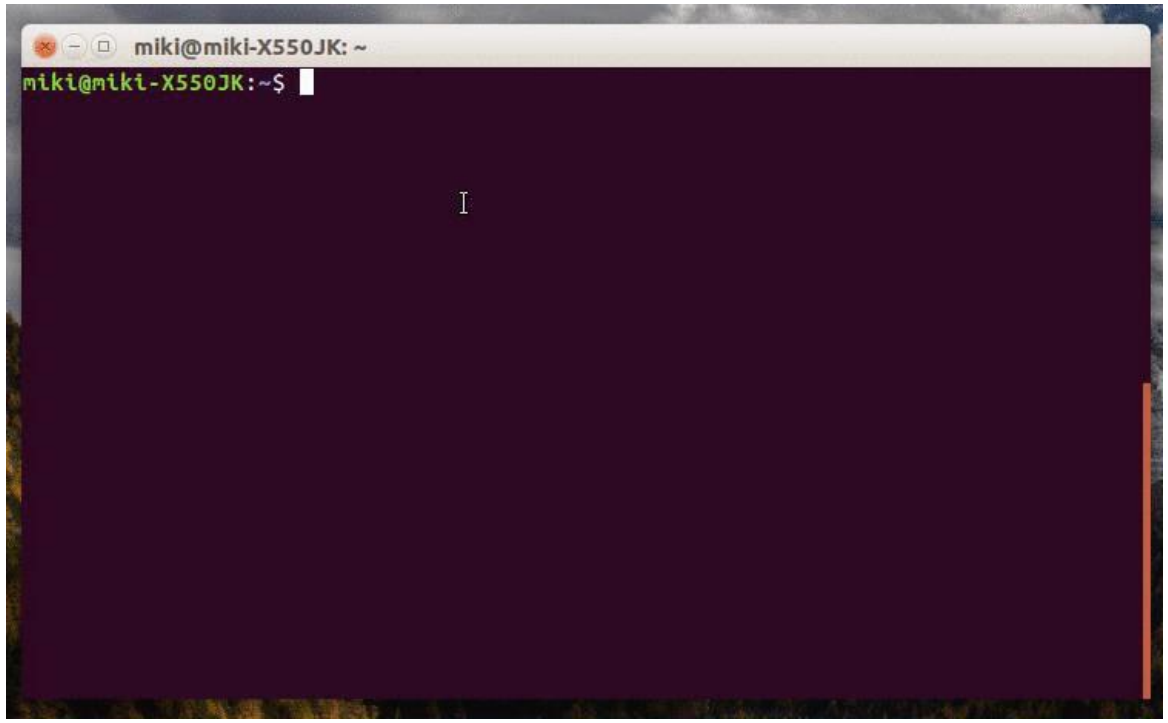
TYBSC CS
PAPER II

LINUX SYSTEM ADMINISTRATION



Technical Summary of Linux Distributions

Distribution (or distro)

Linux itself is the core of the operating system: **the *kernel.***

- It is responsible for
  - ✓ starting and stopping other programs (such as editors)
  - ✓ handling requests for memory
  - ✓ accessing disks
  - ✓ managing network connections
- The kernel is a nontrivial program.
- Linux distributions
  - **Red Hat Enterprise Linux (RHEL)**
  - **Fedora**
  - **Debian**
  - **Mandrake**
  - **Ubuntu**
  - **Kubuntu**
  - **openSUSE**
  - **goBuntu, and so on**

Linux distributions can be broadly categorized into two groups:

➢ Commercial distros

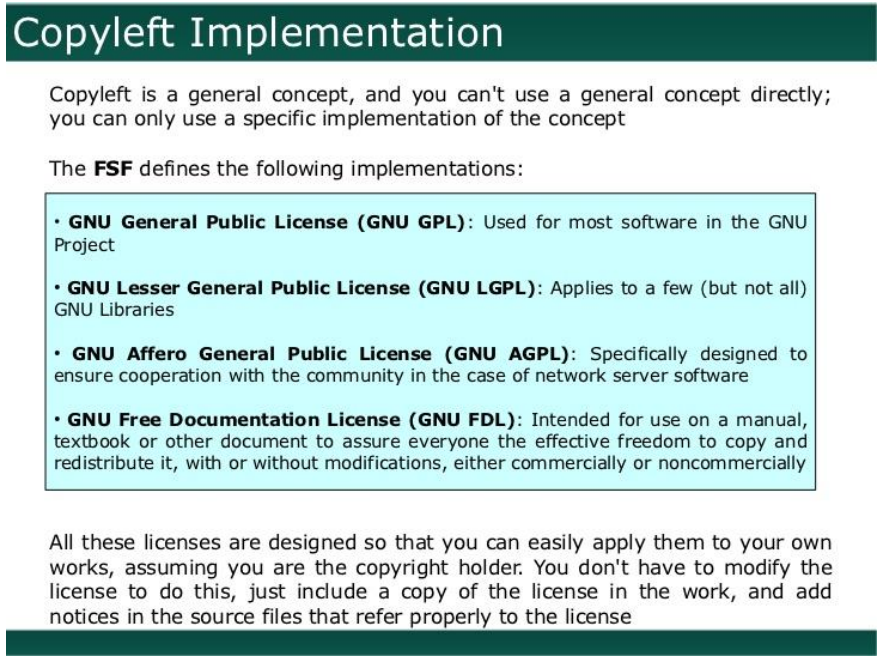
- The commercial distros generally offer support for their distribution—at a cost. The commercial distros also tend to have a longer release life cycle.
- The companies that offer the purely commercial flavours have vested interests in making sure that free distros exist.
- Some of the companies use the free distros as the proofing and testing ground for software that ends up in the commercial spins
- Examples of commercial flavours of Linux-based distros are RHEL, SuSE Linux Enterprise (SLE), etc.

➢ Non-commercial distros.

- The non-commercial distros, on the other hand, are free.
- The non-commercial distros try to adhere to the original spirit of the open source software.
- They are mostly community supported and maintained—the community consists of the users and developers.
- The community support and enthusiasm can sometimes supersede that provided by the commercial offerings. Several of the so-called non-commercial distros also have the backing and support of their commercial counterparts.
- Examples of non-commercial flavours of Linux-based distros are Fedora, OpenSuSE, Ubuntu, goBuntu, Debian, etc. What's interesting about the commercial Linux distributions is that most of the tools

What Is the GNU Public License?

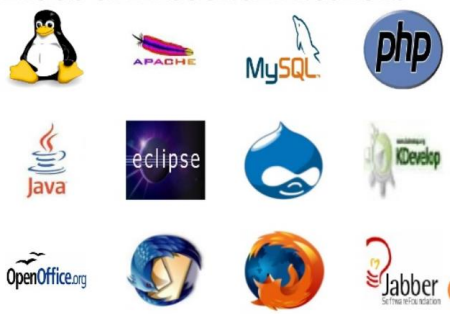
- An important thing to emerge from the GNU project has been the *GNU Public License (GPL).*
- This license explicitly states that the software being released is free and that no one can ever take away these freedoms.
- It is acceptable to take the software and resell it, even for a profit; however, in this resale, the seller must release the full source code, including any changes.
- Because the resold package remains under the GPL, the package can be distributed for free and resold yet again by anyone else for a profit.
- Of primary importance is the liability clause: The programmers are not liable for any damages caused by their software.
- It should be noted that the GPL is not the only license used by open source software developers (although it is arguably the most popular).
- Other licenses, such as BSD and Apache, have similar liability clauses but differ in terms of their redistribution.
- For instance, the BSD license allows people to make changes to the code and ship those changes without having to disclose the added code.



WHAT IS OPEN SOURCE?

A software, come with the source code, and can be used for free with an agreement. Any one can contribute in this software. You can embed it in your projects. You can redistribute this software and drive it.

FAMOUS OPEN SOURCE PROJECTS



DISADVANTAGES OF OPEN SOURCE

- No warranties regarding media, viruses, and performance.
- No maintenance and support (unless purchased separately).
- It is inherently insecure because of the availability of the source code.

ADVANTAGES OF OPEN SOURCE

- No or low license fees.
- The availability of the source code.
- The right of code modification, improvement and redistribution.
- Free marketing and support for your project.
- Community reaction to bug reports is much faster compared to closed source software which makes it easier to fix bugs and make the component highly secure.
- Best choice for small enterprises especially at start-up.
- Have alternatives providing maintenance and support rather than tailored to specific vendor in case of closed source software.

THE DIFFERENCES BETWEEN WINDOWS AND LINUX

Single Users vs. Multiple Users vs. Network Users

- Windows was designed according to the “one computer, one desk, one user” vision of Microsoft’s cofounder Bill Gates.
- For the sake of discussion, we’ll call this philosophy ***single-user.***
- In this arrangement, two people cannot work in parallel running (for example) Microsoft Word on the same machine at the same time.
- You can buy Windows and run what is known as Terminal Server, but this requires huge computing power and extra costs in licensing.

- Of course, with Linux, you don’t run into the cost problem, and Linux will run fairly well on just about any hardware.
- It required a design that allowed for ***multiple users*** to log into the central machine at the same time.
- Various people could be editing documents, compiling programs, and doing other work at the exact same time.
- The operating system on the central machine took care of the “sharing” details so that each user seemed to have an individual system.

- Today, the most common implementation of a multiuser setup is to support ***servers***—systems dedicated to running large programs for use by many clients.
- Each member of a department can have a smaller workstation on the desktop, with enough power for day to-day work.
- When they need to do something requiring significantly more processing power or memory, they can run the operation on the server.

The Monolithic Kernel and the Micro-Kernel

- In operating systems, there are two forms of kernels.
- You have a monolithic kernel that provides all the services the user applications need.
- And then you have the micro-kernel, a small core set of services and other modules that perform other functions.
- Linux, for the most, part adopts the monolithic kernel architecture; it handles everything dealing with the hardware and system calls.
- Windows works off a micro-kernel design.
- The kernel provides a small set of services and then interfaces with other executive services that provide process management, input/output (I/O) management, and other services.

DIFFERENCE BETWEEN MONOLITHIC AND MICRO KERNEL

Monolithic kernel design is much older than the microkernel idea, which appeared at the end of the 1980's

MONOLITHIC	MICRO KERNEL
Monolithic kernels are usually faster than microkernels. The first microkernel Mach was 50% slower than most monolithic kernels, while later ones like L4 were only 2% or 4% slower than the monolithic designs.	Microkernels are comparatively slower.
All the parts of a kernel like the Scheduler, File System, Memory Management, Networking Stacks, Device Drivers, etc., are maintained in one unit within the kernel in **Monolithic Kernel**	Only the very important parts like IPC(Inter process Communication), basic scheduler, basic memory handling, basic I/O primitives etc., are put into the kernel. Communication happen via message passing. Others are maintained as server processes in User Space
Faster processing	Slower Processing due to additional Message Passing
Monolithic kernel is a single large process running entirely in a single address space. It is a single static binary file. All kernel services exist and execute in the kernel address space. The kernel can invoke functions directly. Examples of monolithic kernel-based OSs: Unix, Linux.	In microkernels, the kernel is broken down into separate processes, known as servers. Some of the servers run in kernel space and some run-in user-space. All servers are kept separate and run-in different address spaces. Servers invoke "services" from each other by sending messages via IPC (Inter process Communication). This separation has the advantage that if one server fails, other servers can still work efficiently. Examples of microkernel-based OSs: Mac OS X and Windows NT