Single user mode
  - 2 – Multiuser
  - 3 – Full multiuser mode
  - 4 – unused
  - 5 – X11 (while if the system is in a multi-user mode it will have a **runlevel 5**.)
  - 6 – reboot
- Init identifies the default initlevel from /etc/inittab and uses that to load all appropriate program.



- Execute ‘grep initdefault /etc/inittab’ on your system to identify the default run level
- If you want to get into trouble, you can set the default run level to 0 or 6. Since you know what 0 and 6 means, probably you might not do that.
- Typically you would set the default run level to either 3 or 5.

## **6. Runlevel programs**

- When the Linux system is booting up, you might see various services getting started. For example, it might say “starting sendmail .... OK”. Those are the runlevel programs, executed from the run level directory as defined by your run level.
- Depending on your default init level setting, the system will execute the programs from one of the following directories.
  - Run level 0 – /etc/rc.d/rc0.d/
  - Run level 1 – /etc/rc.d/rc1.d/
  - Run level 2 – /etc/rc.d/rc2.d/
  - Run level 3 – /etc/rc.d/rc3.d/
  - Run level 4 – /etc/rc.d/rc4.d/
  - Run level 5 – /etc/rc.d/rc5.d/
  - Run level 6 – /etc/rc.d/rc6.d/
- Please note that there are also symbolic links available for these directory under /etc directly. So, /etc/rc0.d is linked to /etc/rc.d/rc0.d.
- Under the /etc/rc.d/rc\*.d/ directories, you would see programs that start with S and K.
- Programs starts with S are used during startup. S for startup.
- Programs starts with K are used during shutdown. K for kill.
- There are numbers right next to S and K in the program names. Those are the sequence number in which the programs should be started or killed.
- For example, S12syslog is to start the syslog daemon, which has the sequence number of 12. S80sendmail is to start the sendmail daemon, which has the sequence number of 80. So, syslog program will be started before sendmail.

There you have it. That is what happens during the Linux boot process.

## **SHUT DOWN PROCESS**

The shutdown command brings down system in a secure way. All the logged-in users are notified about the system shutdown.

Signal SIGTERM notifies all the processes that the system is going down, so that processes can be saved and exit properly.

Command shutdown signals the init process to change the runlevel.

Runlevel 0 halts the system

Runlevel 6 reboots the system

Runlevel 1 is default state.



Five minutes before shutdown sequence starts, file /etc/nologin is created when shutdown is scheduled for future which does not allow new user logins.

If by any reason, command shutdown is stopped before signalling init, this file is removed. It is also removed to change runlevel before signalling init.

To run shutdown command root user access is required.

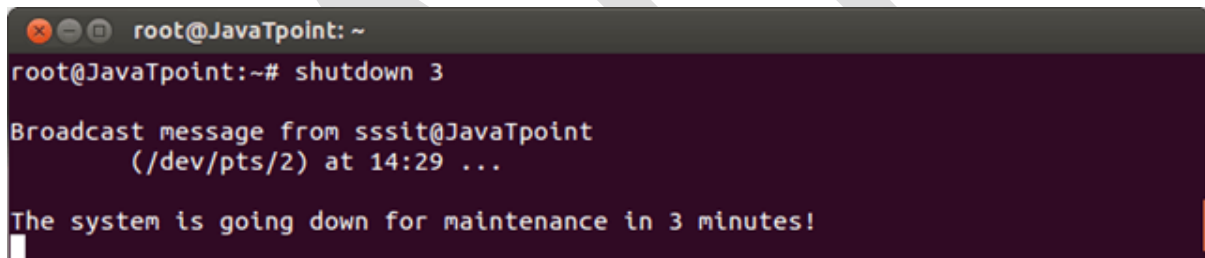
You can shutdown a system by passing a definite time (in minutes). System will automatically shutdown after specified minute giving a message and time to save all work.

**Syntax:**

1. shutdown <time>

**Example:**

shutdown 3

A terminal window with a dark background. The prompt is 'root@JavaTpoint: ~'. The command 'shutdown 3' has been entered. The output shows a broadcast message from 'sssit@JavaTpoint' on '/dev/pts/2' at '14:29 ...'. Below that, it says 'The system is going down for maintenance in 3 minutes!'.

Look at the above snapshot, message is displayed on the terminal.

To immediately shutdown the system, use **now** option,

**Syntax:**

1. shutdown now

System will shutdown immediately.

KARPAGAM

1. Which command is used to display the operating system name

- a) os
- b) unix
- c) kernel
- d) uname

Answer (d)

2. Which command is used to display the unix version

- a) uname -r
- b) uname -n
- c) uname -t
- d) kernel

Answer (a)

3. Which command is used to print a file

- a) print
- b) ptr
- c) lpr
- d) none of the mentioned

Answer (c )

4. Which option of ls command used to view file inode number

- a) -l
- b) -o
- c) -a
- d) -i

Answer (d)

5. What is UNIX?

- a) an operating system
- b) a text editor
- c) programming language
- d) software program

Answer (a)

6. In which language UNIX is written?

- a) JAVA
- b) Python

- c) C++
- d) C

Answer (d)

7. Which of the following is not a feature of UNIX?

- a) multitasking
- b) multiuser
- c) portability
- d) easy to use

Answer (d)

8. Which part of the UNIX operating system interacts with the hardware?

- a) Kernel
- b) Shell
- c) vi editor
- d) application program

Answer (a)

9. ----- is a command interpreter used for interacting with a UNIX system.

- a) Shell
- b) kernel
- c) vi editor
- d) application program

10. Which of the following is not a part of all the versions of UNIX?

- a) Kernel and Shell
- b) Commands and utilities
- c) Graphical user interface
- d) System Calls

Answer (c )

11. What are Commands?

- a) specific instructions for performing a particular task
- b) part of the operating system
- c) part of the shell
- d) special instructions

Answer (a)

12. In how many categories, commands of UNIX operating system classified?

- a) 1
- b) 2
- c) Many
- d) 0

Answer (b)

13. Which command is used for extracting the details of the operating system?

- a) cd
- b) echo
- c) uname
- d) wc

Answer (c )

14. How can we specify more than one command in the command line at the same time?

- a) using ;
- b) using >
- c) using ==
- d) not possible

Answer (a)

15. Which command is used to display the documentation of commands in UNIX?

- a) help
- b) search
- c) whatis
- d) man

Answer (d)

16. Which command is used for displaying date and calendar in UNIX?

- a) date and cal
- b) DATE and CAL
- c) date and calendar
- d) dt and cl

Answer (a)

17. What is the output of who command?

- a) display information about users who are currently logged in.
- b) display file hierarchy
- c) display administrator information
- d) display processes

Answer (a)

18. What are meta-characters?

- a) special characters having predefined meaning to the shell
- b) special symbols
- c) shell symbols
- d) command symbols

Answer (a)

19. Which command is used for displaying date in the format dd/mm/yyyy ?

- a) date +%m
- b) date +%h
- c) date +"%d/%m/%Y"
- d) date +"%h %m"

Answer (c )

20. echo command is used for \_\_\_\_\_

- a) displaying errors
- b) displaying operating system details
- c) displaying diagnostic messages
- d) displaying date and time

Answer (c )

21. Which command is used as an alternative to echo command?

- a) ls
- b) printf
- c) wc
- d) ps

Answer (b)

22. script command is used for \_\_\_\_\_

- a) recording history
- b) recording our session
- c) recording passwords
- d) recording scripts

Answer (b)

23. Which of the following is not true about UNIX?

- a) Many people can use a UNIX based computer at the same time; hence UNIX is called as a multiuser system
- b) A user can run multiple programs at the same time; hence UNIX is called a multitasking environment
- c) UNIX was not written in 'C' language
- d) Linux is also known as a version of UNIX

Answer (c)

24. Who is a superuser?

- a) system manager
- b) normal user
- c) administrator
- d) a user with special rights

Answer (a)

25. What is the windowing system of UNIX known as?

- a) X Window system
- b) LINUX
- c) Red Hat
- d) DOS

Answer (a)

26. In vi-editor, for navigation purposes, the mode should be \_\_\_\_\_ mode.

- a) command
- b) input
- c) insert
- d) ex

Answer: (a)

27. In vi-editor, Using 'b' command we can \_\_\_\_\_

- a) move forward to the end of the word
- b) move back to the beginning of the word
- c) move forward to the beginning of the word
- d) move back to the end of the word

Answer: (b)

28. In vi-editor, Using 'e' command we can \_\_\_\_\_

- a) move back to the beginning of the word
- b) move forward to the end of the word

- c) move forward to the beginning of the word
- d) move back to the end of the word

Answer: (b)

29. In vi-editor, Using 'w' command we can \_\_\_\_\_

- a) move back to the beginning of the word
- b) move forward to the end of the word
- c) move forward to the beginning of the word
- d) move back to the end of the word

Answer: (c )

30. On a UNIX system, there can be \_\_\_\_ shells running simultaneously.

- a) 1
- b) 2
- c) many
- d) 4

Answer: (c )

31. The prompt issued by the shell is called \_\_\_\_\_

- a) prompt
- b) command translator
- c) command prompt
- d) command executor

Answer: (c )

32. In UNIX there are \_\_\_\_ major types of shells.

- a) 2
- b) 3
- c) 4
- d) many

Answer: (a)



33. What is the default symbol for command prompt in Bourne shell?

- a) \$
- b) %
- c) #
- d) @

Answer: (a)

34. What is the default symbol for command prompt in C shell?

- a) \$
- b) %
- c) #
- d) @

Answer: (b)

35. Which one of the following command will display the name of the shell we are working on?

- a) echo shell
- b) echo \$
- c) echo \$SHELL
- d) echo \$\$

Answer: (c )

36. Which shell is the most common and best to use?

- a) Korn shell
- b) POSIX shell
- c) C shell
- d) Bash shell

Answer: (d)

37. Which command does not terminates unless we log out of the system?

- a) history
- b) shell
- c) echo
- d) login

Answer: (b)

38. In Shell's interpretive cycle, the shell first scans for \_\_\_\_ in the entered command.

- a) characters
- b) priority
- c) meta-characters
- d) wildcards

Answer: (c )

39. Which of the following is/are true about Shell?

- a) Shell is a multi-faceted program
- b) Shell is a command interpreter
- c) Shell provides us with an environment to work in
- d) Shell is a multi-faceted, command interpreter and provides an environment to work in

Answer: (d)

40. To change the login shell in some system (like Linux) we can use \_\_\_\_ command.

- a) chshell
- b) chshl
- c) chsh
- d) ch

Answer: (c )

41. The UNIX shell is both \_\_\_\_\_ and \_\_\_\_\_ language.

- a) interactive, responsive
- b) interpreter, executing
- c) scripting, interpreter
- d) high level, low level

Answer: (c )

42. Which one of the following command is used to create a child shell?

- a) fork
- b) wait
- c) sh
- d) env

Answer: (c )

43. Which of the following function(s) are performed by an interactive shell?

- a) job control
- b) history
- c) aliases
- d) job control, history, aliases

Answer: (d)

44. Which one of the following is not an environment variable?

- a) HOME
- b) PATH
- c) USER
- d) env

Answer: (d)

45. Which environment variable is used to display our username?

- a) PATH
- b) MAIL
- c) LOGNAME
- d) HOME

Answer: (c )

46. \_\_\_\_\_ is the commercial UNIX-based operating system of Sun Microsystems.

- a) Solaris
- b) UNIX
- c) Linux
- d) Macintosh

Answer: (a)

47. \_\_\_\_\_ is an example of an open-source operating system.

- a) Windows 3.1
- b) Windows NT
- c) Macintosh
- d) GNU/Linux

Answer: (d)

48. The kernel is a \_\_\_\_\_

- a) Memory manager
- b) Resource manager
- c) File manager
- d) Directory manager

Answer: (b)

## UNIT-II

### SYLLABUS

External and internal commands, Creation of partitions in OS, Processes and its creation phases – Fork, Exec, wait

#### Internal and External Commands in Linux:

The UNIX system is command-based *i.e* things happen because of the commands that you key in. All UNIX commands are seldom more than four characters long.

**They are grouped into two categories:**

- **Internal Commands:** Commands which are built into the shell. For all the shell built-in commands, execution of the same is fast in the sense that the shell doesn't have to search the given path for them in the PATH variable and also no process needs to be spawned for executing it. Examples: source, cd, fg etc.
- **External Commands:** Commands which aren't built into the shell. When an external command has to be executed, the shell looks for its path given in PATH variable and also a new process has to be spawned and the command gets executed. They are usually located in /bin or /usr/bin. For example, when you execute the "cat" command, which usually is at /usr/bin, the executable /usr/bin/cat gets executed. Examples: ls, cat etc.

If you know about UNIX commands, you must have heard about the **ls** command. Since **ls** is a program or file having an independent existence in the /bin directory(or /usr/bin), it is branded as an **external command** that actually means that the ls command is not built into the shell and these are executables present in separate file. In simple words, when you will key in the ls command, to be executed it will be found in /bin. Most commands are external in nature, but there are some which are not really found anywhere, and some which are normally not executed even if they are in one of the directories specified by PATH. For instance, take **echo** command:

#### **\$type echo**

echo is a shell builtin

**echo** isn't an external command in the sense that, when you type **echo**, the shell won't look in its PATH to locate it (even if it is there in /bin). Rather, it will execute it from its own set of built-in commands that are not stored as separate files. These built-in commands, of which **echo** is a member, are known as **internal commands**.

You now might have noticed that it's the shell that actually does all this works. This program starts running as soon the user log in, and dies when the user log out. The shell is an external command with a difference, it possesses its own set of internal commands. So, if a command exists both as an internal command of the shell as well as external one(in /bin or /usr/bin), the shell will accord top priority to its own internal command of the same name.

This is exactly the case with **echo** which is also found in /bin, but rarely ever executed because the shell makes sure that the internal **echo** command takes precedence over the external. Now, talk more about the internal and external commands.

### Getting the list of Internal Commands

If you are using bash shell you can get the list of shell built-in commands with **help** command:

```
$help
```

```
// this will list all  
the shell built-in commands //
```

### How to find out whether a command is internal or external?

In addition to this you can also find out about a particular command *i.e* whether it is internal or external with the help of **type** command :

```
$type cat
```

```
cat is /bin/cat
```

```
//specifying that cat is  
external type//
```

```
$type cd
```

```
cd is a shell builtin
```

```
//specifying that cd is  
internal type//
```

### Internal

vs

### External

The question that when to use which command between internal and external command is of no use cause the user uses a command according to the need of the problem he wants to solve. The only difference that exists between internal and external command is that internal commands work much faster than the external ones as the shell has to look for the path when it comes to the use of external commands.

There are some cases where you can avoid the use of external by using internal in place of them, like if you need to add two numbers you can do it as:

```
//use of internal command let
```

```
for addition//
```

```
$let c=a+b
```

instead of using :

```
//use of external command expr
```

```
for addition//
```

```
$c=`expr $a+$b`
```

In such a case, use of let will be more better option as it is a shell built-in command so will work faster than the expr which is an external command.

## Creation of partitions in OS

Partitions are divisions in the formatting of the hard disk. It's a logical – as opposed to a physical – division, so you can edit and manipulate them for various purposes. Think breaking a disk into two configuration parts. Partitions are really handy because they act as a sandbox. If you have a 1 TB hard drive partitioned into a 250 GB partition and a 750 GB partition, what you have on the latter will not affect the other, and vice versa. You can share one of those partitions on the network and never worry about people accessing information on the other. One could have Windows installed, riddled with viruses and trojans. The other could be running a very obsolete, security-hole added Linux installation. Never shall the two interfere.

The other useful thing is that you can have multiple partitions, each formatted with a different “file system.” A file system is a formatting of the disk into a table that the operating system can read, interpret, and write to. With the only one hard disk, you can install multiple operating systems on it without actually having another physical disk. While there are tons of file system types, there are only three kinds of partitions: primary, extended, and logical. Any given hard disk can only have a maximum of four primary partitions. This limitation is due to something called the Master Boot Record which tells the computer which partitions it can boot from, and so primary partitions are usually reserved for operating systems.

Partitioning enables you to split your hard drive into multiple parts, where each part acts as its own hard drive and this is useful when you are installing multiple operating systems in the same machine.

In this section, it is explained how to partition a storage disk in Linux using the parted command.

(i) The first step is to view the partition table or layout on all block devices. This helps you identify the storage device you want to partition. You can do this using parted or fdisk command. We will use the former for purposes of demonstration, as follows, where the -l flag means list partition layout on all block devices.

```
# parted -l
```

From the output of the above command, there are two hard disks attached to the test system, the first is /dev/sda and the second is /dev/sdb.

In this case, we want to partition hard disk /dev/sdb. To manipulate disk partitions, open the hard disk to start working on it, as shown.

```
# parted /dev/sdb
```

At the parted prompt, make a partition table by running mklabel msdos or gpt, then enter Y/es to accept it.



```
(parted) mklabel msdos
```

```
[root@server1 ~]# parted /dev/sdb
GNU Parted 3.2
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) mklabel msdos
Warning: The existing disk label on /dev/sdb will be destroyed and all data on
this disk will be lost. Do you want to continue?
Yes/No? Y
(parted)
```

Important: Make sure to specify the correct device for partition in the command. If you run parted command without a partition device name, it will randomly pick a storage device to modify.

Next, create a new primary partition on the hard disk and print the partition table as shown.

```
(parted) mkpart primary ext4 0 10024MB

(parted) print
```

To quit, issue the quit command and all changes are automatically saved.

Next, create the file system type on each partition, you can use the mkfs utility (replace ext4 with the file system type you wish to use).

```
# mkfs.ext4 /dev/sdb1

# mkfs.ext4 /dev/sdb2
```

Last but not least, to access the storage space on the partitions, you need to mount them by creating the mount points and mount the partitions as follows.

```
# mkdir -p /mnt/sdb1
```

```
# mkdir -p /mnt/sdb2

# mount -t auto /dev/sdb1 /mnt/sdb1

# mount -t auto /dev/sdb2 /mnt/sdb2
```

To check if the partitions are actually mounted, run the [df command](#) to report file system disk space usage.

```
# df -hT
```

You may need to update /etc/fstab file to mount newly created partitions automatically at boot time.

## **Process in UNIX/LINUX**

A **process** is a program in execution in memory or in other words, an instance of a program in memory. Any program executed creates a process. A program can be a command, a shell script, or any binary executable or any application. a process has lots of properties associated to it.

### **Process attributes:**

A process has some properties associated to it:

**PID** : Process-Id. Every process created in Unix/Linux has an identification number associated to it which is called the process-id. This process id is used by the kernel to identify the process similar to how the inode number is used for file identification. The PID is unique for a process at any given point of time. However, it gets recycled.

**PPID** : Parent Process Id: Every process has to be created by some other process. The process which creates a process is the parent process, and the process being created is the child process. The PID of the parent process is called the parent process id(PPID). At any

point of time, no two processes with the same pid exist in the system because it is the pid that Unix uses to track each process.

**TTY:** Terminal to which the process is associated to. Every command is run from a terminal which is associated to the process. However, not all processes are associated to a terminal. There are some processes which do not belong to any terminal. These are called daemons.

**UID:** User Id- The user to whom the process belongs to. And the user who is the owner of the process can only kill the process (Of course, root user can kill any process). When a process tries to access files, the accessibility depends on the permissions the process owner has on those files.

### Parent & Child Process:

Every process in Unix has to be created by some other process. Hence, the ps command is also created by some other process. The 'ps' command is being run from the login shell, ksh. The ksh shell is a process running in the memory right from the moment the user logged in. So, for all the commands triggered from the login shell, the login shell will be the parent process and the process created for the command executed will be the child process. In the same lines, the 'ksh' is the parent process for the child process 'ps'.

The below command shows the process list along with the PPID.

```
$ ps -o pid,ppid,args
```

```
PID  PPID COMMAND
```

```
2666744 3317840 ps -o pid,ppid,args
```

```
3317840 1 -ksh
```

The PID of the ksh is same as the PPID of the ps command which means the ksh process is the parent of the ps command. The '-o' option of the ps command allows the user to specify only the fields which he needs to display.

### Init Process:

When the Unix system boots, the first process to be created is the *init* process. This init process will have the PID as 1 and PPID as 0. All the other processes are created by the init process and gets branched from there on. Note in the above command, the process of the login shell has the PPID 1 which is the PID of the *init* process.

### Initializing a process

A process can be run in two ways:

1. **Foreground Process** : Every process when started runs in foreground by default, receives input from the keyboard and sends output to the screen.  
When issuing pwd command

2. **\$ ls pwd**

Output:

```
$ /home/geeksforgeeks/root
```

When a command/process is running in the foreground and is taking a lot of time, no other processes can be run or started because the prompt would not be available until the program finishes processing and comes out.

3. **Background Process** : It runs in the background without keyboard input and waits till keyboard input is required. Thus, other processes can be done in parallel with the process running in background since they do not have to wait for the previous process to be completed.

Adding & along with the command starts it as a background process

4. **\$ pwd &**

Since pwd does not want any input from the keyboard, it goes to the stop state until moved to the foreground and given any data input. Thus, on pressing Enter, :

Output:

```
[1]  +  Done                pwd
$
```

That first line contains information about the background process – the job number and the process ID. It tells you that the ls command background process finishes successfully. The second is a prompt for another command.

### Tracking ongoing processes

ps (Process status) can be used to see/list all the running processes.

```
$ ps
```

PID	TTY	TIME	CMD
19	pts/1	00:00:00	sh
24	pts/1	00:00:00	ps

For more information -f (full) can be used along with ps

```
$ ps -f
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
52471	19	1	0	07:20	pts/1	00:00:00	f sh
52471	25	19	0	08:04	pts/1	00:00:00	ps -f

For a single process information, ps along with process id is used

```
$ ps 19
```

PID	TTY	TIME	CMD
19	pts/1	00:00:00	sh

For a running program (named process) **Pidof** finds the process id's (pids)

**Fields described by ps are described as:**

**UID:** User ID that this process belongs to (the person running it)

**PID:** Process ID

**PPID:** Parent process ID (the ID of the process that started it)

**C:** CPU utilization of process

**STIME:** Process start time

**TTY:** Terminal type associated with the process

**TIME:** CPU time taken by the process

**CMD:** The command that started this process

There are other options which can be used along with ps command :

**-a:** Shows information about all users

**-x:** Shows information about processes without terminals

**-u:** Shows additional information like -f option

**-e:** Displays extended information

### Stopping a process

When running in foreground, hitting Ctrl + c (interrupt character) will exit the command. For processes running in background kill command can be used if it's pid is known.

```
$ ps -f
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
52471	19	1	0	07:20	pts/1	00:00:00	sh
52471	25	19	0	08:04	pts/1	00:00:00	ps -f

```
$ kill 19
```

Terminated

If a process ignores a regular kill command, you can use kill -9 followed by the process ID .

```
$ kill -9 19
```

Terminated

## Process Control Functions

### The fork() Function

As already discussed in the article [creating a daemon process in C](#), the **fork** function is used to create a process from within a process.

The resultant new process created by `fork()` is known as child process while the original process (from which `fork()` was called) becomes the parent process.

The function `fork()` is called once (in the parent process) but it returns twice. Once it returns in the parent process while the second time it returns in the child process. Note that the order of execution of the parent and the child may vary depending upon the process scheduling algorithm. So we see that `fork` function is used in **process creation**.

The signature of `fork()` is :

```
pid_t fork(void);
```

### The exec Family of Functions

Another set of functions that are generally used for creating a process is the **exec** family of functions. These functions are mainly used where there is a requirement of running an existing binary from within a process.

For example, suppose we want to run the 'whoami' command from within a process, then in these kind of scenarios the `exec()` function or other members of this family is used. A point worth noting here is that with a call to any of the `exec` family of functions, the current process image is replaced by a new process image.

A common member of this family is the `execv()` function. Its signature is :

```
int execv(const char *path, char *const argv[]);
```

**Note:** Please refer to the man-page of `exec` to have a look at the other members of this family.

## The wait() and waitpid() Functions

There are certain situations where when a child process terminates or changes state then the parent process should come to know about the change of the state or termination status of the child process. In that case functions like **wait()** are used by the parent process where the parent can query the change in state of the child process using these functions.

The signature of wait() is :

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

For the cases where a parent process has more than one child processes, there is a function **waitpid()** that can be used by the parent process to query the change state of a particular child.

The signature of waitpid() is :

```
pid_t waitpid(pid_t pid, int *status, int options);
```

By default, waitpid() waits only for terminated children, but this behavior is modifiable via the options argument, as described below.

The value of pid can be:

- < -1 : Wait for any child process whose process group ID is equal to the absolute value of pid.
- -1 : Wait for any child process.
- 0 : Wait for any child process whose process group ID is equal to that of the calling process.
- > 0 : Wait for the child whose process ID is equal to the value of pid.

The value of options is an OR of zero or more of the following constants:

- WNOHANG : Return immediately if no child has exited.
- WUNTRACED : Also return if a child has stopped. Status for traced children which have stopped is provided even if this option is not specified.
- WCONTINUED : Also return if a stopped child has been resumed by delivery of SIGCONT.

## UNIT II

1. nice command is a \_\_\_\_\_ command in C shell.

- a) internal
- b) external
- c) built-in
- d) directory

Answer: c

2. Which option can be used explicitly to reduce the priority of any process.

- a) -a
- b) -n
- c) -o
- d) -q

Answer: b

3. Which command is used for premature termination of a process?

- a) signal
- b) nice
- c) kill
- d) nohup

Answer: c

4. Which one of the following command is used for killing the last background job?

- a) kill \$
- b) kill \$\$
- c) kill \$!
- d) kill !

Answer: c

5. Which of the followings command(s) is used to kill the login shell?

- a) kill 0
- b) kill -9 \$\$
- c) kill -s KILL 0
- d) kill -9 \$\$ and kill -s KILL 0

Answer: d

6. A process is an instance of \_\_\_\_\_ program.

- a) waiting
- b) executing
- c) terminated
- d) halted

Answer: b

7. A process is said to be \_\_\_\_\_ when it starts its execution.

- a) born
- b) die
- c) waiting
- d) terminated

Answer: a



8. When the process has completed its execution it is called \_\_\_\_\_

- a) born
- b) terminated
- c) waiting
- d) exit

Answer: d

9. Which data structure is used to store information about a process?

- a) process control block (pcb)
- b) array
- c) queue
- d) program control block

Answer: a

10. Some attributes of every process are maintained by the kernel in memory in a separate structure called the \_\_\_\_\_

- a) pcb
- b) task control block
- c) process table
- d) task table

Answer: c

11. Each process is identified by a unique integer called \_\_\_\_\_

- a) PID
- b) PPID
- c) TID
- d) PTID

Answer: a

12. The parent id of a child is called \_\_\_\_\_

- a) PID
- b) PPID
- c) TID
- d) PTID

Answer: b

13. Which process is immediately set up by the kernel when we log on to a UNIX system?

- a) shell
- b) parent
- c) kernal
- d) bash

Answer: a

14. To know the PID of your current shell, which command will be used?

- a) echo \$\$
- b) echo \$
- c) \$SHELL
- d) \$PATH

Answer: a

15. What is the PID of the first process that is set up when the system is booted?

- a) 1

- b) 0
- c) any
- d) 2

Answer: b

16. Which of the following command doesn't create a process?

- a) pwd
- b) fork
- c) cd
- d) pwd and cd

Answer: d

17. Which command shows some attributes of a process?

- a) pid
- b) \$\$
- c) ps
- d) HOME

Answer: c

18. Which of the following attribute is not shown by ps command?

- a) PID
- b) PPID
- c) tty
- d) size

Answer: d

19. Which option is used by ps command to get a detailed listing of process attributes?

- a) -u
- b) -f
- c) -l
- d) -x

Answer: b

20. Which option is used by the system administrator for displaying processes of a user?

- a) -f
- b) -u
- c) -a
- d) -e

Answer: b

21. Which option is used with ps command to list system processes?

- a) -A
- b) -a
- c) -e
- d) -A and -e

Answer: d

22. What will the output of the following command?

```
$ ps -t dev/console
```

- a) processes running on terminal named console
- b) undefined output
- c) erroneous
- d) processes running on the current terminal

Answer: a

23. There are \_\_\_\_ distinct phases of a process.

- a) 2
- b) 5
- c) 4
- d) 3

Answer: d

24. Which of the following system call is used for creating a new process?

- a) read
- b) fork
- c) wait
- d) new

Answer: b

25. What is the value returned by fork system call, when the creation of child process is unsuccessful?

- a) positive integer
- b) negative integer
- c) zero
- d) fractional value

Answer: b

26. Which system call is used to run a new program?

- a) fork
- b) wait
- c) exec
- d) exit

Answer: c

27. Which system call is used by the parent process to wait for the child process to complete?

- a) wait
- b) exec
- c) fork
- d) exit

Answer: b

28. Shell \_\_\_\_ operator is used for running jobs in the background.

- a) \$
- b) #
- c) |
- d) &

Answer: d

29. Which command is used for running jobs in the background?

- a) nice
- b) ps

- c) nohup
- d) exec

Answer: c

30. Which of the following shell(s) allows the user to run jobs in the background even when the user has logged out (without using nohup command)?

- a) C
- b) bash
- c) Korn
- d) C and bash

Answer: d

31. When nohup command is used, shells returns the \_\_\_\_\_

- a) PID
- b) PPID
- c) tty
- d) TTy

Answer: a

32. What is the PID of the process who takes the parentage of the process run with nohup command?

- a) 0
- b) 1
- c) 2
- d) Infinite

Answer: b

33. Which command is used for executing jobs according to their priority?

- a) nohup
- b) \$
- c) &
- d) nice

Answer: d

34. What is a job?

- a) group of tasks
- b) group of commands
- c) group of processes
- d) group of signals

Answer: c

35. Which of the following command is used to suspend a job?

- a) ctrl-Z
- b) ctrl-Q
- c) bg
- d) \$

Answer: a

36. Which command will push the current foreground job to the background?

- a) bg
- b) fg

- c) ctrl-Z
- d) kill

Answer: a

37. \_\_\_\_ command will bring the background jobs to the foreground.

- a) bg
- b) fg
- c) ctrl-Z
- d) kill

Answer: b

38. What does the following command do?

\$ kill %2

- a) kills job number 2
- b) kills the second background job
- c) invalid command
- d) kill all foreground & background jobs

39. Which of the following is not a part of job control facilities?

- a) relate a job to the background
- b) bring it back to the foreground
- c) kill a job
- d) create a new job

Answer: d

40. Which command is used to list the status of jobs?

- a) fg
- b) JOBS
- c) jobs
- d) fg

Answer: c

41. We can schedule a job to run at a specified time of day using \_\_\_\_\_ command.

- a) batch
- b) at
- c) cron
- d) jobs

Answer: b

42. We can use the \_\_\_\_\_ symbol with at command to redirect our output to a specified file.

- a) >
- b) <
- c) >>
- d) %

Answer: a

43. Which of the following keyword is not supported by at command?

- a) now
- b) noon
- c) tomorrow
- d) evening

Answer: d

44. Which one of the following forms used with at command is invalid?

- a) at noon
- b) at now +2 years
- c) at 3:07 + 1 day
- d) at morning

Answer: d

45. We can list the jobs queued using at command by using \_\_\_\_ option.

- a) -p
- b) -v
- c) -l
- d) -r

Answer: c

46. To remove a job from the queue, which option is used with at command?

- a) -r
- b) -l
- c) -e
- d) -t

Answer: a

47. Which command permits to schedule jobs for later execution, as soon as the system load permits?

- a) at
- b) %
- c) batch
- d) cron

Answer: c

48. Jobs scheduled using batch command can be removed using \_\_\_\_ option.

- a) -a
- b) -d
- c) -f
- d) -r

Answer: d

49. What is a daemon?

- a) process whose parent has died
- b) process who has completed its execution but still has an entry in the process table
- c) process which is running infinitely
- d) process which runs automatically without any user interaction

Answer: d

50. What is cron?

- a) a simple process
- b) an orphan process
- c) a daemon
- d) a zombie process

Answer: c

51. Which of the following command will remove the current crontab?

- a) crontab -p
- b) crontab -l
- c) crontab -e
- d) crontab -r

Answer: d

52. To find out how efficiently a program a used the system resources, which command is used?

- a) sys
- b) time
- c) crontab
- d) daemon

Answer: b

53. Which one of the following is an internal command?

- a) cut
- b) expr
- c) set
- d) ls

Answer: c

**UNIT-III**

**USER MANAGEMENT AND THE FILE SYSTEM**

**SYLLABUS**

Types of Users, Creating users, Granting rights User management commands, File quota and various file systems available, File System Management and Layout, File permissions, Login process, Managing Disk Quotas, Links (hard links, symbolic links)

**Types of Users:**

Linux is a multi-user operating system i.e., it allows multiple users on different computers or terminals to access a single system. This makes it mandatory to know how to perform effective user management; how to add, modify, suspend, or delete user accounts, along with granting them the necessary permissions to do their assigned tasks. For this multi-user design to work properly there needs to be a method to enforce concurrency control. This is where permissions come in to play.

Normally Linux/Unix based systems have two user accounts;

**a general user account,**

**and the root account,** which is the super user that can access everything on the machine, make system changes, and administer other users. Some variants of Linux work a little differently though.

**Creating Users:**

To add/create a new user, all you've to follow the command 'useradd' or 'adduser' with 'username'. The 'username' is a user login name, that is used by user to login into the system.

Only one user can be added and that username must be unique (different from other username already exists on the system).

For example, to add a new user called 'tecmint', use the following command.

```
[root@tecmint ~]# useraddtecmint
```

When we add a new user in Linux with 'useradd' command it gets created in locked state and to unlock that user account, we need to set a password for that account with 'passwd' command.

```
[root@tecmint ~]# passwdtecmint
```

Changing passwo



```
rd for user tecmint.
```

```
New UNIX password:
```

```
Retype new UNIX password:
```

```
passwd: all authentication tokens updated successfully.
```

Once a new user created, it's entry automatically added to the '/etc/passwd' file. The file is used to store users information and the entry should be.

```
tecmint:x:504:504:tecmint:/home/tecmint:/bin/bash
```

The above entry contains a set of seven colon-separated fields, each field has it's own meaning. Let's see what are these fields:

1. Username: User login name used to login into system. It should be between 1 to 32 characters long.
2. Password: User password (or x character) stored in /etc/shadow file in encrypted format.
3. User ID (UID): Every user must have a User ID (UID) User Identification Number. By default UID 0 is reserved for root user and UID's ranging from 1-99 are reserved for other predefined accounts. Further UID's ranging from 100-999 are reserved for system accounts and groups.
4. Group ID (GID): The primary Group ID (GID) Group Identification Number stored in /etc/group file.
5. User Info: This field is optional and allow you to define extra information about the user. For example, user full name. This field is filled by 'finger' command.
6. Home Directory: The absolute location of user's home directory.
7. Shell: The absolute location of a user's shell i.e. /bin/bash.

## **2. Create a User with Different Home Directory**

By default 'useradd' command creates a user's home directory under /home directory with username. Thus, for example, we've seen above the default home directory for the user 'tecmint' is '/home/tecmint'.

However, this action can be changed by using '-d' option along with the location of new home directory (i.e. /data/projects). For example, the following command will create a user 'anusha' with a home directory '/data/projects'.

```
[root@tecmint ~]# useradd -d /data/projects anusha
```

You can see the user home directory and other user related information like user id, group id, shell and comments.

```
[root@tecmint ~]# cat /etc/passwd | grep anusha
```

```
anusha:x:505:505::/data/projects:/bin/bash
```

### **3. Create a User with Specific User ID**

In Linux, every user has its own UID (Unique Identification Number). By default, whenever we create a new user accounts in Linux, it assigns userid 500, 501, 502 and so on... But, we can create user's with custom userid with '-u' option. For example, the following command will create a user 'navin' with custom userid '999'.

```
[root@tecmint ~]# useradd -u 999 navin
```

Now, let's verify that the user created with a defined userid (999) using following command.

```
[root@tecmint ~]# cat /etc/passwd | grep navin
```

```
navin:x:999:999::/home/navin:/bin/bash
```

**NOTE:** Make sure the value of a user ID must be unique from any other already created users on the system.

### **4. Create a User with Specific Group ID**

Similarly, every user has its own GID (Group Identification Number). We can create users with specific group ID's as well with -g option.

Here in this example, we will add a user 'tarunika' with a specific UID and GID simultaneously with the help of '-u' and '-g' options.

```
[root@tecmint ~]# useradd -u 1000 -g 500 tarunika
```

Now, see the assigned user id and group id in ‘/etc/passwd’ file.

```
[root@tecmint ~]# cat /etc/passwd | greptarunika
```

```
tarunika:x:1000:500::/home/tarunika:/bin/bash
```

### **Granting rights User management commands**

To add a new user account, you can run either of the following two commands as root.

```
# adduser [new_account]
```

```
# useradd [new_account]
```

When a new user account is added to the system, the following operations are performed.

1. His/her home directory is created (/home/username by default).
2. The following hidden files are copied into the user’s home directory, and will be used to provide environment variables for his/her user session.

.bash\_logout

.bash\_profile

.bashrc

3. A mail spool is created for the user at /var/spool/mail/username.

4. A group is created and given the same name as the new user account.

#### Understanding /etc/passwd

The full account information is stored in the /etc/passwd file. This file contains a record per system user account and has the following format (fields are delimited by a colon).

[username]:[x]:[UID]:[GID]:[Comment]:[Home directory]:[Default shell]

Fields [username] and [Comment] are self explanatory.

The x in the second field indicates that the account is protected by a shadowed password (in /etc/shadow), which is needed to logon as [username].

The [UID] and [GID] fields are integers that represent the User IDentification and the primary Group IDentification to which [username] belongs, respectively.

The [Home directory] indicates the absolute path to [username]'s home directory, and

The [Default shell] is the shell that will be made available to this user when he or she logs the system.

#### Understanding /etc/group

Group information is stored in the /etc/group file. Each record has the following format.

[Group name]:[Group password]:[GID]:[Group members]

[Group name] is the name of group.

An x in [Group password] indicates group passwords are not being used.

[GID]: same as in /etc/passwd.

[Group members]: a comma separated list of users who are members of [Group name].

#### Add User Accounts in Linux

After adding an account, you can edit the following information (to name a few fields) using the usermod command, whose basic syntax of usermod is as follows.

```
# usermod [options] [username]
```

Setting the expiry date for an account

Use the `--expiredate` flag followed by a date in YYYY-MM-DD format.

```
# usermod --expiredate 2014-10-30 tecmint
```

Adding the user to supplementary groups

Use the combined `-aG`, or `--append --groups` options, followed by a comma separated list of groups.

```
# usermod --append --groups root,users tecmint
```

Changing the default location of the user's home directory

Use the `-d`, or `--home` options, followed by the absolute path to the new home directory.

```
# usermod --home /tmp tecmint
```

Changing the shell the user will use by default

Use `--shell`, followed by the path to the new shell.

```
# usermod --shell /bin/sh tecmint
```

Displaying the groups an user is a member of

```
# groups tecmint
```

```
# id tecmint
```

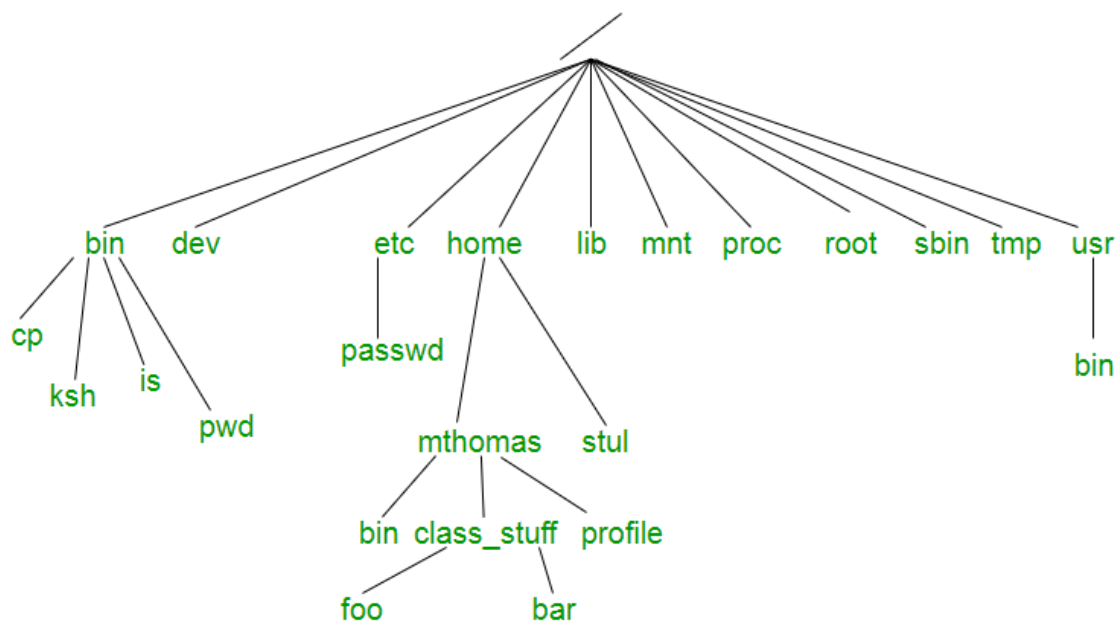
Now let's execute all the above commands in one go.

```
# usermod --expiredate 2014-10-30 --append --groups root,user
```

## **Unix File System**

Unix file system is a logical method of organizing and storing large amounts of information in a way that makes it easy to manage. A file is a smallest unit in which the information is stored. Unix file system has several important features. All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the file system.

Files in Unix System are organized into multi-level hierarchy structure known as a directory tree. At the very top of the file system is a directory called “root” which is represented by a “/”. All other files are “descendants” of root.



Directories or Files and their description –

/ : The slash / character alone denotes the root of the filesystem tree.

/bin : Stands for “binaries” and contains certain fundamental utilities, such as ls or cp, which are generally needed by all users.

/boot : Contains all the files that are required for successful booting process.

**KARPAGAM ACADEMY OF HIGHER EDUCATION**

**CLASS: III B.Sc CS**

**COURSE CODE: UNIT II: SYSTEM PROCESSES (AN OVERVIEW) BATCH: 2017-2020**

**/dev** : Stands for “devices”. Contains file representations of peripheral devices and pseudo-devices.

**/etc** : Contains system-wide configuration files and system databases. Originally also contained “dangerous maintenance utilities” such as `init`, but these have typically been moved to `/sbin` or elsewhere.

**/home** : Contains the home directories for the users.

**/lib** : Contains system libraries, and some critical files such as kernel modules or device drivers.

**/media** : Default mount point for removable devices, such as USB sticks, media players, etc.

**/mnt** : Stands for “mount”. Contains filesystem mount points. These are used, for example, if the system uses multiple hard disks or hard disk partitions. It is also often used for remote (network) filesystems, CD-ROM/DVD drives, and so on.

**/proc** : `procfs` virtual filesystem showing information about processes as files.

**/root** : The home directory for the superuser “root” – that is, the system administrator. This account’s home directory is usually on the initial filesystem, and hence not in `/home` (which may be a mount point for another filesystem) in case specific maintenance needs to be performed, during which other filesystems are not available. Such a case could occur, for example, if a hard disk drive suffers physical failures and cannot be properly mounted.

**/tmp** : A place for temporary files. Many systems clear this directory upon startup; it might have `tmpfs` mounted atop it, in which case its contents do not survive a reboot, or it might be explicitly cleared by a startup script at boot time.

**/usr** : Originally the directory holding user home directories, its use has changed. It now holds executables, libraries, and shared resources that are not system critical, like the X Window System, KDE, Perl, etc. However, on some Unix systems, some user accounts may still have a home directory that is a direct subdirectory of `/usr`, such as the default as in Minix. (on modern systems, these user accounts are often related to server or system use, and not directly used by a person).

**/usr/bin** : This directory stores all binary programs distributed with the operating system not residing in `/bin`, `/sbin` or (rarely) `/etc`.

**/usr/include** : Stores the development headers used throughout the system. Header files are mostly used by the `#include` directive in C/C++ programming language.

/usr/lib : Stores the required libraries and data files for programs stored within /usr or elsewhere.

/var : A short for “variable.” A place for files that may change often – especially in size, for example e-mail sent to users on the system, or process-ID lock files.

/var/log : Contains system log files.

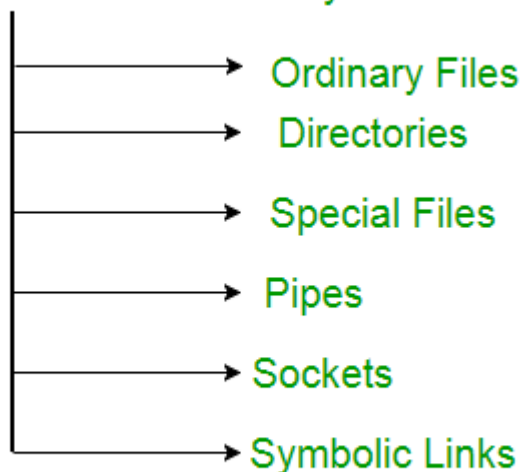
/var/mail : The place where all the incoming mails are stored. Users (other than root) can access their own mail only. Often, this directory is a symbolic link to /var/spool/mail.

/var/spool : Spool directory. Contains print jobs, mail spools and other queued tasks.

/var/tmp : A place for temporary files which should be preserved between system reboots.

Types of Unix files – The UNIX files system contains several different types of files :

### Classification of Unix File System :



**1. Ordinary files** – An ordinary file is a file on the system that contains data, text, or program instructions.

Used to store your information, such as some text you have written or an image you have drawn. This is the type of file that you usually work with.

Always located within/under a directory file.

Do not contain other files.



In long-format output of `ls -l`, this type of file is specified by the “-” symbol.

**2. Directories** – Directories store both special and ordinary files. For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders. A directory file contains an entry for every file and subdirectory that it houses. If you have 10 files in a directory, there will be 10 entries in the directory. Each entry has two components.

(1) The Filename

(2) A unique identification number for the file or directory (called the inode number)

Branching points in the hierarchical tree.

Used to organize groups of files.

May contain ordinary files, special files or other directories.

Never contain “real” information which you would work with (such as text). Basically, just used for organizing files.

All files are descendants of the root directory, ( named / ) located at the top of the tree.

In long-format output of `ls -l` , this type of file is specified by the “d” symbol.

**3. Special Files** – Used to represent a real physical device such as a printer, tape drive or terminal, used for Input/Output (I/O) operations. Device or special files are used for device Input/Output(I/O) on UNIX and Linux systems. They appear in a file system just like an ordinary file or a directory.

On UNIX systems there are two flavors of special files for each device, character special files and block special files :

When a character special file is used for device Input/Output(I/O), data is transferred one character at a time. This type of access is called raw device access.

When a block special file is used for device Input/Output(I/O), data is transferred in large fixed-size blocks. This type of access is called block device access.

For terminal devices, it’s one character at a time. For disk devices though, raw access means reading or writing in whole chunks of data – blocks, which are native to your disk.

In long-format output of `ls -l`, character special files are marked by the “c” symbol.

In long-format output of `ls -l`, block special files are marked by the “b” symbol.

**4. Pipes** – UNIX allows you to link commands together using a pipe. The pipe acts a temporary file which only exists to hold data from one command until it is read by another. A Unix pipe provides a one-way flow of data. The output or result of the first command sequence is used as the input to the second command sequence. To make a pipe, put a vertical bar (|) on the command line between two commands. For example: `who | wc -l`

In long-format output of `ls -l`, named pipes are marked by the “p” symbol.

**5. Sockets** – A Unix socket (or Inter-process communication socket) is a special file which allows for advanced inter-process communication. A Unix Socket is used in a client-server application framework. In essence, it is a stream of data, very similar to network stream (and network sockets), but all the transactions are local to the filesystem.

In long-format output of `ls -l`, Unix sockets are marked by “s” symbol.

**6. Symbolic Link** – Symbolic link is used for referencing some other file of the file system. Symbolic link is also known as Soft link. It contains a text form of the path to the file it references. To an end user, symbolic link will appear to have its own name, but when you try reading or writing data to this file, it will instead reference these operations to the file it points to. If we delete the soft link itself, the data file would still be there. If we delete the source file or move it to a different location, symbolic file will not function properly.

In long-format output of `ls -l`, Symbolic link are marked by the “l” symbol (that’s a lower case L).

## **Linux File System Layout**

The Linux File System is very important to understand.

On top of all the file system there is a directory called root (/). The root file system is the standing point of everything here.

So in the root directory, there are couple of default directory you can see always.

Those file systems are,

/ – root directory

/boot – Which contains everything your system needs to start up.

/usr – This is very important directory where the program file contents are stored

/etc – The directory which contains all the configuration files.

/home – The directory contains users individual files.

/mnt – Used to mount the cd/dvd

### **File System Management**

When migrating from another operating system such as Microsoft Windows to another; one thing that will profoundly affect the end user greatly will be the differences between the file systems.

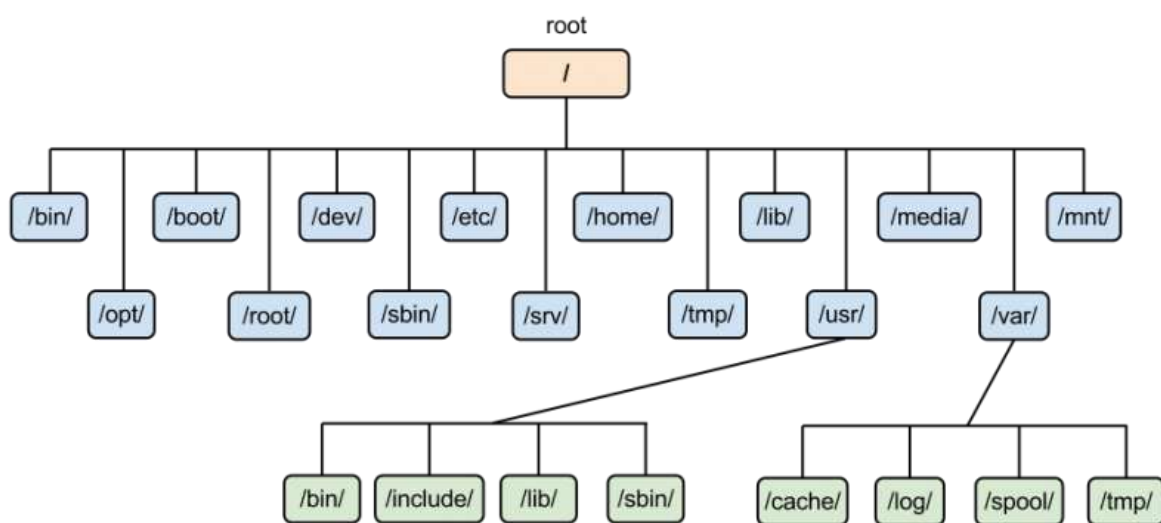
A filesystem is the methods and data structures that an operating system uses to keep track of files on a disk or partition; that is, the way the files are organized on the disk. The word is also used to refer to a partition or disk that is used to store the files or the type of the filesystem. Thus, one might say I have two filesystems meaning one has two partitions on which one stores files, or that one is using the extended filesystem, meaning the type of the filesystem.

The difference between a disk or partition and the filesystem it contains is important. A few programs (including, reasonably enough, programs that create filesystems) operate directly on the raw sectors of a disk or partition; if there is an existing file system there it will be destroyed or seriously corrupted. Most programs operate on a filesystem, and therefore won't work on a partition that doesn't contain one (or that contains one of the wrong type).

Before a partition or disk can be used as a filesystem, it needs to be initialized, and the bookkeeping data structures need to be written to the disk. This process is called making a filesystem.

Most UNIX filesystem types have a similar general structure, although the exact details vary quite a bit. The central concepts are superblock, inode, data block, directory block, and indirection block. The superblock contains information about the filesystem as a whole, such as its size (the exact information here depends on the filesystem). An inode contains all information about a file, except its name. The name is stored in the directory, together with the number of the inode. A directory entry consists of a filename and the number of the inode which represents the file. The inode contains the numbers of several data blocks, which are used to store the data in the file. There is space only for a few data block numbers in the inode, however, and if more are needed, more space for pointers to the data blocks is allocated dynamically. These dynamically allocated blocks are indirect blocks; the name indicates that in order to find the data block, one has to find its number in the indirect block first.

Like UNIX, Linux chooses to have a single hierarchical directory structure. Everything starts from the root directory, represented by /, and then expands into sub-directories instead of having so-called 'drives'. In the Windows environment, one may put one's files almost anywhere: on C drive, D drive, E drive, etc. Such a file system is called a hierarchical structure and is managed by the programs themselves (program directories), not by the operating system. On the other hand, Linux sorts directories descending from the root directory / according to their importance to the boot process.



If you're wondering why Linux uses the frontslash / instead of the backslash \ as in Windows it's because it's simply following the UNIX tradition. Linux, like Unix also chooses to be case

sensitive. What this means is that the case, whether in capitals or not, of the characters, becomes very important. So this is not the same as THIS. This feature accounts for a fairly large proportion of problems for new users especially during file transfer operations whether it may be via removable disk media such as a floppy disk or over the wire by way of FTP.

The filesystem order is specific to the function of a file and not to its program context (the majority of Linux filesystems are 'Second Extended File Systems', short 'EXT2' (aka 'ext2fs' or 'extfs2') or are themselves subsets of this filesystem such as ext3 and Reiserfs). It is within this filesystem that the operating system determines into which directories programs store their files.

If you install a program in Windows, it usually stores most of its files in its own directory structure. A help file for instance may be in C:\Program Files\[program name]\ or in C:\Program Files\[program-name]\help or in C:\Program Files\[program - name]\humpty\dummy\doo. In Linux, programs put their documentation into /usr/share/doc/[program-name], man(ual) pages into /usr/share/man/man[1-9] and info pages into /usr/share/info. They are merged into and with the system hierarchy.

Unless you mount a partition or a device, the system does not know of the existence of that partition or device. This might not appear to be the easiest way to provide access to your partitions or devices, however, it offers the advantage of far greater flexibility when compared to other operating systems. This kind of layout, known as the unified filesystem, does offer several advantages over the approach that Windows uses. Let's take the example of the /usr directory. This sub-directory of the root directory contains most of the system executables. With the Linux filesystem, you can choose to mount it off another partition or even off another machine over the network using an innumerable set of protocols such as NFS (Sun), Coda (CMU) or AFS (IBM). The underlying system will not and need not know the difference. The presence of the /usr directory is completely transparent. It appears to be a local directory that is part of the local directory structure.

### **File Permissions in Linux/Unix with Example**

## **KARPAGAM ACADEMY OF HIGHER EDUCATION**

**CLASS: III B.Sc CS**

**COURSE CODE: UNIT II: SYSTEM PROCESSES (AN OVERVIEW) BATCH: 2017-2020**

Linux is a clone of UNIX, the multi-user operating system which can be accessed by many users simultaneously. Linux can also be used in mainframes and servers without any modifications. But this raises security concerns as an unsolicited or malign user can corrupt, change or remove crucial data. For effective security, Linux divides authorization into 2 levels.

- (i) Ownership
- (ii) Permission

The concept of permissions and ownership is crucial in Linux

### **Ownership of Linux files**

Every file and directory on your Unix/Linux system is assigned 3 types of owner, given below.

#### **User**

A user is the owner of the file. By default, the person who created a file becomes its owner. Hence, a user is also sometimes called an owner.

#### **Group**

A user- group can contain multiple users. All users belonging to a group will have the same access permissions to the file. Suppose you have a project where a number of people require access to a file. Instead of manually assigning permissions to each user, you could add all users to a group, and assign group permission to file such that only this group members and no one else can read or modify the files.

#### **Other**

Any other user who has access to a file. This person has neither created the file, nor he belongs to a usergroup who could own the file. Practically, it means everybody else. Hence, when you set the permission for others, it is also referred as set permissions for the world.

Now, the big question arises how does Linux distinguish between these three user types so that a user 'A' cannot affect a file which contains some other user 'B's' vital information/data. It is like you do not want your colleague, who works on your Linux computer, to view your images. This is where Permissions set in, and they define user behavior.

Every file and directory in your UNIX/Linux system has following 3 permissions defined for all the 3 owners discussed above.

**Read:** This permission give you the authority to open and read a file. Read permission on a directory gives you the ability to lists its content.

**Write:** The write permission gives you the authority to modify the contents of a file. The write permission on a directory gives you the authority to add, remove and rename files stored in the directory. Consider a scenario where you have to write permission on file but do not have write permission on the directory where the file is stored. You will be able to modify the file contents. But you will not be able to rename, move or remove the file from the directory.

**Execute:** In Windows, an executable program usually has an extension ".exe" and which you can easily run. In Unix/Linux, you cannot run a program unless the execute permission is set. If the execute permission is not set, you might still be able to see/modify the program code(provided read & write permissions are set), but not run it.

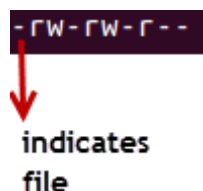
**ls - l** on terminal gives

```
ls - l
```

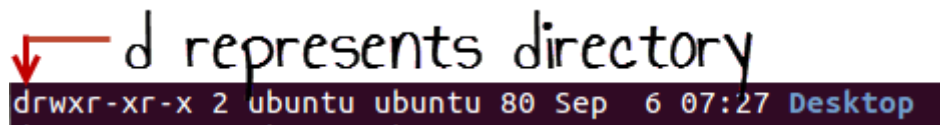


Here, we have highlighted '-rw-rw-r--' and this weird looking code is the one that tells us about the permissions given to the owner, user group and the world.

Here, the first '-' implies that we have selected a file.p>



Else, if it were a directory, **d** would have been shown.



The characters are pretty easy to remember.

**r** = read permission

**w** = write permission

**x** = execute permission

**-** = no permission

## **Login Process in Linux:**

### **Login process steps**

- (i) Init starts getty process
- (ii) getty process initiates login prompt on terminal
- (iii) login command check user credentials from /etc/passwd
- (iv) getty starts user shell process
- (v) shell reads the system wide files /etc/profile, /etc/bashrc
- (vi) Shell reads user specific files .profile, .login
- (vii) Now it reads shell specific configuration file .bashrc
- (viii) Shell displays the default prompt

login is used when signing onto a system. It can also be used to switch from one user to another at any time.

### **NAME**

login - sign on

### **SYNOPSIS**

login [ name ]

login -p

login -h hostname

login -f name

### **DESCRIPTION**

login is used when signing onto a system. It can also be used to switch from one user to another at any time (most modern shells have support for this feature built into them, however).

If an argument is not given, login prompts for the username.



## **KARPAGAM ACADEMY OF HIGHER EDUCATION**

**CLASS: III B.Sc CS**

**COURSE CODE: UNIT II: SYSTEM PROCESSES (AN OVERVIEW) BATCH: 2017-2020**

If the user is not root, and if /etc/nologin exists, the contents of this file are printed to the screen, and the login is terminated. This is typically used to prevent logins when the system is being taken down.

If special access restrictions are specified for the user in /etc/usertty, these must be met, or the log in attempt will be denied and a syslog message will be generated. See the section on "Special Access Restrictions".

If the user is root, then the login must be occurring on a tty listed in /etc/securetty. Failures will be logged with the syslog facility.

After these conditions have been checked, the password will be requested and checked (if a password is required for this username). Ten attempts are allowed before login dies, but after the first three, the response starts to get very slow. Login failures are reported via the syslog facility. This facility is also used to report any successful root logins.

If the file .hushlogin exists, then a "quiet" login is performed (this disables the checking of mail and the printing of the last login time and message of the day). Otherwise, if /var/log/lastlog exists, the last login time is printed (and the current login is recorded).

Random administrative things, such as setting the UID and GID of the tty are performed. The TERM environment variable is preserved, if it exists (other environment variables are preserved if the -p option is used). Then the HOME, PATH, SHELL, TERM, MAIL, and LOGNAME environment variables are set. PATH defaults to /usr/local/bin:/bin:/usr/bin for normal users, and to /usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin for root. Last, if this is not a "quiet" login, the message of the day is printed and the file with the user's name in /var/spool/mail will be checked, and a message printed if it has non-zero length.

The user's shell is then started. If no shell is specified for the user in /etc/passwd, then /bin/sh is used. If there is no directory specified in /etc/passwd, then / is used (the home directory is checked for the .hushlogin file described above).

### **OPTIONS**

Tag	Description
-----	-------------

-p	Used by getty(8) to tell login not to destroy the environment
----	---

-f	Used to skip a second login authentication. This specifically does not work for root, and does not appear to work well under Linux.
----	---

-h Used by other servers (i.e., telnetd(8)) to pass the name of the remote host to login so that it may be placed in utmp and wtmp. Only the superuser may use this option.

## **Soft (Symbolic) and Hard links in Unix/Linux**

A link in UNIX is a pointer to a file. Like pointers in any programming languages, links in UNIX are pointers pointing to a file or a directory. Creating links is a kind of shortcuts to access a file. Links allow more than one file name to refer to the same file, elsewhere.

There are two types of links :

Soft Link or Symbolic links

Hard Links

These links behave differently when the source of the link (what is being linked to) is moved or removed. Symbolic links are not updated (they merely contain a string which is the pathname of its target); hard links always refer to the source, even if moved or removed.

For example, if we have a file a.txt. If we create a hard link to the file and then delete the file, we can still access the file using hard link. But if we create a soft link of the file and then delete the file, we can't access the file through soft link and soft link becomes dangling. Basically hard link increases reference count of a location while soft links work as a shortcut (like in Windows)

### **1. Hard Links**

- Each hard linked file is assigned the same Inode value as the original, therefore they reference the same physical file location. Hard links more flexible and remain linked even if the original or linked files are moved throughout the file system, although hard links are unable to cross different file systems.
- `ls -l` command shows all the links with the link column shows number of links.
- Links have actual file contents
- Removing any link, just reduces the link count, but doesn't affect other links.
- We cannot create a hard link for a directory to avoid recursive loops.
- If original file is removed then the link will still show the content of the file.
- Command to create a hard link is:
- `$ ln [original filename] [link name]`

### **2. Soft Links**

- A soft link is similar to the file shortcut feature which is used in Windows Operating systems. Each soft linked file contains a separate Inode value that points to the original file. As similar to hard links, any changes to the data in either file is reflected in the other. Soft links can be linked across different file systems, although if the original file is deleted or moved, the soft linked file will not work correctly (called hanging link).
- `ls -l` command shows all links with first column value `l` and the link points to original file.
- Soft Link contains the path for original file and not the contents.

- Removing soft link doesn't affect anything but removing original file, the link becomes "dangling" link which points to nonexistent file.
- A soft link can link to a directory.
- Link across filesystems: If you want to link files across the filesystems, you can only use symlinks/soft links.
- Command to create a Soft link is:  

```
$ ln -s [original filename] [link name]
```

### **Managing Disk Quotas:**

**Disk Quotas:** This feature of Linux allows the system administrator to allocate a maximum amount of disk space a user or group may use. It can be flexible in its adherence to the rules assigned and is applied per filesystem. The default Linux Kernel which comes with Redhat and Fedora Core comes with quota support compiled in.

**Two versions of quotas have been released.** Version 2 is used by the Linux 2.4 and 2.6 kernel. Quotas version 1 is used by the Linux 2.2

Configuration of disk usage quotas on Linux - Perform the following as root:

1. Edit file /etc/fstab to add qualifier "usrquota" or "grpquota" to the partition. The following file system mounting options can be specified in /etc/fstab: grpquota, noquota, quota and usrquota.

The filesystem when mounted will show up in the file /etc/mtab, the list of all currently mounted filesystems.

To enable user quota support on a file system, add "usrquota" to the fourth field containing the word "defaults".

...

```
/dev/hda2 /home ext3 defaults,usrquota 1 1
```

...

Replace "usrquota" with "grpquota", should you need group quota support on a file system

...

```
/dev/hda2 /home ext3 defaults,grpquota 1 1
```

...

Need both user quota and group quota support on a file system?

...

**KARPAGAM ACADEMY OF HIGHER EDUCATION**

**CLASS: III B.Sc CS**

**COURSE CODE: UNIT II: SYSTEM PROCESSES (AN OVERVIEW) BATCH: 2017-2020**

```
/dev/hda2 /home ext3 defaults,usrquota,grpquota 1 1
```

...

This enables user and group quotas support on the /home file system.

```
touch /partition/aquota.user
```

where the partition might be /home or some partition defined in /etc/fstab.

then

```
chmod 600 /partition/aquota.user
```

The file should be owned by root. Quotas may also be set for groups by using the file aquota.group

**Quota file names:**

Quota Version 2 (Linux 2.4/2.6 kernel: Red Hat 7.1+/8/9,FC 1-3): aquota.user, aquota.group

Quota Version 1 (Linux 2.2 kernel: Red Hat 6, 7.0): quota.user, quota.group

The files can be converted/upgraded using the **convertquota** command.

Re-boot or re-mount file partition with quotas.

Re-boot: shutdown -r now

Re-mount partition: mount -o remount /partition

After re-booting or re-mounting the file system, the partition will show up in the list of mounted filesystems as having quotas. Check /etc/mtab:

...

```
/dev/hda5 / ext3 rw,usrquota 0 0
```

...

### UNIT – III

1. Which library function is used for printing error messages?

- a) strerror
- b) stderr
- c) strer
- d) ptrerror

Answer: a

2. What is the significance of errno 13?

- a) not a super user
- b) no such file and directory
- c) permission denied
- d) no space left on device

Answer: b

3. What is the symbolic constant for errno 2?

- a) EPERM
- b) ENDENT
- c) EIO
- d) EEXIST

Answer: b

4. Which of the following is not a valid symbolic constant?

- a) EPERM
- b) ENDENT
- c) EIOR
- d) EEXIST

Answer: c

5. How many data structures does the kernel maintain in memory that contain information about an open file?

- a) 3
- b) 2
- c) 5
- d) 1

Answer: a

6. The file table contains \_\_\_\_\_

- a) mode of opening
- b) status flags
- c) offset pointer, reference count
- d) mode of opening, status flags, offset printer and reference count

Answer: d

7. The vnode table is also called \_\_\_\_\_

- a) inode table
- b) file table
- c) vtable
- d) vtable

Answer: a

8. Which of the following system calls performs the action of cd command?

- a) chdir
- b) fchdir
- c) mkdir
- d) chdir and fchdir

Answer: d

9. For creating and removing directories, which of the following system calls are used?

- a) mkdir
- b) rmdir
- c) chdir
- d) mkdir and rmdir

Answer: d

10. A super user can use \_\_\_\_ call for creating a directory.

- a) mknod
- b) mknod
- c) rmdir
- d) chdir

Answer: a

11. For creating a hard and symbolic link, which system calls are used?

- a) link, unlink
- b) link, symlink
- c) unlink, ulink
- d) hlink, slink

Answer: b

12. For removing a link, \_\_\_\_ is used.

- a) link
- b) symlink
- c) unlink
- d) delink

Answer: c

13. \_\_\_\_ system call is used for renaming a file, directory or symbolic link.

- a) renam
- b) rename
- c) ren
- d) change

Answer: b

14. Which of the following macro returns true if the file type is a directory?

- a) S\_ISREG
- b) S\_ISDIR
- c) S\_ISCHR
- d) S\_ISFIFO

Answer: b

15. For checking a file access rights, \_\_\_\_ system call is used.

- a) acc
- b) access

- c) axs
- d) filert

Answer: b

16. Which of the following is used with access call for checking the owner's permissions?

- a) R\_OK
- b) W\_OK
- c) X\_OK
- d) R\_OK, W\_OK, X\_OK

Answer: d

17. Both the owner and group owner can be changed by \_\_\_\_ call.

- a) chown
- b) chgrp
- c) chuser
- d) ch

Answer: a

18. For changing the time stamps, \_\_\_\_\_ system call is invoked.

- a) atime
- b) utime
- c) mtime
- d) ch

Answer: b

19. Which of the following system call uses file descriptor as an argument?

- a) read
- b) write
- c) close
- d) read, write, close

Answer: d

20. Which of the following system call is used for closing a file?

- a) open
- b) lseek
- c) close
- d) write

Answer: c

21. close system call returns \_\_\_\_

- a) 0
- b) -1
- c) 1
- d) 0 and -1

Answer: d

22. \_\_\_\_ system call is used for writing to a file.

- a) read
- b) write
- c) close
- d) seek

Answer: b

23. write system call returns -1 when \_\_\_\_\_

- a) if disk fills up while write is in progress
- b) when file doesn't exist
- c) if the file size exceeds the system's limit
- d) if disk fills up while write is in progress and if the file size exceeds

Answer: d

24. \_\_\_\_ system call is used for positioning the offset pointer.

- a) read
- b) write
- c) open
- d) lseek

Answer: d

25. Which of the following offset is used with lseek system call to set the offset pointer to the end of the file?

- a) SEEK\_SET
- b) SEEK\_END
- c) SEEK\_CUR
- d) SEEK\_CR

Answer: b

26. Which of the following system call is used for truncating a file?

- a) truncate
- b) ftruncate
- c) trunk
- d) truncate and ftruncate

Answer: d

27. truncate needs the \_\_\_\_ of the file as an argument but ftruncate works with \_\_\_\_\_

- a) pathname, file descriptor
- b) file descriptor, pathname
- c) pathname, pathname
- d) file descriptor, file descriptor

Answer: a

28.



## UNIT IV NOTES

### SYLLABUS

**Shell introduction and Shell Scripting** What is shell and various type of shell, Various editors present in Linux Different modes of operation in vi editor, What is shell script, Writing and executing the shell script , Shell variable (user defined and system variables)

#### What is shell ?

**Shell** is a UNIX term for the interactive user interface with an operating system.

The **shell** is the layer of programming that understands and executes the commands a user enters. In some systems, the **shell** is called a command interpreter.

#### Types of Shells

Let's study different kind of shells with their features, functionalities and speed of executions.

#### Bash Shell

Bash is a Unix shell. It was created as a substitute for Bourne shell and include much more scripting tools than Bourne shell like the csh and ksh shells.

Bash is a very common shell and you actually might be running it by default on your machine. It is almost always available on all Linux distributions. One of the contender to Bash shell is dash which is becoming more popular by the Ubuntu project.

#### Zsh Shell

Zsh shell is 100% compatible with bash. This means that whatever scripts run on Bash runs on Zsh shell exactly the same. To add, Zsh shell includes more features.

Most common features in Zsh shell are spelling correction, intelligent command-line completion, pluggable modules that increase shell capabilities, aliases with global access that allow a user to alias file names or anything else instead of just commands and much better theming support.

Even better feature in favor of Zsh shell is that if a user is known to Bash shell, it is very easy to switch to Zsh shell without getting used to a different syntax.

## Csh Shell

Bill Joy created it at the University of California at Berkeley. It incorporated features such as aliases and command history. It includes helpful programming features like built-in arithmetic and C-like expression syntax.

Csh is an improved C shell. It is most popular in terms of a login shell and shell command interpreter. Most favorable features of this shell are:

- Syntax similar to C
- Control over jobs
- Intelligent spell correction
- Command-line editor
- Filename completion

## The Bourne Shell –

Denoted as **sh**

It was written by Steve Bourne at AT&T Bell Labs. It is the original UNIX shell. It is faster and more preferred. It lacks features for interactive use like the ability to recall previous commands. It also lacks built-in arithmetic and logical expression handling. It is default shell for Solaris OS.

For the Bourne shell the:

Command full-path name is /bin/sh and /sbin/sh,

Non-root user default prompt is \$,

Root user default prompt is #.

## The Korn Shell

It is denoted as **ksh**

It Was written by David Korn at AT&T Bell LabsIt is a superset of the Bourne shell.So it supports everything in the Bourne shell.It has interactive features. It includes features like built-in arithmetic and C-like arrays, functions, and string-manipulation facilities.It is faster than C shell. It is compatible with script written for C shell.

## VARIOUS EDITORS PRESENT IN LINUX

In this article I am taking a look at some of the best 12 open source commonly used text editors in Linux on both server and desktops.

### 1. Vi/Vim Editor

---

Vim is a powerful command line based text editor that has enhanced the functionalities of the old Unix Vi text editor. It is one the most popular and widely used text editors among System Administrators and programmers that is why many users often refer to it as a programmer's editor. It enables syntax highlighting when writing code or editing configuration files.

If you want to see our complete series on vi(m), please refer the links below:

- [Learn and Use Vi/Vim as a Full Text Editor in Linux](#)
- [Learn 'Vi/Vim' Editor Tips and Tricks to Enhance Your Skills](#)
- [8 Interesting 'Vi/Vim' Editor Tips and Tricks](#)

### 2. Gedit

---

This is a general purpose GUI based text editor and is installed by default text editor on Gnome desktop environment. It is simple to use, highly pluggable and a powerful editor with the following features:

- Support for UTF-8
- Use of configurable font size and colors
- Highly customizable syntax highlighting
- Undo and redo functionalities
- Reverting of files
- Remote editing of files
- Search and replace text
- Clipboard support functionalities and many more

### 3. Nano Editor

---

Nano is an easy to use text editor especially for both new and advanced Linux users. It enhances usability by providing customizable key binding.

Nano has the following features:

- Highly customizable key bindings
- Syntax highlighting
- Undo and redo options
- Full line display on the standard output
- Pager support to read from standard input

You can check our complete guide for editing files with Nano editor at:

- [How to Use Nano Editor in Linux](#)

#### 4. GNU Emacs

---

This is a highly extensible and customizable text editor that also offers interpretation of the Lisp programming language at its core. Different extensions can be added to support text editing functionalities.

Emacs has the following features:

- User documentation and tutorials
- Syntax highlighting using colors even for plain text.
- Unicode supports many natural languages.
- Various extension including mail and news, debugger interface, calendar and many more

#### 5. Kate/Kwrite

---

Kate is a feature rich and highly pluggable text editor that comes with KDesktop Environment (KDE). The Kate project aims at development of two main products that is: KatePart and Kate.

KatePart is an advanced text editor component included in many KDE applications which may require users to edit text whereas Kate is an multiple document interface(MDI) text editor.

The following are some of its general features:

- Extensible through scripting
- Encoding support such as unicode mode
- Text rendering in bi-directional mode
- Line ending support with auto detection functionalities

Also remote file editing and many other features including advanced editor features, applications features, programming features, text highlighting features, backup features and search and replace features.

#### 6. Lime Text

---

This is a powerful IDE-like text editor which is free and open-source successor of popular Sublime Text. It has a few frontends such as command-line interface that you can use with the pluggable backend.

#### 7. Pico Editor

---

Pico is also a command line based text editor that comes with the Pine news and email client. It is a good editor for new Linux users because of its simplicity in relation to many GUI text editors.

---

## Different modes of operation in vi editor

---

There are three modes of operation in vi: Command Mode, Input Mode, and Line Mode.

- **Command Mode:** When vi starts up, it is in Command Mode. This mode is where vi interprets any characters we type as commands and thus does not display them in the xterm window. This mode allows us to move through a file, and to delete, copy, or paste a piece of text. To enter into Command Mode from any other mode, it suffices to press the [Esc] key. If we press [Esc] when we are already in Command Mode, then vi will beep or flash the screen.
- **Input Mode:** In Input Mode, vi accepts keystrokes as text and displays the text as it is entered from the keyboard. vi must be in Input Mode before we can insert text into a file. To enter into Input Mode, we need to put vi into Command Mode and type the key [i].
- **Line Mode:** Line Mode is invoked by typing a colon [:] or a slash [/] while vi is in Command Mode. The cursor will jump to the last line of the screen and vi will wait for a command.

What is shell script?

A **shell script** is a text file that contains a sequence of commands for a UNIX-based operating system. It's called a **shell script** because it combines into a "**script**" in a single file a sequence of commands that would otherwise have to be presented to the system from a keyboard one at a time.

Steps to write and execute a script

### Steps to write and execute a script

- Open the terminal. Go to the directory where you want to create your script.
- Create a file with **.sh** extension.
- Write the script in the file using an editor.
- Make the script executable with command **chmod +x <fileName>**.
- Run the script using **./<fileName>**.

**Note:** In the last step you have to mention the path of the script if your script is in other directory.

## Writing and naming

A shell script is a sequence of commands for which you have a repeated use. This sequence is typically executed by entering the name of the script on the command line. Alternatively, you can use scripts to automate tasks using the cron facility. Another use for scripts is in the UNIX boot and shutdown procedure, where operation of daemons and services are defined in init scripts.

To create a shell script, open a new empty file in your editor. Any text editor will do: **vim**, **emacs**, **gedit**, **dtpad** et cetera are all valid. You might want to choose a more advanced editor like **vim** or **emacs**, however, because these can be configured to recognize shell and Bash syntax and can be a great help in preventing those errors that beginners frequently make, such as forgetting brackets and semi-colons.

In order to activate syntax highlighting in **vim**, use the command

**:syntax enable** or **:sy enable**

Or **:syn enable**

You can add this setting to your **.vimrc** file to make it permanent.

Put UNIX commands in the new empty file, like you would enter them on the command line. As discussed in the previous chapter (see [Section 1.3](#)), commands can be shell functions, shell built-ins, UNIX commands and other scripts.

Give your script a sensible name that gives a hint about what the script does. Make sure that your script name does not conflict with existing commands. In order to ensure that no confusion can arise, script names often end in **.sh**; even so, there might be other scripts on your system with the same name as the one you chose. Check using **which**, **whereis** and other commands for finding information about programs and files:

**which -a script\_name**

**whereis script\_name**

**locate script\_name**

## TYPES OF SHELL VARIABLES

### User Defined Variables in Linux

These variables are defined by **users**. A shell script allows us to set and use our **own variables** within the script. Setting variables allows you to **temporarily store data** and use it throughout the script, making the shell script more like a real computer program.

**User variables** can be any text string of up to **20 letters, digits, or an underscore character**. User variables are case sensitive, so the variable Var1 is different from the variable var1. This little rule often gets novice script programmers in trouble.

Values are assigned to user variables using an **equal sign**. No spaces can appear between the variable, the equal sign, and the value (another trouble spot for novices). Here are a few examples of assigning values to user variables:

```
var1=10
```

```
var2=-57
```

```
var3=testing
```

```
var4="still more testing"
```

The shell script **automatically determines the data type** used for the variable value. Variables defined within the shell script maintain their values throughout the life of the shell script but are deleted when the shell script completes.

### System Variables in Linux

Environment variables are dynamic values which affect the processes or programs on a computer. They exist in every operating system, but types may vary. Environment variables can be created, edited, saved, and deleted and give information about the system behavior.

Environment variables can change the way a software/programs behave.

E.g. \$LANG environment variable stores the value of the language that the user understands. This value is read by an application such that a Chinese user is shown a Mandarin interface while an American user is shown an English interface.

Prepared by Dr.M.Sakthi Asvini, Assistant Professor, CS,CA & IT

## UNIT IV

1. What is a shell script?

- a) group of commands
- b) a file containing special symbols
- c) a file containing a series of commands
- d) group of functions

Answer: c

2. The first line in any shell script begins with a \_\_\_\_\_

- a) &
- b) !
- c) \$
- d) #

Answer: d

3. To run the script, we should make it executable first by using \_\_\_\_\_

- a) chmod +x
- b) chmod +r
- c) chmod +w
- d) chmod +rwx

Answer: a

4. To spawn a child of our own choice for running the script, we can use \_\_\_\_ command.

- a) ps
- b) pr
- c) sh
- d) \$\$

Answer: c

5. Which command is used for making the scripts interactive?

- a) ip
- b) input
- c) read
- d) write

Answer: c

6. What are positional parameters?

- a) special variables for assigning arguments from the command line
- b) pattern matching parameters
- c) special variables for reading user input
- d) special variables and patterns

Answer: a

7. The first argument is read by the shell into the parameter \_\_\_\_

- a) 1\$
- b) \$3
- c) \$\$
- d) \$1

Answer: d



8. The complete set of positional parameters is stored in \_\_\_\_\_ as a single string.

- a) \$n
- b) \$#
- c) \$\*
- d) \$\$

Answer: c

9. Which of the following is used for storing the number of positional parameters?

- a) \$n
- b) \$#
- c) \$\*
- d) \$2

Answer: b

10. Which of the following commands let us perform a set of instructions repeatedly?

- a) for
- b) while
- c) until
- d) for, while, until

Answer: d

11. Which of the following keywords are used in while loop?

- a) do
- b) done
- c) then
- d) do and done

Answer: d

12. Which one of the following is used for looping with a list?

- a) while
- b) until
- c) case
- d) for

Answer: d

13. Which of the following loop statements uses do and done keyword?

- a) for
- b) while
- c) case
- d) for and while

Answer: d

14. Which command is used for changing filename extensions?

- a) chown
- b) rename
- c) basename
- d) rm

Answer: c

15. Which command is used by the shell for manipulating positional parameters?

- a) set
- b) cut

- c) case
- d) paste

Answer: a

16. \_\_\_\_ statement is used for shifting arguments left.

- a) set
- b) shift
- c) cut
- d) paste

Answer: b

17. The \_\_\_\_ allows us to read data from the same file containing the script.

- a) >>
- b) <<
- c) !!
- d) —

Answer: b

18. Which of the following command doesn't accept a filename as an argument?

- a) cut
- b) ls
- c) paste
- d) mailx

Answer: d

19. \_\_\_\_ command is the appropriate way to interrupt a program.

- a) kill
- b) SIGKILL
- c) INT
- d) trap

Answer: d

20. Which of the following option is used with a set for debugging shell scripts?

- a) -a
- b) -x
- c) -d
- d) -e

Answer: b

21. Suppose  $x = 10$ , then what will be the value of  $x$x$?$

- a) undefined
- b) erroneous
- c) 100
- d)  $x10$$

Answer: d

22. A shell script stopped running when we change its name. Why?

- a) location of the file changed
- b) we can't change the name of the script
- c) \$0 was used in the script
- d) many possible reasons

Answer: c

23. Where is the exit status of a command stored?

- a) \$0
- b) \$>
- c) \$1
- d) \$?

Answer: d

24. Which of the following is false?

- a) here document provides standard input to any script non interactively
- b) read command is used for making scripts interactive
- c) \$\* stores the number of arguments specified
- d) && and || are logical operators

Answer: c

25. test statement cannot \_\_\_\_\_

- a) compare two numbers
- b) compare two strings
- c) compare two files
- d) check a file's attributes

Answer: c

26. \_\_\_\_ option is used with a test for checking if the file exists and has the size greater than zero.

- a) -f
- b) -r
- c) -e
- d) -s

Answer: d

27. Which one of the following keyword is used with the set command to avoid overwrite existing file?

- a) nooverwrite
- b) ignore
- c) clobber
- d) noclobber

Answer: d

28. To prevent accident logging out, we can use \_\_\_\_ as a keyword with the set command.

- a) noclobber
- b) log out
- c) ignoreeof
- d) ignore

Answer: c

29. To reverse any setting updated by set command we can use \_\_\_\_ symbol.

- a) -
- b) /
- c) +
- d) \$

Answer: c

30. Which file is executed every time a second shell is called up?

- a) .profile
- b) rc file
- c) .script
- d) env.file

Answer: b

31. Which of the following are history variables?

- a) HISTSIZE
- b) HISTFILESIZE
- c) HISTORY
- d) HISTSIZE and HISTFILESIZE

Answer: d

32. Which one of the following command will search history list for the previous command having the last occurrence of string cvf?

- a) /cvf
- b) /cvf [enter]
- c) /bkw
- d) !cvf

Answer: a

## UNIT V

**System calls, Using system calls Pipes and Filters, Decision making in Shell Scripts (If else, switch), Loops in shell, Functions, Utility programs (cut, paste, join, tr, uniq utilities), Pattern matching utility (grep)**

Linux looks and feels much like any other UNIX system. Its development began in 1991, when a Finnish student, Linus Torvalds, wrote and christened **Linux**, a small but self-contained kernel for the 80386 processor, the first true 32-bit processor in Intel's range of PC-compatible CPUs. From an initial kernel that partially implemented a small subset of the UNIX system services, the Linux system has grown to include much ifUNIX functionality. In its early days, Linux development revolved largely around the central operating-system kernel—the core, privileged executive that manages all system resources and that interacts directly with the computer hardware. The **Linux kernel** is an entirely original piece of software developed from scratch by the Linux community. The **Linux system**, as we know it today, includes a multitude of components, some written from scratch, others borrowed from other development projects, and still others created in collaboration with other teams. A **Linux distribution** includes all the standard components of the Linux system, plus a set of administrative tools to simplify the initial installation and subsequent upgrading of Linux and to manage installation and removal of other packages on the system.

### 1.2 The Linux Kernel

The first Linux kernel released to the public was Version 0.01, dated May 14, 1991. It had no networking, ran only on 80386-compatible Intel processors and PC hardware, and had extremely limited device-driver support. The virtual memory subsystem was also fairly basic and included no support for memory mapped files; however, even this early incarnation supported shared pages with copy-on-write. The only file system supported was the Minix file system -the first Linux kernels were cross-developed on a Minix platform.

The next milestone version, Linux 1.0, was released on March 14, 1994. This release culminated three years of rapid development of the LinuX kernel. Perhaps the single biggest new feature was networking: 1.0 included support for UNIX's standard

TCP/IP networking protocols, as well as a BSD-compatible socket interface for networking programming. Device-driver support was added for running IP over an Ethernet or (using PPP or SLIP protocols) over serial lines or modems. The 1.0 kernel also included a new, much enhanced file system without the limitations of the original Minix file system and supported a range of SCSI controllers for high-performance disk access. The developers extended the virtual memory subsystem to support paging to swap files and memory mapping of arbitrary files (but only read-only memory mapping was implemented in 1.0). A range of extra hardware support was also included in this release.

Although still restricted to the Intel PC platform, hardware support had grown to include floppy-disk and CD-ROM devices, as well as sound cards, a range of mice, and international keyboards. Floating-point emulation was provided in the kernel for 80386 users who had no 80387 math coprocessor; System V UNIX-style including shared memory, semaphores, and message queues, was implemented. Simple support for dynamically loadable and unloadable kernel modules was supplied as well. At this point, development started on the 1.1 kernel stream, but numerous bug-fix patches were released subsequently against 1.0.

In March 1995, the 1.2 kernel was released. This release did not offer nearly the same improvement in functionality as the 1.0 release, but it did support a much wider variety of hardware, including the new PCI hardware bus architecture. Developers added another PC-specific feature-support for the 80386 CPU's virtual8086 mode to allow emulation of the DOS operating system for PC computers. They also updated the networking stack to provide support for the IPX protocol and made the IP implementation more complete by including accounting and firewalling functionality.

The 1.2 kernel was the final PC-only Linux kernel. The source distribution for Linux 1.2 included partially implemented support for SPARC, Alpha, and MIPS CPUs, but full integration of these other architectures did not begin until after the 1.2 stable kernels was released. The Linux 1.2 release concentrated on wider hardware support and more complete implementations of existing functionality.

Much new functionality was under development at the time, but integration of the new code into the main kernel source code had been deferred until after the stable 1.2 kernel had been released. As a result, the 1.3 development stream saw a great deal of new

functionality added to the kernel. This work was finally released as Linux 2.0 in since 1996. This release was given a major version-number increment on account of two major new capabilities: support for multiple architectures, including a 64-bit native Alpha port, and support for multiprocessor architectures. Linux distributions based on 2.0 are also available for the Motorola 68000-series processors and for Sun's SPARC systems.

A derived version of Linux running on top of the Mach microkernel also runs on PC and PowerMac systems. The changes in 2.0 did not stop there. The memory-management code was substantially improved to provide a unified cache for file-system data independent of the caching of block devices. As a result of this change, the kernel offered greatly increased file-system and virtual memory performance. For the first time, file-system caching was extended to networked file systems, and writable memory-mapped regions also were supported. The 2.0 kernel also included much improved TCP/IP performance, and a number of new networking protocols were added, including Apple

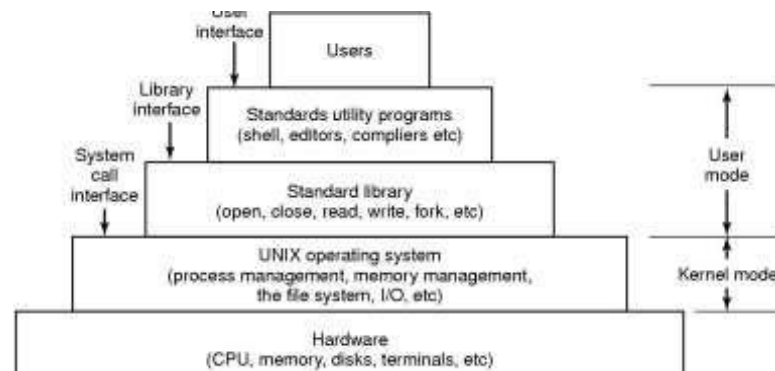
Talk, AX.25an'ateur radio networking, and ISDN support. The ability to mount remote netware and SMB (Microsoft LanManager) network volumes was added. Other major improvements in 2.0 were support for internal kernel threads, for handling dependencies between loadable modules, and for automatic loading of modules on demand. Dynamic configuration of the kernel at run time was much improved through a new, standardized configuration interface. Additional new features included file-system quotas and POSIX-compatible real-time process-scheduling classes. Improvements continued with the release of Linux 2.2 in January 1999.

A port for UltraSPARC systems was added. Networking was enhanced with more flexible firewalling, better routing and traffic management, and support for TCP large window and selective acks. Acorn, Apple, and NT disks could now be read, and NFS was enhanced and a kernel-mode NFS daemon added. Signal handling, interrupts, and some I/O were locked at a finer level than before to improve symmetric multiprocessor (SMP) performance. Advances in the 2.4 and 2.6 releases of the kernel include increased support for SMP systems, journaling file systems, and enhancements to the memory management system. The process scheduler was modified in Version 2.6, providing an efficient  $O(1)$  scheduling algorithm. In addition, the LimiX 2.6 kernel is now preemptive, allowing a process to be preempted while running in kernel mode.

### 1.3 The Linux System: Design Principles

- Linux is a multiuser, multitasking system with a full set of UNIX-compatible tools.
- Its file system adheres to traditional UNIX semantics, and it fully implements the standard UNIX networking model.
- Main design goals are speed, efficiency, flexibility and standardization.
- Linux is designed to be compliant with the relevant POSIX documents; some Linux distributions have achieved official POSIX certification.
- The Linux programming interface adheres to the SVR4 UNIX semantics, rather than to BSD behavior.

### 1.4 The layers of a UNIX system



### 1.5 Components of a UNIX System

- Like most UNIX implementations, Linux is composed of three main bodies of code:
  - Standard Utilities Programs
  - Standard Library
  - Kernel
- Standard Utilities Programs perform individual specialized management tasks.
  - Shell



- Commands for the management of files and directories
- Filters
- Compilers
- Editors
- Commands for the administration of the system

## 1.6 The shell of a UNIX System

- The UNIX systems have a Graphical User Interface (Linux uses KDE, GNOME ...), but the programmers prefer to type the commands.
- **Shell:** the user process which executes programs (command interpreter)
  - User types command
  - Shell reads command (read from input) and translates it to the operating system.
- Can run external programs (e.g. netscape) or internal shell commands (e.g. cd)
- Various different shells available:
  - Bourne shell (sh), C shell (csh), Korn shell (ksh), TC shell (tcsh), Bourne Again shell (bash).
- The administrator of the system provides to the user a default shell, but the user can change shell.

## 1.7 Memory Management

Memory management under Linux has two components. The first deals with allocating and freeing physical memory—pages, groups of pages, and small blocks of memory. The second handles virtual memory, which is memory mapped into the address space of running processes.

### Management of Physical Memory

Due to specific hardware characteristics, Linux separates physical memory into three different zones identifying different regions of memory. The zones are identified as:

- ZONE\_DMA
- ZONE\_NORMAL
- ZONE\_HIGHMEM

These zones are architecture specific. For example, on the Intel 80x86 architecture, certain ISA (industry standard architecture) devices can only access the

lower 16 MB of physical memory using DMA. On these systems, the first 16 MB of physical memory comprise ZONE-DMA. ZONE\_NORMAL identifies physical memory that is mapped to the CPU's address space. This zone is used for most routine memory requests. For architectures that do not limit what DMA can access, ZONE\_DMA is not present, and ZONE\_NORMAL is used.

Finally, ZONE\_HIGHMEM (for "high memory") refers to physical memory that is not mapped into the kernel address space. For example, on the 32-bit Intel architecture (where 2<sup>32</sup> provides a 4-GB address space), the kernel is mapped into the first 896 MB of the address space; the remaining memory is referred to as high memory and is allocated from ZONE\_HIGHMEM.

## 1.7 File Systems

Linux retains UNIX's standard file-system model. In UNIX, a file does not have to be an object stored on disk or fetched over a network from a remote file server. Rather, UNIX files can be anything capable of handling the input or output of a stream of data. Device drivers can appear as files, and inter process communication channels or network connections also look like files to the user. The Linux kernel handles all these types of file by hiding the implementation details of any single file type behind a layer of software, the virtual file system (VFS). Here, we first cover the virtual file system and then discuss the standard Linux file system—ext2fs.

### The Virtual File System

The Linux VFS is designed around object-oriented principles. It has two components: a set of definitions that specify what file-system objects are allowed to look like and a layer of software to manipulate the objects. The VFS defines four main object types:

- An **inode object** represents an individual file.
- A **file object** represents an open file.
- A **superblock object** represents an entire file system.
- A **dentry object** represents an individual directory entry.

For each of these four object types, the VFS defines a set of operations. Every object of one of these types contains a pointer to a function table. The function table lists the

addresses of the actual functions that implement the defined operations for that object. For example, an abbreviated API for some of the file object's operations includes:

- `int open (. . .)` — Open a file.
- `ssize_t read (. . .)` — Read from a file.
- `ssize_t write (. . .)` — Write to a file.
- `int mmap (. . .)` — Memory-map a file.

The complete definition of the file object is specified in the struct `file_operations`, which is located in the file `/usr/include/linux/fs.h`. An implementation of the file object (for a specific file type) is required to implement each function specified in the definition of the file object.

**The VFS** software layer can perform an operation on one of the file-system objects by calling the appropriate function from the object's function table, without having to know in advance exactly what kind of object it is dealing with. The VFS does not know, or care, whether an inode represents a networked file, a disk file, a network socket, or a directory file. The appropriate function for that file's `readQ` operation will always be at the same place in its function table, and the VFS software layer will call that function without caring how the data are actually read.

The inode and file objects are the mechanisms used to access files. An inode object is a data structure containing pointers to the disk blocks that contain the actual file contents, and a file object represents a point of access to the data in an open file. A process cannot access an inode's contents without first obtaining a file object pointing to the inode. The file object keeps track of where in the file the process is currently reading or writing, to keep track of sequential file I/O.

It also remembers whether the process asked for write permissions when the file was opened and tracks the process's activity if necessary to perform adaptive read-ahead, fetching file data into memory before the process requests the data, to improve performance. File objects typically belong to a single process, but inode objects do not.

Even when a file is no longer being used by any processes, its inode object may still be cached by the VFS to improve performance if the file is used again in the near

future. All cached file data are linked onto a list in the file's inode object. The inode also maintains standard information about each file, such as the owner, size, and time most recently modified.

Directory files are dealt with slightly differently from other files. The UNIX programming interface defines a number of operations on directories, such as creating, deleting, and renaming a file in a directory.

The system calls for these directory operations do not require that the user open the files concerned, unlike the case for reading or writing data. The VFS therefore defines these directory operations in the inode object, rather than in the file object. The superblock object represents a connected set of files that form a self-contained file system. The operating-system kernel maintains a single superblock object for each disk device mounted as a file system and for each networked file system currently connected.

The main responsibility of the superblock object is to provide access to inodes. The VFS identifies every inode by a unique (file-system/inode number) pair, and it finds the inode corresponding to a particular inode number by asking the superblock object to return the inode with that number.

Finally, a dentry object represents a directory entry that may include the name of a directory in the path name of a file (such as /usr) or the actual file (such as `stdio.h`). For example, the file `/usr/include/stdio.h` contains the directory entries (1) /, (2) usr, (3) include, and (4) `stdio.h`. Each one of these values is represented by a separate dentry object.

As an example of how dentry objects are used, consider the situation in which a process wishes to open the file with the pathname `/usr/include/stdio.h` using an editor. Because Linux treats directory names as files, translating this path requires first obtaining the inode for the root /. The operating system must then read through this file to obtain the inode for the file `include`. It must continue this process until it obtains the inode for the file `stdio.h`. Because path-name translation can be a time-consuming task, Linux maintains a cache of dentry objects, which is consulted during path-name translation. Obtaining the inode from the dentry cache is considerably faster than having to read the on-disk file.

## **The Linux ext2fs File System**

The standard on-disk file system used by Linux is called **ext2fs**, for historical reasons. Linux was originally programmed with a Minix-compatible filesystem, to ease exchanging data with the Minix development system, but that file system was severely restricted by 14-character file-name limits and a maximum file-system size of 64 MB. The Minix file system was superseded by a new file system, which was christened the **extended file system (extfs)**. A later redesign of this file system to improve performance and scalability and to add a few missing features led to the **second extended file system (ext2fs)**.

## **2. Windows 2000**

**Windows 2000** is an operating system for use on both client and server computers. It was produced by Microsoft and released to manufacturing on December 15, 1999 and launched to retail on February 17, 2000. It is the successor to Windows NT 4.0, and is the last version of Microsoft Windows to display the "Windows NT" designation.<sup>[7]</sup> It is succeeded by Windows XP (released in October 2001) and Windows Server 2003 (released in April 2003). During development, Windows 2000 was known as Windows NT 5.0.

Four editions of Windows 2000 were released: Professional, Server, Advanced Server, and Datacenter Server; the latter was both released to manufacturing and launched months after the other editions.

While each edition of Windows 2000 was targeted at a different market, they shared a core set of features, including many system utilities such as the Microsoft Management Console and standard system administration applications.

Support for people with disabilities was improved over Windows NT 4.0 with a number of new assistive technologies, and Microsoft increased support for different languages and locale information.

### **2.1 The Windows 2000 File System**

Windows 2000 supports several file systems, the most important of which are FAT-16, FAT-32, and NTFS. This sample chapter examines the NTFS file system because it is a modern file system unencumbered by the need to be fully compatible with the MS-DOS file system.

---

Windows 2000 supports several file systems, the most important of which are **FAT-16**, **FAT-32**, and **NTFS (NT File System)**. FAT-16 is the old MS-DOS file system. It uses 16-bit disk addresses, which limits it to disk partitions no larger than 2 GB. FAT-32 uses 32-bit disk addresses and supports disk partitions up to 2 TB. NTFS is a new file system developed specifically for Windows NT and carried over to Windows 2000. It uses 64-bit disk addresses and can (theoretically) support disk partitions up to  $2^{64}$  bytes, although other considerations limit it to smaller sizes. Windows 2000 also supports read-only file systems for CD-ROMs and DVDs. It is possible (even common) to have the same running system have access to multiple file system types available at the same time.

### **Fundamental Concepts**

Individual file names in NTFS are limited to 255 characters; full paths are limited to 32,767 characters. File names are in Unicode, allowing people in countries not using the Latin alphabet (e.g., Greece, Japan, India, Russia, and Israel) to write file names in their native language. For example, file is a perfectly legal file name. NTFS fully supports case sensitive names (so foo is different from Foo and FOO). Unfortunately, the Win32 API does not fully support case-sensitivity for file names and not at all for directory names, so this advantage is lost to programs restricted to using Win32 (e.g., for Windows 98 compatibility).

An NTFS file is not just a linear sequence of bytes, as FAT-32 and UNIX files are. Instead, a file consists of multiple attributes, each of which is represented by a stream of bytes. Most files have a few short streams, such as the name of the file and its 64-bit object ID, plus one long (unnamed) stream with the data. However, a file can also have two or more (long) data streams as well. Each stream has a name consisting of the file name, a colon, and the stream name, as in foo:stream1. Each stream has its own size and is lockable independently of all the other streams. The idea of multiple streams in a file was borrowed from the Apple Macintosh, in which files have two streams, the data fork and the resource fork. This concept was incorporated into NTFS to allow an NTFS server be able to serve Macintosh clients.

File streams can be used for purposes other than Macintosh compatibility. For example, a photo editing program could use the unnamed stream for the main image and a named stream for a small thumbnail version. This scheme is simpler than the traditional way of putting them in the same file one after another. Another use of streams is in word processing. These programs often make two versions of a document, a temporary one for use during editing and a final one when the user is done. By making the temporary one a named stream and the final one the unnamed stream, both versions automatically share a file name, security information, timestamps, etc. with no extra work.

The maximum stream length is  $2^{64}$  bytes. To get some idea of how big a  $2^{64}$ -byte stream is, imagine that the stream were written out in binary, with each of the 0s and 1s in each byte occupying 1 mm of space. The  $2^{67}$ -mm listing would be 15 light-years long, reaching far beyond the solar system, to Alpha Centauri and back. File pointers are used to keep track of where a process is in each stream, and these are 64 bits wide to handle the maximum length stream, which is about 18.4 exabytes.

The Win32 API function calls for file and directory manipulation are roughly similar to their UNIX counterparts, except most have more parameters and the security model is different. Opening a file returns a handle, which is then used for reading and writing the file. For graphical applications, no file handles are predefined. Standard input, standard output, and standard error have to be acquired explicitly if needed; in console mode they are preopened, however. Win32 also has a number of additional calls not present in UNIX.

### **3. Windows XP**

The Microsoft Windows XP operating system is a 32/64-bit preemptive multitasking operating system for AMD K6/K7, Intel IA32/IA64, and later microprocessors. The successor to Windows NT and Windows 2000, Windows XP is also intended to replace the Windows 95/98 operating system. Key goals for the system are security, reliability, ease of use, Windows and POSIX application compatibility, high performance, extensibility, portability, and international support.

#### **3.1 History**

In the mid-1980s, Microsoft and IBM cooperated to develop the OS/2 operating system, which was written in assembly language for single-processor Intel 80286 systems. In 1988, Microsoft decided to make a fresh start and to develop a "new technology" (or NT) portable operating system that supported both the OS/2 and POSIX application-programming interfaces (APIs). In October 1988, Dave Cutler, the architect of the DEC VAX/VMS operating system, was hired and given the charter of building this new operating system. Originally, the team planned for NT to use the OS/2 API as its native environment, but during development, NT was changed to use the 32-bit Windows API (or Win32 API), reflecting the popularity of Windows 3.0.

The first versions of NT were Windows NT 3.1 and Windows NT 3.1 Advanced Server. (At that time, 16-bit Windows was at version 3.1.) Windows NT version 4.0 adopted the Windows 95 user interface and incorporated Internet web-server and web-

browser software. In addition, user-interface routines and all graphics code were moved into the kernel to improve performance, with the side effect of decreased system reliability. Although previous versions of NT had been ported to other microprocessor architectures, the Windows 2000 version, released in February 2000, discontinued support for other than Intel (and compatible) processors due to marketplace factors. Windows 2000 incorporated significant changes over Windows NT. It added Active Directory (an X.500-based directory service), better networking and laptop support, support for plug-and-play devices, a distributed file system, and support for more processors and more memory.

In October 2001, Windows XP was released as both an update to the Windows 2000 desktop operating system and a replacement for Windows 95/98. In 2002, the server versions of Windows XP became available (called Windows .Net Server). Windows XP updates the graphical user interface (GUI) with a visual design that takes advantage of more recent hardware advances and many new ease-of-use features. Numerous features have been added to automatically repair problems in applications and the operating system itself. Windows XP provides better networking and device experience (including zero-configuration wireless, instant messaging, streaming media, and digital photography/video), dramatic performance improvements both for the desktop and large multiprocessors, and better reliability and security than even Windows 2000.

Windows XP uses a client-server architecture (like Mach) to implement multiple operating-system personalities, such as Win32 API and POSIX, with user-level processes called subsystems. The subsystem architecture allows enhancements to be made to one operating-system personality without affecting the application compatibility of any others.

Windows XP is a multiuser operating system, supporting simultaneous access through distributed services or through multiple instances of the graphical user interface via the Windows terminal server. The server versions of Windows XP support simultaneous terminal server sessions from Windows desktop systems. The desktop versions of terminal server multiplex the keyboard, mouse, and monitor between virtual terminal sessions for each logged-on user. This feature, called fast user switching, allows



users to preempt each other at the console of a PC without having to log off and onto the system.

Windows XP is the first version of Windows to ship a 64-bit version. The native NT file system (NTFS) and many of the Win32 APIs have always used 64-bit integers where appropriate—so the major extension to 64-bit in Windows XP is support for large addresses.

There are two desktop versions of Windows XP. Windows XP Professional is the premium desktop system for power users at work and at home. For home users migrating from Windows 95/98, Windows XP Personal provides the reliability and ease of use of Windows XP, but lacks the more advanced features needed to work seamlessly with Active Directory or run POSIX applications. The members of the Windows .Net Server family use the same core components as the desktop versions but add a range of features needed for uses such as webserver farms, print/file servers, clustered systems, and, large datacenter machines. The large datacenter machines can have up to 64 GB of memory and 32 processors on IA32 systems and 128 GB and 64 processors on IA64 systems.

---

### 3.1 Design Principles

Microsoft's design goals for Windows XP include security, reliability, Windows and POSIX application compatibility, high performance, extensibility, portability, and international support.

#### **Security**

Windows XP **security** goals required more than just adherence to the design standards that enabled Windows NT 4.0 to receive a C-2 security classification from the U.S. government (which signifies a moderate level of protection from defective software and malicious attacks). Extensive code review and testing were combined with sophisticated automatic analysis tools to identify and investigate potential defects that might represent security vulnerabilities.

#### **Reliability**

Windows 2000 was the most reliable, stable operating system Microsoft had ever shipped to that point. Much of this reliability came from maturity in the source code, extensive stress testing of the system, and automatic detection of many serious errors in drivers. The **reliability** requirements for Windows XP were even more stringent.

Microsoft used extensive manual and automatic code review to identify over 63,000 lines in the source files that might contain issues not detected by testing and then set about reviewing each area to verify that the code was indeed correct.

Windows XP extends driver verification to catch more subtle bugs, improves the facilities for catching programming errors in user-level code, and subjects third-party applications, drivers, and devices to a rigorous certification process. Furthermore, Windows XP adds new facilities for monitoring the health of the PC, including downloading fixes for problems before they are encountered by users. The perceived reliability of Windows XP was also improved by making the graphical user interface easier to use through better visual design, simpler menus, and measured improvements in the ease with which users can discover how to perform common tasks.

### **Windows and POSIX Application Compatibility**

Windows XP is not only an update of Windows 2000; it is a replacement for Windows 95/98. Windows 2000 focused primarily on compatibility for business applications. The requirements for Windows XP include a much higher compatibility with consumer applications that run on Windows 95/98. **Application compatibility** is difficult to achieve because each application checks for a particular version of Windows, may have some dependence on the quirks of the implementation of APIs, may have latent application bugs that were masked in the previous system, and so forth. Windows XP introduces a compatibility layer that falls between applications and the Win32 APIs. This layer makes Windows XP look (almost) bug-for-bug compatible with previous versions of Windows.

Windows XP, like earlier NT releases, maintains support for running many 16-bit applications using a thunking, or conversion, layer that translates 16-bit API calls into equivalent 32-bit calls. Similarly, the 64-bit version of Windows XP provides a thunking layer that translates 32-bit API calls into native 64-bit calls. POSIX support in Windows XP is much improved. A new POSIX subsystem called Interix is now available. Most available UNIX-compatible software compiles and runs under Interix without modification.

### **High Performance**

Windows XP is designed to provide **high performance** on desktop systems (which are largely constrained by I/O performance), server systems (where the CPU is often the bottleneck), and large multithreaded and multiprocessor environments (where locking and cache-line management are key to scalability).

High performance has been an increasingly important goal for Windows XP. Windows 2000 with SQL 2000 on Compaq hardware achieved top TPC-C numbers at the time it shipped. To satisfy performance requirements, NT uses a variety of techniques, such as asynchronous I/O, optimized protocols for networks (for example, optimistic locking of distributed data, batching of requests), kernel-based graphics, and sophisticated caching of file-system data.

The memory-management and synchronization algorithms are designed with an awareness of the performance considerations related to cache lines and multiprocessors. Windows XP has further improved performance by reducing the code-path length in critical functions, using better algorithms and per-processor data structures, using memory coloring for NUMA (non-uniform memory access) machines, and implementing more scalable locking protocols, such as queued spinlocks. The new locking protocols help reduce system bus cycles and include lock-free lists and queues, use of atomic read-modify-write operations (like interlocked increment), and other advanced locking techniques.

## **Extensibility**

**Extensibility** refers to the capacity of an operating system to keep up with advances in computing technology. So that changes over time are facilitated, the developers implemented Windows XP using a layered architecture. The Windows XP executive runs in kernel or protected mode and provides the basic system services. On top of the executive, several server subsystems operate in user mode.

Among them are **environmental subsystems** that emulate different operating systems. Thus, programs written for MS-DOS, Microsoft Windows, and POSIX all run on Windows XP in the appropriate environment.

Because of the modular structure, additional environmental subsystems can be added without affecting the executive. In addition, Windows XP uses loadable drivers in the I/O system, so new file systems, new kinds of I/O devices, and new kinds of

networking can be added while the system is running. Windows XP uses a client-server model like the Mach operating system and supports distributed processing by remote procedure calls (RPCs) as defined by the Open Software Foundation.

### **Portability**

An operating system is **portable** if it can be moved from one hardware architecture to another with relatively few changes. Windows XP is designed to be portable. As is true of the UNIX operating system, the majority of the system is written in C and C++. Most processor-dependent code is isolated in a dynamic link library (DLL) called the **hardware-abstraction layer (HAL)**. A DLL is a file that is mapped into a process's address space such that any functions in the DLL appear to be part of the process.

### **3.2 File System**

Historically, MS-DOS systems have used the file-allocation table (FAT) file system. The 16-bit FAT file system has several shortcomings, including internal fragmentation, a size limitation of 2 GB, and a lack of access protection for files. The 32-bit FAT file system has solved the size and fragmentation problems, but its performance and features are still weak by comparison with modern file systems. The NTFS file system is much better. It was designed to include many features, including data recovery, security, fault tolerance, large files and file systems, multiple data streams, UNICODE names, sparse files, encryption, journaling, volume shadow copies, and file compression. Windows XP uses NTFS as its basic file system, and we focus on it here. Windows XP continues to use FAT16, however, to read floppies and other removable media. And despite the advantages of NTFS, FAT32 continues to be important for interoperability of media with Windows 95/98 systems. Windows XP supports additional file-system types for the common formats used for CD and DVD media.

**POSSIBLE QUESTIONS**  
**UNIT – V**  
**PART – A (20 MARKS)**  
**(Q.NO 1 TO 20 Online Examinations)**

**PART – B (2 MARKS)**

1. What do you mean by system calls?
2. What are pipes?
3. What are filters?
4. What is decision making in shell script?.
5. What are utility programs?

**PART – C (6 MARKS)**

1. Discuss the system calls.
2. Explain the pipes and filters.
3. Explain the decision making in shell script.
4. Discuss the utility programs of Unix and Linux.
5. Explain the pattern matching utility.

Prepared by Dr.M.Sakthi Asvini, Assistant Professor, CS,CA,IT

## UNIT V

1. Which command is used to get intermediate result in a pipeline of commands?

- a) tee
- b) tea
- c) get
- d) filter

Answer : a)

2. Which of the following features of UNIX may be used for inter process communication?

- a) signals
- b) pipes
- c) semaphore
- d) all of the given answers

Answer: d)

3. Which command is used to extract a column from a text file?

- A. paste
- B. get
- C. cut
- D. tar

Answer: c)

4. What is the function of cp command in UNIX?

- a) list all the available files in the current directory
- b) delete a given file
- c) cp is a command used for copying files and directories
- d) change the directory

Answer: c

5. What happens if the destination file specified in cp command does not exist?

- a) file will not be copied
- b) an error will be produced
- c) destination file will be automatically created
- d) none of the mentioned

Answer: c

6. Which of the following is not an option of cp command?

- a) -z
- b) -i
- c) -R
- d) -u

Answer: a

7. What is the correct syntax for copying multiple files with a filename starting as 'file' into another file named as 'directory\_one'?

- a) `cp -i file directory_one`
- b) `cp -R file directory/directory_one`
- c) `cp file* directory_one`
- d) none of the mentioned

Answer: c

8. How can we copy an entire directory under another directory?

- a) using -R option
- b) using -a option
- c) using -u option
- d) none of the mentioned

Answer: a

9. How can we copy a file into our current directory?

- a) `cp file1`
- b) `ct file1`
- c) `cp file*`
- d) `ct file*`

Answer: b

10. What does the following command do?

`cp -u * dir_file`

- a) copy all files to directory dir\_file
- b) update all files
- c) delete all files
- d) update all files in the current working directory and copy newer ones to directory dir\_file

Answer: d

11. -n option is used with cp command for what purpose?

- a) existing file should not be overwritten
- b) to update file
- c) interactive copying
- d) recursive copying

Answer: a

12. Which option is used with cp command for linking files instead of copying?

- a) -v
- b) -l
- c) -f
- d) -x

Answer: b

13. -v option is used with cp command for displaying \_\_\_\_\_

- a) errors
- b) informative messages

- c) diagnostic messages
- d) file contents

Answer: b

14. Which command is used for displaying contents of a file?

- a) cp
- b) rm
- c) cat
- d) mkdir

Answer: c

15. Apart from displaying file contents, cat command is also used for \_\_\_\_\_ files.

- a) displaying
- b) deleting
- c) copying
- d) creating

Answer: d

16. Which symbol is used with cat command for creating files?

- a) >
- b) <
- c) \*
- d) /

Answer: a

17. If we create a file using cat command with the same filename which already exists in the current directory then,

- a) existing file is deleted
- b) new file will be created separately
- c) existing file will be overwritten
- d) an error will be produced

Answer: c

18. Which symbol is used to append an existing file?

- a) >
- b) <
- c) >>
- d) \$

Answer: c

19. Which option is used with cat command for displaying non-printable characters?

- a) -v
- b) -n
- c) -x
- d) -a

Answer: a



20. Which option is used with the cat command for displaying file with line numbers?

- a) -n
- b) -v
- c) -a
- d) -x

Answer: a

21. Which of the following cannot be performed by cat command?

- a) displaying files
- b) creating files
- c) appending files
- d) deleting files

Answer: d

22. What does cat file01 file01 file01 display?

- a) error
- b) blank terminal
- c) contents of file01 three times successively
- d) contents of file01 single time

Answer: c

23. Which files will be displayed by the following command:

cat \*file\*

- a) all files in the directory
- b) all files with filename containing 'file'
- c) no files will be displayed
- d) a single file

Answer: b

24. Which command is used to create empty files?

- a) cp
- b) cat
- c) touch
- d) create

Answer: d

25. Which option is used with touch command which forces the command not to create file, if it does not exists.

- a) -h
- b) -c
- c) -t
- d) -f

Answer: b

26. Which of the following operators are used for logical execution?

- a) ||
- b) &&

- c) %%
- d) && and ||

Answer: d

27. The syntax for using && is \_\_\_\_\_

- a) cmd1 && cmd2
- b) cmd1 cmd2 &&
- c) cmd1 & cmd2&
- d) cmd1

Answer: a

28. To perform decision depending on the fulfillment of certain criteria, \_\_\_\_\_ is used.

- a) if
- b) else
- c) for
- d) if and else

Answer: d

29. Every if is closed with a corresponding \_\_\_\_\_

- a) else
- b) fi
- c) if
- d) else if

Answer: b

30. To check more than two conditions, \_\_\_\_\_ is used with if-else statements.

- a) while
- b) for
- c) elif
- d) for

Answer: c

31. The name of the script is stored in which special parameter?

- a) \$1
- b) \$0
- c) \$#
- d) \$\*

Answer: b

32. \_\_\_\_\_ statement matches an expression for more than one alternative.

- a) for
- b) while
- c) elif
- d) case

Answer: d

33. Every pattern in case statement is terminated with a \_\_\_\_\_

- a) ;

- b) :
- c) ;;
- d) //

Answer: c

34. The \_\_\_\_ option in case statement matches any option not matched by the previous options.

- a) ^
- b) \$
- c) \*
- d) //

Answer: c

35. Which command is used for computation and string handling?

- a) expr
- b) case
- c) if
- d) read

Answer: a

36. expr can perform \_\_\_\_ arithmetic operations.

- a) 2
- b) 4
- c) 5
- d) 3

Answer: c

37. Which of the following is performed by expr string handling's function?

- a) determine the length of string
- b) extract a substring
- c) locate the position of a character in a string
- d) determine the length of string, extract and locate the position of the string

Answer: d

38. expr is a \_\_\_\_ command

- a) internal
- b) external
- c) shell
- d) derived

Answer: b

39. Which command is used for locating repeated and non-repeated lines?

- a) sort
- b) uniq
- c) cut
- d) paste

Answer: b

40. Which option is used with uniq command for selecting non-repeated lines?

- a) -i

- b) -c
- c) -u
- d) -a

Answer: c

41. Which option is used for selecting repeated entries?

- a) -d
- b) -c
- c) -u
- d) -a

Answer: a

42. \_\_\_\_\_ option is used for counting frequency of occurrence.

- a) -d
- b) -c
- c) -u
- d) -a

Answer: b

43. test command can be used to check which of the following?

- a) Compare two numbers
- b) Compare two strings
- c) Check attributes of a file
- d) All of the mentioned

Answer: d

44. \_\_\_\_ implies greater than and \_\_\_\_ implies less than.

- a) gt, le
- b) gt, lt
- c) ge,le
- d) ge,lt

Answer: b

45. Which one of the following option is used for AND operation in test command?

- a) -o
- b) -a
- c) -e
- d) -an

Answer: b

46. Which one of the following option is used for OR operation in test command?

- a) -o
- b) -a
- c) -e
- d) -an

Answer: a

47. Which one of the following option is used for checking that the string is not null?

- a) -a
- b) -o
- c) -z
- d) -n

Answer: d

48. To extract specific columns from a file, \_\_\_\_ command is used.

- a) tail
- b) head
- c) pr
- d) cut

Answer: d

49. \_\_\_\_ option is used with the cut command for cutting fields.

- a) -c
- b) -n
- c) -f
- d) -a

Answer: c

50. Which command is used for pasting files?

- a) cut
- b) paste
- c) tail
- d) head

Answer: b

51. \_\_\_\_ option is used with paste command if we want to specify our own delimiter.

- a) -d
- b) -c
- c) -a
- d) -e

Answer: a

52. Which option is used with paste command for joining lines?

- a) -s
- b) -c
- c) -a
- d) -e

Answer: a