Linux and Open Source

Satvik Patwardhan

The Debuggers Coding Club
Department of Computer Science, Banaras Hindu University, Varanasi

What is an OS?

- The operating system (OS) is the software that provides an interface between the user and the hardware of the computer.
- The hardware provides "raw computing power", the OS manages this computing power to make it easily available to the user and make sure it is used efficiently.
- Examples: Windows, Linux, macOS, Android

What does an OS do?

- OS is responsible for managing processes (running programs) in the system
- Responsible for managing memory (RAM)
- Responsible for all communication with hardware
- Responsible for handling files

Modes of Execution

- Programs run in an OS consist of instructions given to the CPU.
- Most CPUs do not allow all programs to run all types of instructions.
- Instructions in programs run by the user run in user mode
- Instructions run directly by the OS run in kernel mode

Kernel

- Kernel is the portion of the OS responsible for important system functions
- All the previous tasks of OS handled by kernel (process management, memory management, I/O management)
- Kernel can access sensitive system commands (stop process, shut down)

Source Code

- The software that we use is made by writing source code, compiling it and sharing the built binary files
- These executables are in machine code: difficult to understand logic or modify
- Proprietary software: Source code is not available to public, distributing modifications illegal (also called closed-source software)

Free and Open Source Software

- In **Free and Open Source Software (FOSS)**, source code is publicly available along with executables
- Modifying and exchanging source code is encouraged
- Examples: Android, Firefox, VLC media player etc.

Why FOSS?

- Many people working and collaborating on same source code can catch bugs more easily
- Typically free of cost, if not can be compiled for free
- Can be modified and customized to your personal needs
- More scope for creativity and innovation, no restrictions from companies

How did FOSS come about?

- Early decades of programming was mostly collaborative
- Software was shared as source code itself
- As software development shifted from academia to corporations, source code sharing became discouraged due to lack of profitability
- In 1980, copyright law in US was changed to allow source code to be copyrighted
- This angered everyone apart from the companies

How did FOSS come about? (contd.)

- The most popular operating system at the time, UNIX, slowly became corporate controlled and proprietary
- In response, Richard Stallman. from the MIT Artificial Intelligence lab, started the GNU Project
- GNU (GNU's Not Unix) aimed to make a set of FOSS software which can be used to run a computer
- He introduced "copyleft" (as opposed to copyright), where anyone could modify and share works released as such, and others cannot take away this right from their modifications either

Four Freedoms

- Richard Stallman gave "four freedoms" as the definition of free software
- Freedom 0: The freedom to use the program for any purpose.
- Freedom 1: The freedom to study how the program works, and change it to make it do what you wish.
- Freedom 2: The freedom to redistribute and make copies so you can help your neighbor.
- Freedom 3: The freedom to improve the program, and release your improvements (and modified versions in general) to the public, so that the whole community benefits.

The History of UNIX

- In the 60s and 70s, UNIX was the most popular operating system
- Made by Ken Thompson (inventor of B programming language), Dennis Ritchie (inventor of C programming language) and others at Bell Labs
- Unix was proprietary and later expensive

The GNU Project

- Richard Stallman started the GNU project (GNU's Not Unix) to make an alternative to UNIX
- GNU project was very succesful: assembler, gcc C compiler, emacs text editor, and most OS system commands completed
- GNU software is still a core part of Linux and Mac, released as open source under GNU General Public License (GPL)
- By 1991, only the kernel was left to be finished

The Linux kernel

- Work on the GNU kernel was going slowly
- MINIX was a proprietary UNIX kernel for made for students to study
- In 1991, Linus Torvalds, a computer science student at Univesity of Helsinki, made a kernel similar to MINIX called “Linux”
- By combining the GNU system software and the Linux kernel, the first FOSS operating system was created

Development of the Linux kernel

- Linux kernel is now developed by the community
- Hobbyists and students contribute just as much as employees of Google and Microsoft
- Focus is entirely on the kernel: everything from basic system commands to GUI is handled by others

Components of a Linux system

Various layers within Linux, also showing separation between the userland and kernel space

User mode	**User applications**	*bash*, *LibreOffice*, *GIMP*, *Blender*, *0 A.D.*, *Mozilla Firefox*, ...				
	System components	**init daemon**: *OpenRC*, *runit*, *systemd*...	**System daemons**: *polkitd*, *smbd*, *sshd*, *udevd*...	**Windowing system**: *X11*, *Wayland*, *SurfaceFlinger* (Android)	**Graphics**: *Mesa*, *AMD Catalyst*, ...	**Other libraries:** *GTK*, *Qt*, *EFL*, *SDL*, *SFML*, *FLTK*, *GNUstep*, ...
	C standard library	`fopen`, `execv`, `malloc`, `memcpy`, `localtime`, `pthread_create`... (up to 2000 subroutines) *glibc* aims to be fast, *musl* aims to be lightweight, *uClibc* targets embedded systems, *bionic* was written for Android, etc. All aim to be POSIX/SUS-compatible.				
Kernel mode	**Linux kernel**	`stat`, `splice`, `dup`, `read`, `open`, `ioctl`, `write`, `mmap`, `close`, `exit`, etc. (about 380 system calls) The Linux kernel System Call Interface (SCI), aims to be POSIX/SUS-compatible[99]				
		Process scheduling subsystem	IPC subsystem	Memory management subsystem	Virtual files subsystem	Networking subsystem
		Other components: ALSA, DRI, evdev, klibc, LVM, device mapper, Linux Network Scheduler, Netfilter Linux Security Modules: *SELinux*, *TOMOYO*, *AppArmor*, *Smack*				
Hardware (CPU, main memory, data storage devices, etc.)						

Source: Wikipedia

System Components

- Init system – responsible for starting and stopping processes
- Daemons – processes that are always running (network, audio, battery monitor etc.)
- Windowing system – responsible for creating, showing and destroying app windows
- Graphics toolkit – responsible for creating and adjusting buttons, menus, content within window

System Components (Contd.)

- Desktop environment – main user interface of the computer. Lets users open and close apps, switch between apps, change system settings etc.
- Package manager – basically an app store like Google Play, Apple App Store etc. Provides both apps and system components

Linux Distributions

- For every single type of system component, there are many different options made by different people
- Users cannot be expected to choose every option wisely
- A Linux distribution (distro) is a compilation of all the system components needed to run the computer
- Examples: Ubuntu, Fedora, Debian, Arch Linux etc.
- A distribution chooses which software to include based on different philosophies (which we do not have time to discuss)

Examples of Linux Distros

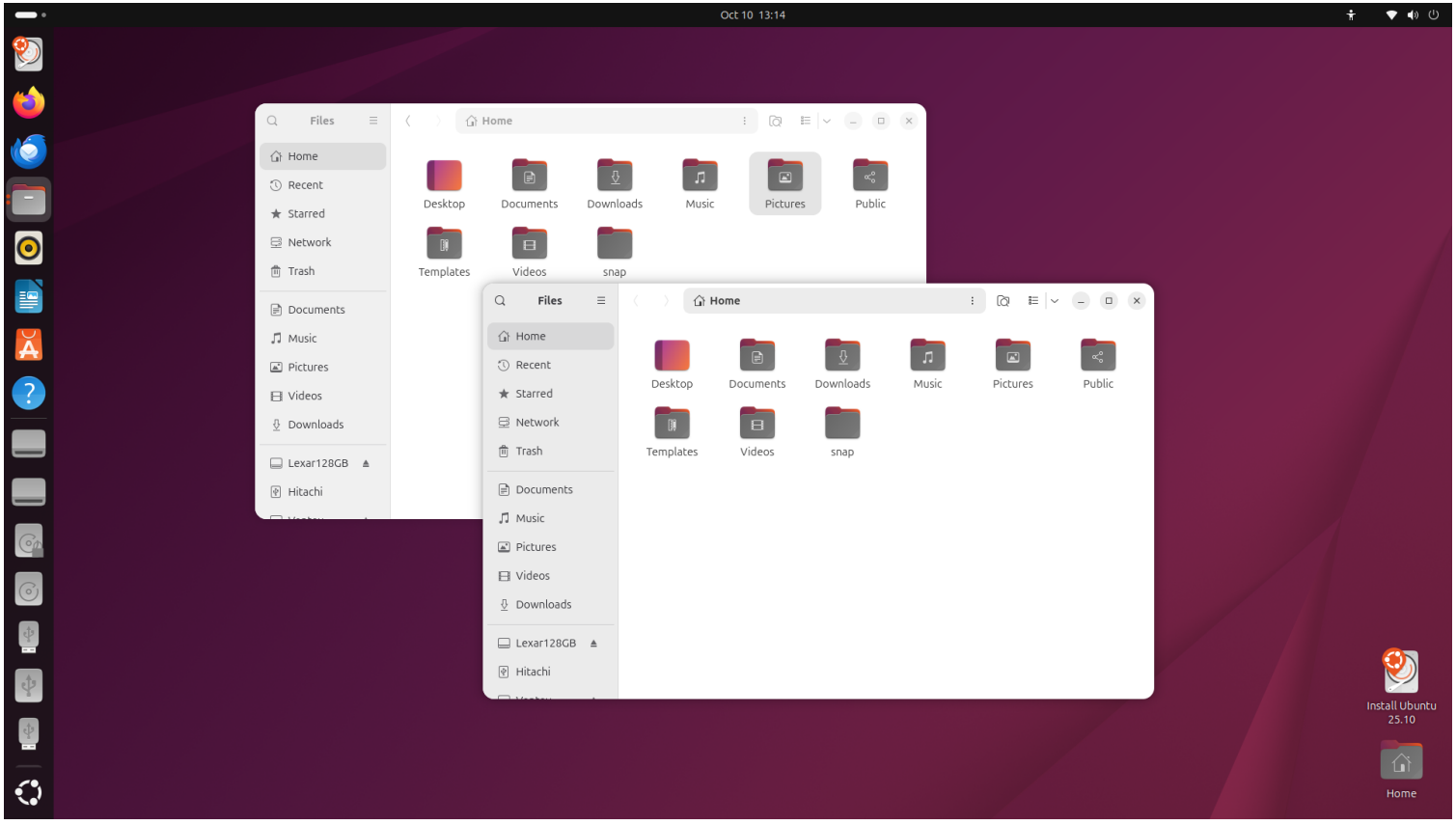

Figure: Ubuntu with GNOME desktop environment

Source: Canonical Limited, GPL <http://www.gnu.org/licenses/gpl.html>, via Wikimedia Commons

Figure: Fedora Linux with KDE Plasma desktop environment

Source: The Fedora Project, GPL <http://www.gnu.org/licenses/gpl.html>, via Wikimedia Commons

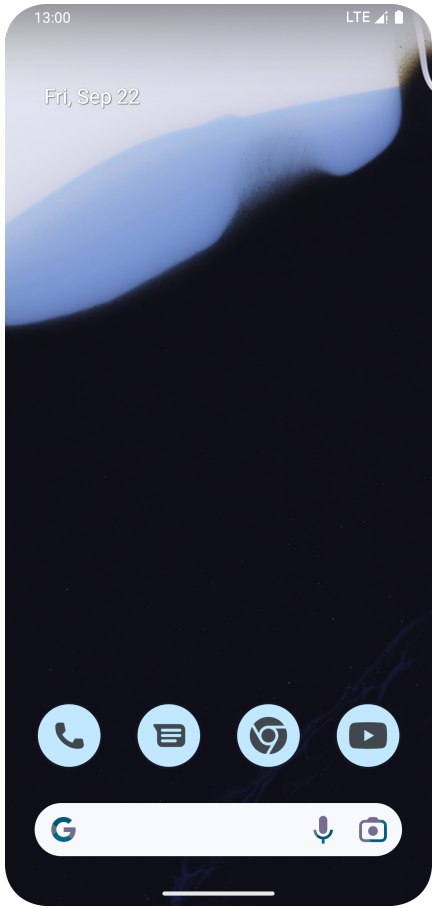

Figure: Android

Source: The Android Open Source Project, Apache License 2.0 <http://www.apache.org/licenses/LICENSE-2.0>, via Wikimedia Commons

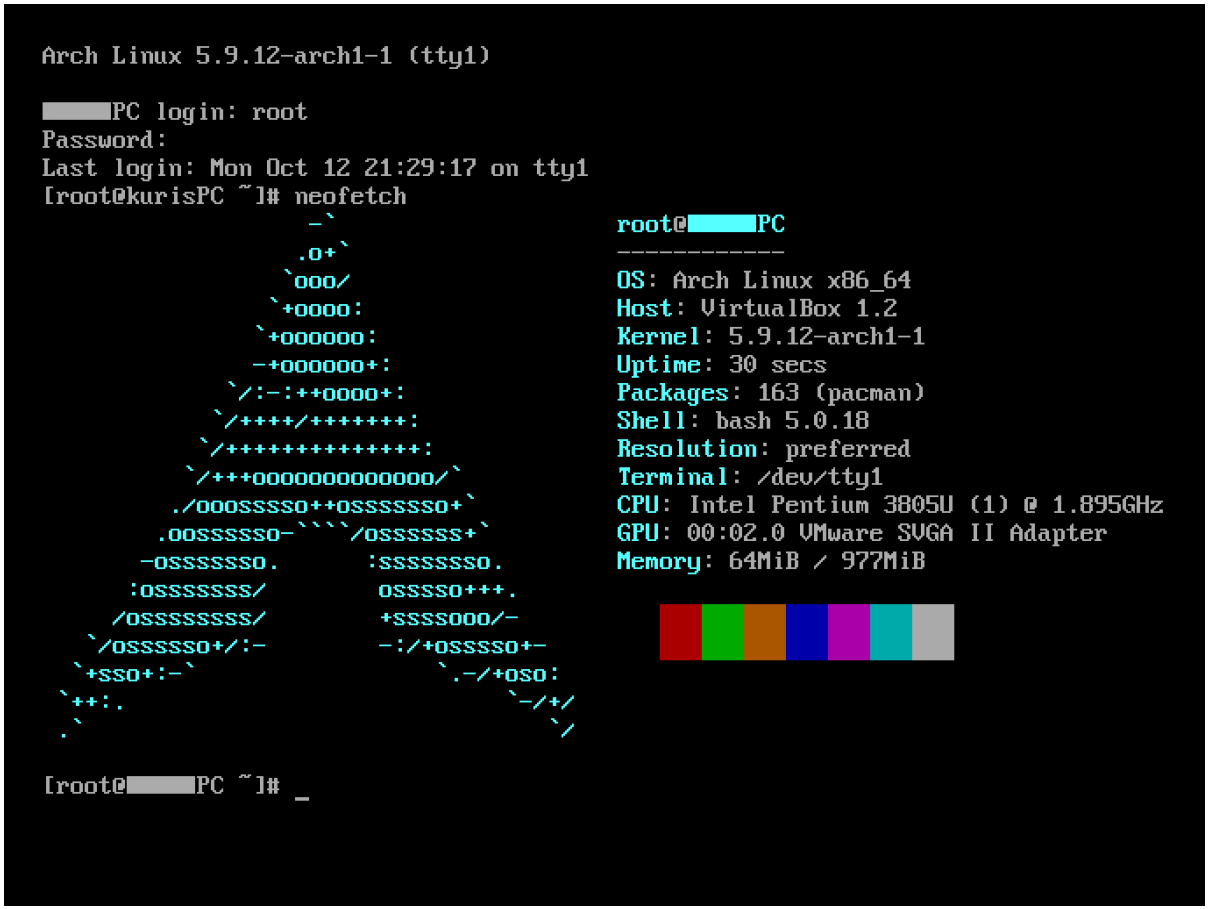

Figure: Arch Linux with no desktop environment installed