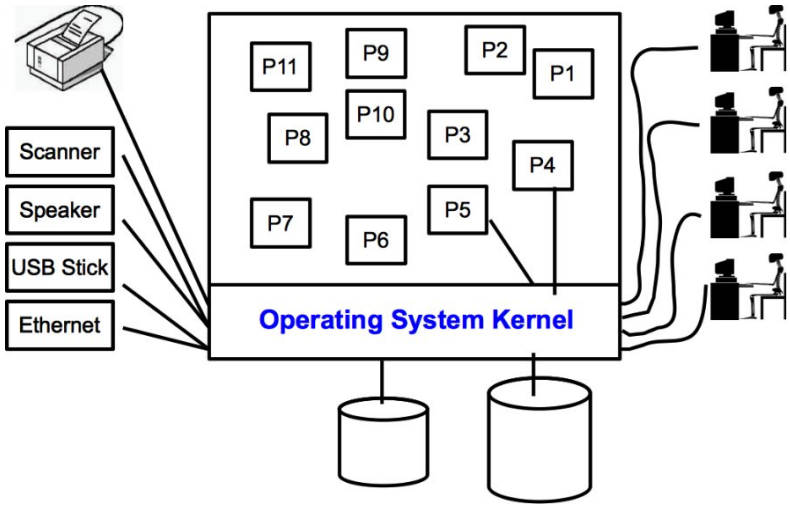
HO#1.3: A Recap of OS with Linux

Overview of Operating Systems

An OS is a program running on the computer (usually called the kernel), that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware. The primary goal of OS is convenience of user and secondary goal is efficient operation of the computer system. It manages computer hardware and provides services for computer programs.



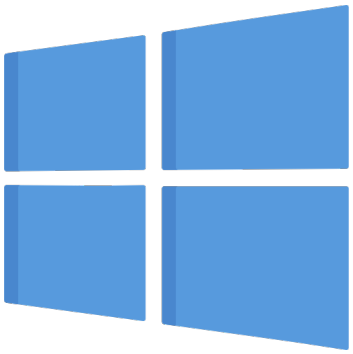
Here's a quick rundown of its key functions:

- **Resource Management**: It manages the computer's hardware resources, such as the CPU, memory, disk space, and input/output devices. It ensures that different programs and users running on the computer don't interfere with each other's operations.
- **File Management**: It handles the creation, deletion, and organization of files and directories on storage devices. It also manages file permissions and access control.
- **Process Management**: It manages the execution of programs (processes), including multitasking, which allows multiple processes to run simultaneously. The OS allocates CPU time and memory space to these processes.
- **User Interface**: It provides an interface for users to interact with the computer, which can be command-line (CLI), graphical (GUI) or Natural User Interface (NUI). A Natural User Interface (NUI) is a revolutionary approach in design that allows users to interact with technology in a way that feels natural and intuitive. Unlike GUI that rely heavily on buttons, menus, and icons, NUIs use gestures, voice commands, touch, and eye movements, mimicking real-world interactions.
- **Security and Access Control**: It ensures that unauthorized users do not access the system or its data. This includes managing user accounts, passwords, and permissions.
- **Device Management**: It controls and coordinates the use of hardware peripherals like printers, keyboards, and external drives through device drivers.

Types of Operating Systems

Popular operating systems include **Windows, macOS, Linux, Android and iOS.** Each of these operating systems has its own set of features and design philosophies, tailored to different types of users and hardware. Here's a detailed overview of various operating systems, that you need NOT to memorize ☺, however, do give them a bird's eye view

- **Desktop and Laptop Operating Systems**
  1. **Windows**: Developed by Microsoft, it is the most widely used desktop OS. Known for its user-friendly graphical interface and broad software compatibility. The evolution of the Windows operating system reflects the broader changes in computing technology and user needs. Here's a detailed look at the major versions of Windows and their evolution:



- **Windows 1.0 (1985)**: The first version of Windows, which was essentially a graphical extension for MS-DOS. It introduced a basic graphical user interface (GUI) with windows, icons, and menus but was limited in functionality.

- **Windows 2.0 (1987)**: Improved upon Windows 1.0 by adding better memory management, support for overlapping windows, and enhanced graphics. It was still dependent on MS-DOS.

- **Windows 3.0 (1990)**: Marked a significant improvement with a more advanced GUI, virtual memory, and improved performance. It introduced the Program Manager and File Manager.

- **Windows 3.1 (1992)**: Introduced TrueType fonts for better typography and improved multimedia support. It was also the first version to gain significant popularity.

- **Windows 95 (1995)**: A major milestone with a new GUI, Start Menu, taskbar, and Plug and Play hardware support. It integrated MS-DOS and Windows into a single operating system.

- **Windows 98 (1998)**: Built on Windows 95 with improved hardware support, Internet Explorer integration, and better system performance. It also introduced the Windows Update feature.

- **Windows ME (Millennium Edition) (2000)**: Focused on multimedia and home networking. It was criticized for its stability issues but introduced System Restore and improved Internet support.

- **Windows XP (2001)**: A highly successful version combining the consumer-oriented Windows 95/98/ME and the business-oriented Windows NT line. Known for its stability, improved GUI, and the introduction of features like fast user switching and enhanced security.

- **Windows Vista (2007)**: Introduced a new Aero GUI with advanced graphics and visual effects. It featured improved security (User Account Control) and better networking but faced criticism for performance issues and compatibility problems.

- **Windows 7 (2009)**: Improved upon Vista with better performance, a more refined interface, and enhanced features like the redesigned taskbar and libraries. It was widely praised for its stability and user experience.

- **Windows 8 (2012)**: A radical redesign focusing on touch-based devices with the introduction of the Metro UI (later known as Modern UI). It featured a Start Screen instead of the traditional Start Menu and brought improvements for tablets and hybrid devices.

- **Windows 8.1 (2013)**: Addressed some criticisms of Windows 8 by reintroducing the Start button and improving multitasking and personalization features.

- **Windows 10 (2015)**: Marked a return to a more familiar desktop interface with the Start Menu and integrated new features like Cortana (digital assistant), Microsoft Edge (web browser), and virtual desktops. It introduced the concept of "Windows as a Service," with regular updates rather than major new versions.

- **Windows 11 (2021)**: Introduced a redesigned Start Menu, centered taskbar, and a more modern interface with rounded corners. It focused on improving productivity and multitasking with features like Snap Layouts and virtual desktops. It also emphasized performance, security, and support for the latest hardware.

2. **Unix-based OS:** Unix is a powerful, multiuser, multitasking operating system originally developed in the late 1960s and early 1970s. It has influenced many modern operating systems and has several variants that have evolved over time. Here's a detailed look at the evolution of Unix and its derivatives:

   - **Unix Version 1 (1969)**: Developed by Ken Thompson, Dennis Ritchie, and others at AT&T's Bell Labs. It was initially created on a PDP-7 and used a simple, monolithic kernel with a command-line interface. It laid the groundwork for many Unix concepts.

   - **Unix Version 6 (1975)**: One of the most influential early versions, it introduced features such as pipes, which allowed the output of one program to be used as the input for another. It was widely distributed and had a significant impact on the spread of Unix.

   - **Unix Version 7 (1979)**: Also known as Unix V7, it included significant improvements, such as the Bourne shell (sh) and the C programming language (which was used to write the kernel and utilities). This version was highly influential and became the basis for many later Unix variants.

- **System III UNIX (1982):** AT&T's Unix System III was an attempt to unify and standardize the various Unix versions. It introduced new features and served as a foundation for future commercial Unix versions.

- **System V UNIX (1983)**: This version, often referred to as Unix System V, introduced several key features, including the System V Interface Definition (SVID), a standardized interface for Unix system calls and utilities. It also introduced important enhancements like the SysV init system, shared libraries, and the STREAMS networking interface.

- **BSD UNIX (Berkeley Software Distribution)**: An influential series of Unix versions developed at the University of California, Berkeley. BSD Unix introduced many innovations such as the TCP/IP stack, the vi editor, and the BSD license.
    - **Versions**:
        1. **BSD 4.2 (1983)**: Introduced important features like the TCP/IP networking stack.
        2. **BSD 4.4 (1994)**: Known for its extensive features, including advanced networking, filesystem improvements, and tools.

- **AIX (1986)**: IBM's Unix variant, originally based on System V. AIX introduced features tailored for IBM hardware and has been used in various IBM servers and workstations.

- **HP-UX (1984)**: Hewlett-Packard's Unix variant, based on System V, but with additional features specific to HP's hardware and software environments.

- **Solaris (1992)**: Sun Microsystems' Unix variant, initially based on System V and later incorporating elements from BSD. Solaris introduced features such as the ZFS filesystem and DTrace for dynamic tracing.

- **Linux (1991-):** Although not a direct derivative of Unix, Linux is a Unix-like operating system created by Linus Torvalds. It is influenced by Unix principles and has become a major operating system for servers, desktops, and embedded systems.

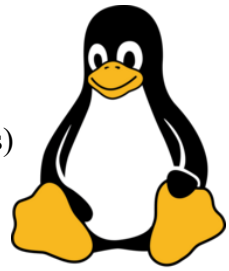
- **macOS (2001-)**: Apple's operating system for Macintosh computers, based on a Unix foundation. It incorporates elements from the NeXTSTEP operating system and provides a Unix-based environment with a user-friendly graphical interface.

- **FreeBSD (1993-)**: A direct descendant of BSD Unix, FreeBSD is known for its advanced networking features, performance, and security. It is widely used in servers and networking environments.

- **OpenBSD (1996-)**: A Unix-like OS focused on security and code correctness. It originated from the BSD lineage and is known for its emphasis on security features and code auditing.

- **NetBSD (1993-)**: Another BSD-derived operating system, NetBSD is known for its portability across a wide range of hardware platforms.

3. **Linux: Description**: An open-source OS with many distributions (distros) available. Known for its flexibility, security, and use in servers and development environments. The evolution of Linux is a story of growth from a hobbyist project to a major force in both server and desktop environments. Here's a detailed look at the major milestones in the history and evolution of Linux Kernel:



Linux™

- **Linux 0.01 (1991)**: The very first release of Linux by Linus Torvalds. It was a basic kernel that provided rudimentary multitasking capabilities and was mainly intended to be a Unix-like operating system for personal computers. It could only run-on Intel 80386 processors and was released to a small group of users.
- **Linux 0.02 - 0.11 (1991-1992)**: These early versions included various improvements such as better hardware support, enhanced stability, and the addition of fundamental features like the Virtual File System (VFS).
- **Linux 1.0 (1994)**: The first stable release of Linux. It included improved support for hardware, networking, and system calls, establishing a more solid foundation for future development.
- **Linux 2.0 (1996)**: Introduced significant improvements such as support for multiple processors (symmetric multiprocessing) and enhanced network capabilities. This version helped Linux gain credibility in server environments.
- **Linux 2.2 (1999)**: Added support for new hardware architectures, better performance, and introduced the concept of kernel modules, which allowed for dynamic loading and unloading of drivers.
- **Linux 2.4 (2001)**: Focused on performance and scalability improvements. It introduced support for USB, improved memory management, and support for larger file systems and more hardware platforms.
- **Linux 2.6 (2003)**: Brought major improvements such as better support for multi-core processors, improved security features, and more advanced file systems (e.g., ext3). This version laid the groundwork for Linux's widespread adoption in servers and enterprise environments.
- **Linux 3.x (2011)**: Marked a new major series with a focus on performance improvements, better power management, and support for new hardware. It also included features like improved virtualization support and enhanced file systems.
- **Linux 4.x (2015)**: Introduced more incremental improvements, including better hardware support, enhanced security features, and more efficient file systems. This series continued to refine the kernel and support a wide range of hardware.
- **Linux 5.x (2019)**: Added support for new hardware, enhanced security features, and improvements in performance and scalability. Key features included support for new filesystems like Btrfs and ext4 improvements, better support for ARM architecture, and various performance optimizations.
- **Linux 6.x (2022-)**: The current series includes ongoing updates and enhancements to performance, hardware support, and security. It introduces improvements such as support for new hardware technologies, enhancements in filesystems, and better security mechanisms.

- **Distributions and Impact**:
  - **Slackware (1993)**: One of the earliest distributions, focusing on simplicity and minimalism.
  - **Red Hat Linux (1994)**: Aimed at providing a more user-friendly experience and support for enterprise environments.
  - **Debian (1993)**: Known for its stability and package management system (APT), Debian became the basis for many other distributions.
  - **Ubuntu (2004)**: A derivative of Debian, Ubuntu aimed to provide an easy-to-use and accessible desktop experience, contributing significantly to Linux's desktop adoption.
  - **Ubuntu Server**: A server variant of the Ubuntu Linux distribution, known for its ease of use, stability, and extensive community support.
  - **Kali (2004)**: Kali Linux (formally known as BackTrack Linux) is an open-source Debian based Linux distribution which allows users to perform advanced penetration testing and security auditing. It has several hundred tools, configurations, and scripts with industry-specific modifications that allow users to focus on tasks such as computer forensics, reverse engineering, and vulnerability detection.
  - **Red Hat Enterprise Linux (RHEL)**: A commercial Linux distribution known for its stability and support, often used in enterprise environments.
  - **CentOS**: A free and open-source distribution derived from RHEL, used in server environments, though CentOS Stream is now the focus for new developments.
  - **Fedora (2003)**: Sponsored by Red Hat, Fedora provides cutting-edge features and serves as a testing ground for technologies that may be included in Red Hat Enterprise Linux (RHEL).
  - **Arch Linux (2002)**: Known for its simplicity, flexibility, and rolling release model, Arch Linux targets advanced users who prefer to build their system from the ground up.

4. **macOS**: macOS is Apple's operating system for Macintosh computers, known for its user-friendly interface, robust performance, and integration with Apple's ecosystem. Here's a detailed overview of macOS, its history, key features, and versions:

- **Mac OS Classic (1984-1999)**: The original Macintosh operating system, introduced with the first Macintosh computer in 1984. It featured a graphical user interface (GUI) with windows, icons, and a menu bar. Over time, it evolved through various versions but was limited by its 32-bit architecture and lack of multitasking.

- **Mac OS X (2001-2012)**: Launched as a complete overhaul of the classic Mac OS, Mac OS X combined elements from the NeXTSTEP operating system (acquired by Apple) with a modern Unix-based architecture. The key versions of Mac OS X are:
  - **Mac OS X 10.0 (Cheetah, 2001)**: The initial release, introducing the Aqua interface and fundamental features.
  - **Mac OS X 10.1 (Puma, 2001)**: Improved performance and added features like DVD playback.
  - **Mac OS X 10.4 (Tiger, 2005)**: Introduced Spotlight search, Dashboard widgets, and improved performance.
  - **Mac OS X 10.5 (Leopard, 2007)**: Added Time Machine for backups, Spaces for virtual desktops, and more.
  - **OS X (2012-2016)**: The rebranded Mac OS X, continuing to refine and modernize the OS with additional features and improvements.
  - **OS X 10.8 (Mountain Lion, 2012)**: Integrated features from iOS, such as Notification Center and Gatekeeper for security.
  - **OS X 10.9 (Mavericks, 2013)**: Introduced improvements in performance and battery life, as well as new features like Finder Tabs.
  - **OS X 10.10 (Yosemite, 2014)**: Redesigned interface with a flatter, more modern look and introduced Continuity features for seamless integration with iOS devices.
  - **OS X 10.11 (El Capitan, 2015)**: Focused on performance and stability enhancements, with features like Split View for multitasking.

- **Mac OS (2016-)**: Rebranded from OS X to macOS to align with Apple's naming conventions for its operating system.
  - **macOS 10.12 (Sierra, 2016)**: Introduced Siri to the Mac, along with Apple Pay integration and Optimized Storage.
  - **macOS 10.13 (High Sierra, 2017)**: Focused on performance and under-the-hood improvements, with a new file system (APFS) and updates to Safari.
  - **macOS 10.14 (Mojave, 2018)**: Introduced Dark Mode, Stacks for file organization, and improved privacy features.
  - **macOS 10.15 (Catalina, 2019)**: Split iTunes into separate apps (Music, Podcasts, TV), introduced Sidecar for using an iPad as a second display, and increased security with notarization of apps.
  - **macOS 11 (Big Sur, 2020)**: Marked a major redesign of the interface and introduced support for Apple Silicon (M1) Macs, along with improvements to privacy and system performance.
  - **macOS 12 (Monterey, 2021)**: Brought features like Focus modes, Universal Control for seamless multi-device interactions, and Shortcuts for automation.
  - **macOS 13 (Ventura, 2022)**: Introduced Stage Manager for better window management, new Continuity features, and enhancements to FaceTime and Spotlight.

- **Mobile Operating Systems**
  1. **Android**: Android is a widely used mobile operating system developed by Google. It powers a vast range of devices, including smartphones, tablets, wearables, and even some smart TVs and automotive systems. Here's a detailed look at Android's history, evolution, key features, and major versions:

a. **Android 1.0 (2008)**: The first commercial version of Android, released on the HTC Dream (also known as T-Mobile G1). It featured a basic set of applications including a web browser, email client, and Google's core services.
b. **Android 1.5 (Cupcake, 2009)**: Introduced several important features including an on-screen keyboard, support for third-party apps, and video recording.
c. **Android 1.6 (Donut, 2009)**: Brought updates such as support for CDMA networks, an improved user interface, and a new search interface.
d. **Android 2.0 - 2.1 (Eclair, 2009-2010)**: Included features like support for multiple accounts, better camera controls, and an updated user interface. The introduction of Google Maps Navigation was a notable addition.
e. **Android 2.2 (Froyo, 2010)**: Enhanced performance with the JIT compiler, introduced Wi-Fi tethering, and improved support for apps.
f. **Android 2.3 (Gingerbread, 2010)**: Focused on performance improvements, a refreshed UI, and better support for gaming. It also introduced support for front-facing cameras and video calls.
g. **Android 3.0 - 3.2 (Honeycomb, 2011)**: A version specifically optimized for tablets, featuring a new user interface designed for larger screens and multitasking.
h. **Android 4.0 (Ice Cream Sandwich, 2011)**: Unified the phone and tablet interfaces into a single operating system. It introduced the Holo theme, improved multitasking, and better facial recognition.
i. **Android 4.1 - 4.3 (Jelly Bean, 2012-2013)**: Improved performance with Project Butter for smoother UI transitions, introduced Google Now, and enhanced notifications.
j. **Android 4.4 (KitKat, 2013)**: Focused on performance optimizations for low-end devices and introduced a new user interface with translucent status and navigation bars.
k. **Android 5.0 - 5.1 (Lollipop, 2014-2015)**: Introduced Material Design, a new visual design language for a consistent and engaging user experience across devices. It also featured improved battery life and performance enhancements.
l. **Android 6.0 (Marshmallow, 2015)**: Introduced features like Doze mode for better battery life, app permissions management, and Google Now on Tap.
m. **Android 7.0 - 7.1 (Nougat, 2016)**: Brought features like split-screen multitasking, quick settings, and improved notifications.
n. **Android 8.0 - 8.1 (Oreo, 2017)**: Focused on performance improvements, battery optimization (Adaptive Battery), and introduced picture-in-picture mode.
o. **Android 9.0 (Pie, 2018)**: Added gesture navigation, improved AI-based features, and focused on digital wellbeing with tools like Dashboard and App Timer.
p. **Android 10 (Quince Tart, 2019)**: Introduced a system-wide dark mode, gesture navigation, and improved privacy controls.
q. **Android 11 (Red Velvet Cake, 2020)**: Focused on user control over notifications, chat bubbles, and privacy settings. Introduced improvements for 5G and foldable devices. (*android11-5.4*)
r. **Android 12 (Snow cone, 2021)**: Introduced the "Material You" design language for personalized themes, improved privacy controls, and enhanced performance. (*android12-5.10*)
s. **Android 13 (Tiramisu, 2022)**: Continued the Material You design with more customization options, improved support for large-screen devices, and enhanced privacy features. (*android13-5.15*)
t. **Android 14 (Upside down cake, 2023)**: Focused on further customization, performance optimizations, and new features for large-screen devices and foldables. (*android14-6.1*)
u. **Android 15 (Vanilla Ice Cream, 2024)**: Focused on further customization, security aspects, and performance optimizations. (*android15-6.6*)

v.

2. **iOS**: iOS is Apple's mobile operating system designed for iPhones, iPads, and iPod Touch devices. Known for its smooth performance, high security, and integration with Apple's ecosystem, iOS has evolved significantly since its inception. Here's a detailed overview of iOS, including its history, key features, and major versions:

   a. **iOS 1.0 (2007)**: Released with the original iPhone, iOS 1.0 introduced a revolutionary touch-based interface with features like the Home screen, Safari browser, and core applications such as Phone, Mail, and Contacts. It set the foundation for iOS with its intuitive user interface.

   b. **iOS 2.x (2008-2009):** Launched with the iPhone 3G, iOS 2.0 introduced the App Store, allowing third-party applications to be downloaded and installed. It also included support for Microsoft Exchange and improved mapping and GPS capabilities.

   c. **iOS 3.x (2009-2010)**: Released with the iPhone 3GS, this version brought features like copy and paste, MMS (multimedia messaging), and Spotlight search. It also improved performance and added support for new hardware features.

   d. **iOS 4.x (2010-2011)**: Introduced with the iPhone 4, iOS 4.0 brought multitasking, the introduction of the FaceTime video calling feature, and the iBooks app. It also introduced the ability to create folders to organize apps and support for the Retina display.

   e. **iOS 5.x (2011-2012)**: Released with the iPhone 4S, iOS 5.0 included features such as iCloud for cloud storage, iMessage for messaging, Notification Center, and improved integration with social networks. It also introduced the Newsstand app and Twitter integration.

   f. **iOS 6.x (2012-2013)**: Launched with the iPhone 5, iOS 6.0 brought features like Apple Maps (replacing Google Maps), Passbook (now Wallet), and improved Siri capabilities. It also introduced the ability to do FaceTime over cellular networks.

   g. **iOS 7.x (2013-2014)**: Released with the iPhone 5S and 5C, iOS 7.0 introduced a major redesign of the user interface with a flatter, more modern look. It included features like Control Center, AirDrop, and iTunes Radio. It also added improved multitasking and Safari enhancements.

h. **iOS 8.x (2014-2015)**: Introduced with the iPhone 6 and 6 Plus, iOS 8.0 added features such as HealthKit, HomeKit, and Continuity (which allows seamless transitions between Apple devices). It also improved Notifications and introduced the ability to use third-party keyboards.

i. **iOS 9.x (2015-2016)**: Launched with the iPhone 6S and 6S Plus, iOS 9.0 brought enhancements like Proactive Assistant, an improved Siri, and a revamped Notes app. It also introduced Split View multitasking for iPad and support for 3D Touch.

j. **iOS 10.x (2016-2017)**: Released with the iPhone 7 and 7 Plus, iOS 10.0 introduced a redesigned lock screen, improved Messages with stickers and effects, and a more powerful Siri with third-party app integration. It also included enhancements to Apple Music and the Home app.

k. **iOS 11.x (2017-2018)**: Launched with the iPhone X and 8, iOS 11.0 introduced a new Control Center, improved multitasking for iPad, and augmented reality (AR) support with ARKit. It also redesigned the App Store and improved the Files app.

l. **iOS 12.x (2018-2019)**: Released with the iPhone XS, XS Max, and XR, iOS 12.0 focused on performance improvements, including faster app launch times and smoother animations. It introduced Screen Time for tracking device usage and new features in Siri and Notifications.

m. **iOS 13.x (2019-2020)**: Introduced with the iPhone 11, 11 Pro, and 11 Pro Max, iOS 13.0 brought Dark Mode, a redesigned Photos app, and improvements to performance and privacy. It also introduced the Sign In with Apple feature and new gestures for iPad.

n. **iOS 14.x (2020-2021)**: Launched with the iPhone 12 series, iOS 14.0 added widgets to the Home screen, a new App Library, and enhancements to Messages and Maps. It also introduced Picture-in-Picture mode and improved privacy features.

o. **iOS 15.x (2021-2022)**: Released with the iPhone 13 series, iOS 15.0 included features such as Focus modes, redesigned FaceTime with spatial audio, and improvements to notifications and privacy. It also introduced new tools in Safari and the Weather app.

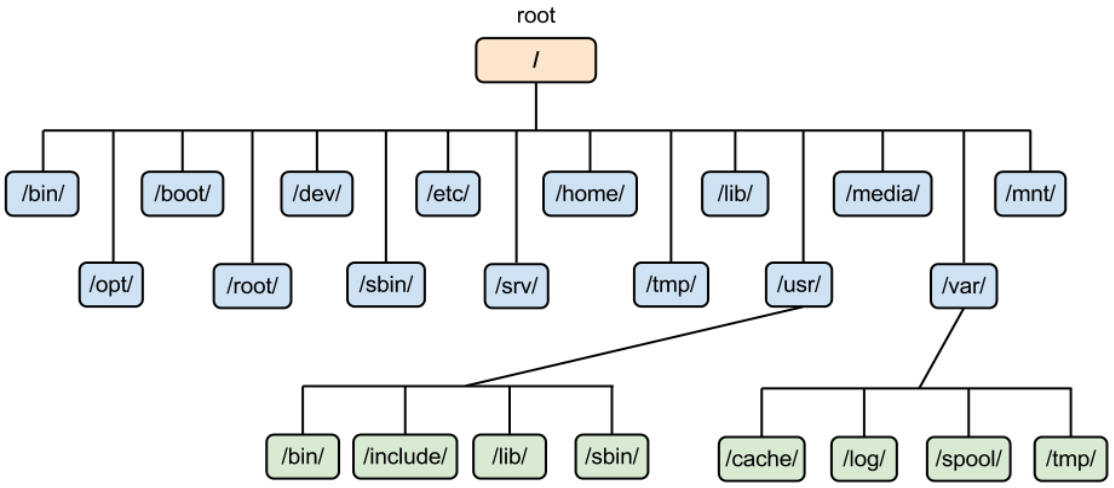
p. **iOS 16.x (2022-2023)**: Launched with the iPhone 14 series, iOS 16.0 brought significant changes such as a customizable Lock Screen, new Focus modes, and enhancements to the Messages and Mail apps. It also improved support for widgets and introduced Live Activities for real-time updates.

q. **iOS 17.x (2023-)**: The most recent version, introduced with the iPhone 15 series, iOS 17.0 continues to refine the user experience with new customization options, improved performance, and additional features aimed at enhancing productivity and connectivity.

Linux File Hierarchy Standard

The Linux File Hierarchy Standard (FHS) defines the structure and organization of directories in a Linux-based operating system. Here's an overview of important directories and how to access information about the file system hierarchy:

- **Accessing FHS Information:** Use the **`man hier`** command or visit the FHS Documentation (https://www.pathname.com/fhs/pub/fhs-2.3.pdf)for detailed information about the file system hierarchy.



- All directories and files in a Linux system are below the root directory. Here is the brief description of some important directories

Directory	Description
root (/)	All files/directories in a Linux system are below the root dir
bin	Contains essential user binaries
sbin	Contains system binaries (superuser binaries)
lib32/lib64	Libraries essential for binaries in bin and sbin
Opt	Optional application software packages
boot	Contains boot loader and kernel files
etc	Contains system-wide configuration files
home	User home directories
root	Root user's home directory
media	Mount point for removable media
mnt	Temporary mount point for filesystems
tmp	Temporary files
dev	Device files
proc	Virtual filesystem providing process and system information
sys	Virtual filesystem exposing kernel data structures
usr	System Resources directory for read-only user data
var	Variable data files such as logs, databases, and temporary files

Basic Shell Commands

Commands	Description
echo	Displays text on stdout **-n** don't append \n **-e** enables escape sequences **-E** disable interpretation of backslash escapes (default) **-c** don't produce any more output
help	Provides detail of internal commands
clear	Clears terminal screen.
type	Display information about command type (external/built-in)
env	Display environment variables
pwd	Shows absolute address of present working directory
passwd	To change user password
man	To view manual pages of different external commands for better understanding. It has 9 sections. **-k** To search string in all available man pages
who	Shows who is logged in (can be multiple)
whoami	Prints effective username (currently active)
whatis	Displays command basic purpose (one line description)
whereis	Tells *source*, *binary files* and *man page* file location of external command
which	Gives path of binary file of external and internal command
locate	To find all location of files by specified name in DB (it don't search in directory hierarchy)
find	search for files in directory hierarchy **-name** Finds by name **-size** Finds by file size (k=Kilobytes, M=Megabytes, G=Gigabytes) **-atime** access time **-ctime** status change time **-mtime** modification time **type** (**f** = normal files, **d** = directories, **s** = sockets, **p** = named pipes, **b**=block, **c**=character, l = soft link)
history	Output the last part of the history list.
info	Reads info document of external and internal command
ls	List directory contents. Read man page for detailed options
touch	Creates 1 or more empty files by touching (updating) modification and access timestamps. If file already exists it updates timestamps: **-m** For updating modification time only **-a** For updating access time only **-c** For updating status change time only
file	Displays type of file
cat	To view contents of a simple file on stdout **-n** To print line numbers as well **-s** To suppress repeated blank lines **-b** To number only non-empty lines (overrides -n)
tac	To view contents of file in reverse (last line 1st)
more	To view contents of large files one screen at a time. It also displays the % of the file displayed, and we can't move back up in it. **ENTER** To move down line-by-line **SPACE** To move down one screen To search *"str"* in file. **/str** Press 'n' to find next Press 'N' to find previous
less	To view contents of large files one page at a time, better than m

	Navigation Arrow keys, Pgup, Pgdwn, ENTER, SPACE (acts as Pgdwn), HOME, END To search *"str"* in file. **/str** Press 'n' to find next Press 'N' to find previous **g, G** 'g' moves to start and 'G' moves to end
head	Displays 1st ten lines **-n** To view 1st n lines
tail	Displays last ten lines **-n** To view last n lines **-f** Output appended data as the file grows **-c** specifies that we want to read n characters not lines
shutdown	To shut-down or restart **shutdown now** Shut-down immediately **shutdown -r now** Restarts immediately **shutdown +0** Shut-down immediately **shutdown +m** Shutdown after m minutes ('+' is optional) **shutdown 22:30** Shut-down at 22:30
cp	To copy files/directories **-p** Preserve permissions while copying **-r** For directories **e.g:** *cp f1 f2* #f1 is source file and f2 is target-file
rm	To remove files/directories **-f** ignore non-existent files and arguments, never prompt **-r** For directories **-i** For confirmation prompt **e.g:** *rm f1 f2* #will delete both f1 & f2
mv	To move files/directories **-i** For confirmation prompt **e.g:** *mv f1 f2* #will move f1 to f2 (also used to rename file)
mkdir	To make directory file **-m** set file mode (as in chmod) **-p** no error if existing, make parent directories as needed
rmdir	To remove directory file **-p** remove DIRECTORY and its ancestors; e.g., 'rmdir -p a/b/c' is similar to 'rmdir a/b/c a/b a'
uname	Prints OS name on stdout **-a** Shows detailed OS info
wc	print number of lines, word, char counts for each file (Ctrl+D to quit) **-l** for lines only **-w** for words only **-m** for character count only **-c** byte count
grep	("General Regular expression Processor") Print lines matching or not matching a pattern. **-i** for case-insensitive search **-v** for negation **-c** print count of lines matching/not matching (for -v)
tar cvf	Create tar file in PWD (1st pass name for archive file then directories and files to archive)
tar tvf	To view .tar files not extract them
tar xvf	To extract .tar files in present working directory
tar xzf	To unzip and extract .tar files in present working directory
gzip	To zip files. Original file is replaced by zip file. (extension = .gz) **Note:** We can zip tar files to obtain "tar balls"(.tar.gz), commonly used for software distribution
gunzip	To unzip files

Process Related Shell Commands

Commands	Description
jobs	Lists currently running jobs and their status **-l** lists process IDs in addition to the normal information **-p** lists process IDs only
fg [pid] **fg %Jid**	To move a background process to foreground using its PID or Job ID
bg [pid] **bg %Jid**	List background jobs or move a process to back ground using its PID or Job ID
kill	Send a signal to a job. (default SIGTERM) **kill -n signum pid** **-l** List the signal names
ps	Report a snapshot of current process (4 columns) See /proc/ directory **-A or -ax** to show all running process **-u [username]** List processes by user (displays 11 columns) **-l** displays 14 columns (long listing) **-a** Select all processes except both session and processes not associated with a terminal.
top	Shows detail process real-time info of top-20 processes, like task manager. Interactive, (continuously refreshes after every **3sec).** **Press:** **h** for help **n** to display only [n] processes (0=unlimited) **u** to display processes of particular user **s** to change refresh time **k** to send signal (it 1st asks for PID then signal number/name)
free	Displays amount of free and used memory in the system (6 columns) **-k** in KB (default) **-m** in MB **-b** in bytes **-g** in GB **--tera** in tera
uptime	It shows system time, uptime, number of logged in users, load average for last 1,5 and 15 minutes respectively.
watch	Executes a program periodically, showing output in full screen (refreshes every 2sec)

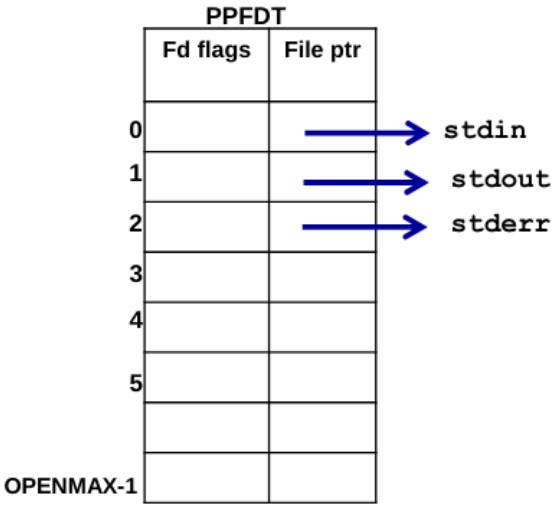
User Management Related Commands

- **Users**: Each user in a Linux system is assigned a unique User ID (UID). Usernames and UIDs are stored in the `/etc/passwd` file. Users cannot read, write, or execute files owned by others without proper permission.
- **Groups**: Users are also assigned to groups with unique Group IDs (GIDs), stored in the `/etc/group` file. Each user has a private group by default but can belong to other groups for additional access. Members of a group can share files that belong to that group.
- **Three Classes of Users**
  - **User/Owner**: The owner is the user who created the file. Any file you create is owned by you.
  - **Group**: The owner can grant access to the file to members of a designated group.
  - **Others**: The owner can also provide access to all other users on the system.

Commands	Description
adduser	Interactive and recommended way to add a user, as compared to useradd command which is a low level command. (***sudo adduser user1***)
deluser	Users we want to delete should be logged out. It doesn't delete user HOME Dir
userdel	Low level also deletes HOME directory and files. **-r** to delete home dir and associated files as well of this user. ***sudo userdel -r user1***
usermod	To modify user info **e.g.:** ***usermod -a -G gp2 user1*** (makes user1 member of gp2) If we don't use -a then it will not append new user but overwrite it (that is all previous group members will be removed) -c to change personal info column value **sudo usermod -c "Personel Info" user2** -s to change default user shell **sudo usermod -s /bin/sh user2** -l to change username **sudo usermod -l user007 user2** (new name 1st) -d to change Home Directory -L to lock user (this user can't log in) -U to unlock locked user -g to change primary group -G to change secondary group
groupadd	To add a new group. (***sudo groupadd gp1***)
groupmod	To modify group. **-n** is used for changing group name.
groupdel	To delete group. (***sudo groupdel name***)
chage	Used to change password expiry info of a user (***sudo chage user2***). **-l** to view just password setting of particular users
chsh	Used to change default user shell
chfn	Used to change user personal info
finger	shows user info in detail (may have to install it manually)
id	it displays ID (UID) and primary GIDs and groups you belong to gid=primary group, groups = Secondary group
su	(switch user) We can use it to login using any username if we know its password (e.g: su -root) Using '-' will also give you the target user environment. You will find yourself in the target user HOME Directory and his default login shell
visudo	used to edit /etc/sudoers file, containing the users who can do sudo

I/O Redirection

1. **Standard descriptors in UNIX**
   - **Standard Input (stdin)**: Represented by file descriptor 0. By default, input to a program is read from the user terminal (keyboard) if no file name is specified.
   - **Standard Output (stdout)**: Represented by file descriptor 1. Output from a program typically goes to the user terminal (monitor) if no file name is specified.
   - **Standard Error (stderr)**: Represented by file descriptor 2. Error messages generated by a program are sent to the user terminal by default if no file name is specified.

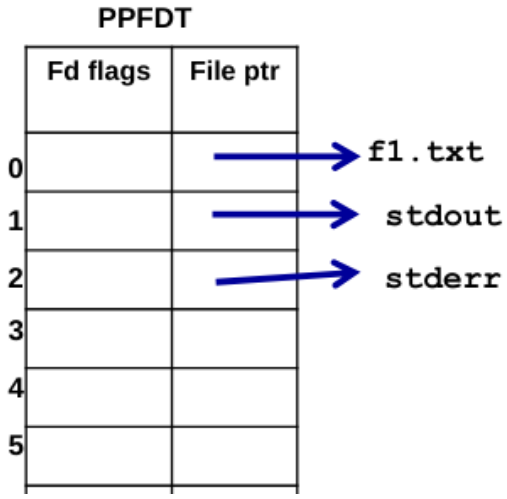


   These standard descriptors are crucial for I/O operations in UNIX/Linux systems. They are referenced by system calls to interact with files. By default, if a file name is not provided, these standard descriptors are used for input, output, and error handling, providing a convenient and consistent interface for process communication.

2. **I/O Redirection:** IO redirection is a powerful feature in UNIX/Linux systems that allows users to control where the input, output, and error messages of commands are directed. Here's how redirection works:

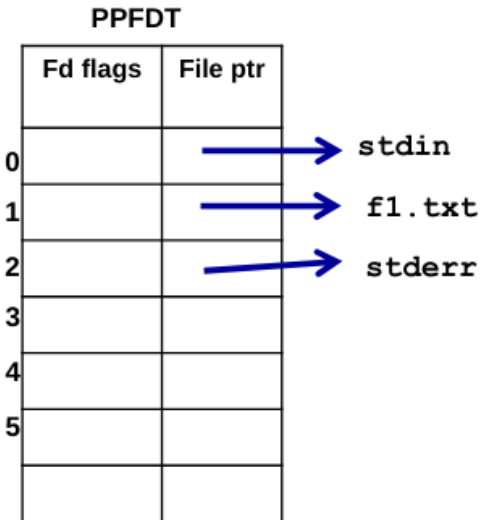
   - **Redirecting Input of a Command (0<)**
     - By default, commands like cat and sort read their input from the standard input (keyboard).
     - To redirect input from a file instead of the keyboard, use the < symbol followed by the file name.

```
$ cat < f1.txt
```



- **Redirecting Output of a Command (1>)**
  - By default, commands like cat and sort send their output to the standard output (display screen).
  - To redirect output to a file instead of the display screen, use the > symbol followed by the file name.

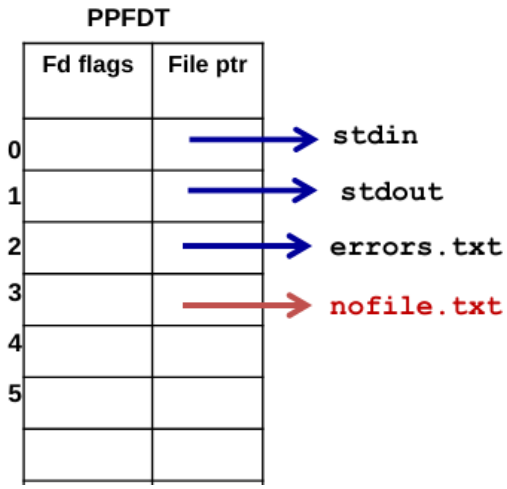
```
$  cat > f1.txt # cat output will be redirected to f1.txt
```



- **Redirecting Error of a Command (2>)**
  - By default, all commands send their error messages to the standard error (display screen).
  - To redirect error messages to a file instead of the display screen, use the 2> symbol followed by the file name.

```
$ cat nofile.txt 2> errors.txt
```

  - This command will redirect the error messages produced by cat to the file errors.txt.



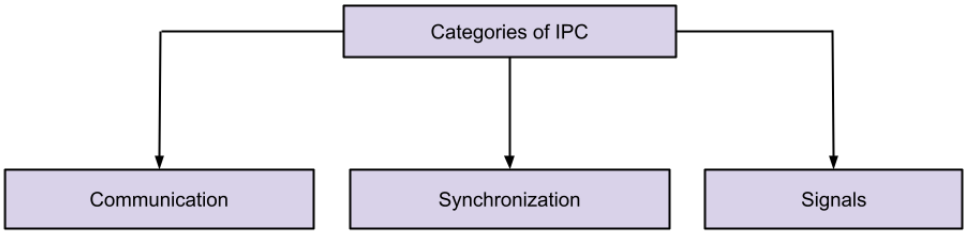
Reference video: https://www.youtube.com/watch?v=ik6TvPquVk8&list=PL7B2bn3G_wfBuJ_WtHADcXC44piWLRzr8&index=9&pp=iAQB

IPC Mechanism

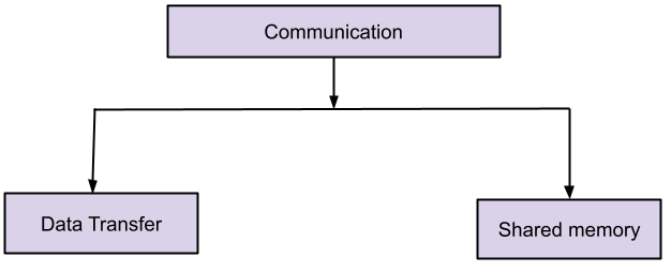
- **Independent Process:**
  - A process that cannot influence or be influenced by the execution of another process.
  - Does not share data with other processes.

- **Cooperating Process:**
  - A process that can influence or be influenced by the execution of another process.
  - Shares data with other processes.
  - **Advantages of Cooperating Processes:**
    - **Information Sharing**: Processes can exchange data to collaborate on tasks.
    - **Computation Speedup**: Multiple processes can work on different parts of a task simultaneously, reducing overall computation time.
    - **Modularity**: Breaking down complex tasks into smaller, manageable processes.
    - **Convenience**: Easier to design and manage smaller processes that work together.

Taxonomy of Inter-Process Communication (IPC)



- **Communication**
  Communication facilities are concerned with the exchange of data among cooperating processes. There are various mechanisms through which processes can communicate and synchronize their actions.



  - **Data Transfer**: Involves one process writing data to an IPC facility and another process reading that data. Characteristics of Data Transfer are:
    - **Destructive Read Semantics**: Once data is read, it is no longer available in the IPC facility.
    - **Implicit Synchronization**: Synchronization between the reader and writer is managed by the IPC mechanism itself, without requiring additional efforts from the programmer.
    - **Examples**: Pipes, fifos, and message queues.

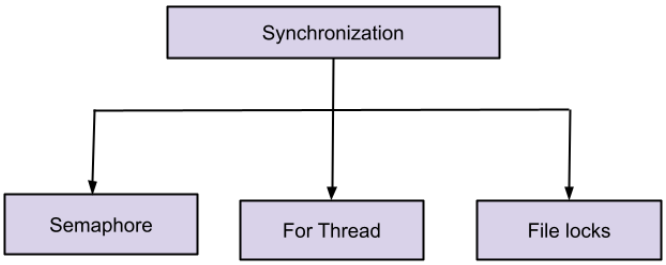
- **Shared Memory**: Establishes a region of memory that is shared among cooperating processes. Characteristics of Shared memory are:
  - **Non-Destructive Read**: Reading data from shared memory does not remove the data, allowing multiple processes to read the same data.
  - **Explicit Synchronization**: Synchronization is not handled by the IPC mechanism; instead, the programmer must ensure proper synchronization to avoid race conditions.
  - **Examples**: Shared memory segments.

- **Note:** For practical implementation of pipes, watch the following video:

  https://www.youtube.com/watch?v=VA8FEgahi1Y&list=PL7B2bn3G_wfC-mRpG7cxJMnGWdPAQTViW&index=26&pp=iAQB

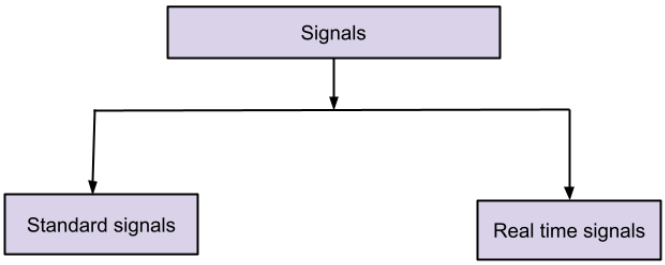
- ## Synchronization

Synchronization facilities are focused on coordinating the actions of cooperating processes to ensure proper sequencing and timing of their operations. This is crucial to avoid conflicts and ensure data integrity.



1. **Mutexes and Semaphores**:
   - **Description**: Used to manage access to shared resources by multiple processes, ensuring that only one process can access a resource at a time or that certain conditions are met before access is granted.
   - **Examples**: POSIX semaphores, System V semaphores.
2. **Monitors and Condition Variables**:
   - **Description**: Higher-level synchronization constructs that provide a structured way to manage complex synchronization scenarios.
   - **Examples**: POSIX condition variables.
3. **Barriers**:
   - **Description**: Synchronization points where multiple processes or threads must all arrive before any can proceed.
   - **Examples**: POSIX barriers.

- **Signals:**
  Although primarily used for other purposes, signals can also serve as synchronization primitives in certain circumstances.



  - **Description**: Signals are a limited form of IPC used to notify a process that a specific event has occurred.
  - **Characteristics**:
    - **Asynchronous Notification**: Allows processes to be interrupted and respond to specific events immediately.
    - **Limited Synchronization**: Can be used to coordinate actions between processes, but with less precision and control compared to other synchronization mechanisms.
  - **Examples**: SIGINT, SIGTERM, SIGCHLD.

  IPC mechanisms enable processes to cooperate by sharing data and resources. The choice of IPC mechanism depends on the specific requirements of the application, such as the need for synchronization, the speed of communication, and the complexity of the data being shared. By leveraging these mechanisms, processes can work together efficiently, leading to improved performance and modularity in application design.

File Permissions Related Commands

Every file stored on the file system of a computer must be secured. No unauthorize person should be able to access it. UNIX traditional privilege scheme defines what actions a user or process can perform on files, directories, and other system resources. The following screenshot shows the long listing of a directory contents having seven different file types in Linux, with their

`inode:perm:linkcount:owner:group:size:time:name`

```
┌──(kali㉿kali)-[~/temp]
└─$ ls -li
total 296
1732903 -rw-rw-r-- 1 kali kali   280848 Aug 30 01:40 B2E
1736228 brw-r--r-- 1 root root 555, 666 Sep 16 22:13 blkspecial
1736253 crw-r--r-- 1 root root 333, 444 Sep 16 22:13 charspecial
1736262 drwxrwxr-x 2 kali kali     4096 Sep 16 22:14 dir1
1733221 -rw-rw-r-- 1 kali kali        6 Sep 16 22:10 f1.txt
1719492 -rw-rw-r-- 2 kali kali      613 Sep 16 22:10 hello.c
1719492 -rw-rw-r-- 2 kali kali      613 Sep 16 22:10 hltohello.c
1736266 prw-rw-r-- 1 kali kali        0 Sep 16 22:16 namedpipe
1733238 -rw-r----- 1 root root       53 Sep 16 22:08 secret.txt
1734111 lrwxrwxrwx 1 kali kali        6 Sep 16 22:11 sltof1.txt → f1.txt
1736342 srwxrwxr-x 1 kali kali        0 Sep 16 22:23 socketfile
```

- **Protection in Linux**
  - **Users**: Each user in a Linux system is assigned a unique User ID (UID). Usernames and UIDs are stored in the `/etc/passwd` file. Users cannot read, write, or execute files owned by others without proper permission.
  - **Groups**: Users are also assigned to groups with unique Group IDs (GIDs), stored in the `/etc/group` file. Each user has a private group by default but can belong to other groups for additional access. Members of a group can share files that belong to that group.
- **Three Classes of Users**
  - **User/Owner**: The owner is the user who created the file. Any file you create is owned by you.
  - **Group**: The owner can grant access to the file to members of a designated group.
  - **Others**: The owner can also provide access to all other users on the system.
- **File and Directory permissions in Linux:** File permissions in UNIX/Linux have different implications for files and directories:
  - **For Files**
    - **Read (r)**:
      - Allows users to open and read the file's contents.
      - Commands used: `less, more, head, tail, cat, grep, sort, view`
    - **Write (w)**:
      - Allows users to open and modify the file's contents.
      - Editors used: `vi, vim, peco, nano`
    - **Execute (x)**:

- Allows users to run the file as a program or script.
- Example: `./script.sh`

- **For Directories**
  - **Read (r)**:
    - Allows users to list the contents of the directory.
    - Command used: `ls.`
  - **Write (w)**:
    - Allows users to create new files and directories, as well as delete files they own within the directory.
    - Commands used: `mkdir, touch, cp, rm.`
  - **Execute (x)**:
    - Allows users to access the directory and perform actions within it, such as searching or changing into it. Without Execute. Permissions, read/write permissions on directory are ineffective.
    - Command used: `cd`.

- **Changing File Ownership:** The `chown` command is used to change ownership of files and directories.
  - To change the owner of a file `myfile.txt` to arif:

```
# chown arif myfile.txt
```

  - To change the owner to arif and the group to developers for a directory `mydir` and its contents:

```
# chown -R arif:developers mydir
```

- **Changing File Permissions:** The `chmod` command is used to change existing permissions of files and directories.
  - Set read (r), write (w), execute (x) permissions for the owner, and read and execute permissions for the group and others on a file `script.sh`

```
# chmod 755 script.sh
```

    or

```
# chmod u=rwx,g=rx,o=rx script.sh
```

  - Add execute permission for the owner of a file `myfile.txt`

```
# chmod u+x myfile.txt
```

  - To remove write permission for the group from a directory mydir:

```
# chmod g-w mydir
```

  - To set the SUID bit on a program `myprog`:

```
# chmod u+s myprog
```

- **Special permissions in Linux**
  - **SUID bit**
    i. When a program has this bit set, it runs with the privileges of the program's owner.
    ii. The SUID bit is typically set for executable programs.
    iii. It is denoted by an 's' in the owner's execute permission or a capital 'S' if the owner's execute permission is off.
    iv. For instance, the `passwd` program has its SUID bit set, allowing users to modify the /etc/shadow file owned by root.
    v. To identify executable files with the SUID bit set:

```
$ find / -type f -perm -u=s -ls 2> /dev/null
```

- **SGID bit**
  i. When a program has this bit set, it runs with the privileges of the program's group.
  ii. The SGID bit is set for both executable programs and directories.
  iii. It is indicated by an 's' in the group's execute permission or a capital 'S' if the group's execute permission is off.
  iv. For example, the `chage` program has its SGID bit set, allowing users to modify the /etc/shadow file.
  v. To find executable files with the SGID bit set:

```
$ find / -type f -perm -g=s -ls 2> /dev/null
```

  vi. SGID bit on directories is useful in shared group environments. Any file created within a directory with the SGID bit set inherits the group membership of that directory.
- **Sticky bit (On Directories):**
  i. If a directory with full permissions has this bit set, users cannot delete each other's files.
  ii. It is indicated by a 't' in the others execute permission or a capital 'T' if the others' execute permission is off.
  iii. For example, the /tmp directory has its sticky bit set, preventing users from deleting each other's files.
  iv. To identify directories with the sticky bit set:

```
$ find / -type d -perm -o=t -ls 2> /dev/null
```

How Operating Systems Implement Security

Operating systems implement security through a variety of mechanisms designed to protect data, prevent unauthorized access, and ensure system integrity. OSs continuously evolve to address emerging threats and vulnerabilities, integrating new technologies and approaches to maintain robust security. Some of the key methods are mentioned below:

- **User Authentication:**
  - Passwords: A common method requiring users to provide a secret passphrase.
  - Multi-Factor Authentication (MFA): Combines something you know (password), something you have (security token), and something you are (biometric) to enhance security.
- **Access Control Models:**
  - Discretionary Access Control (DAC): In DAC, owner of the object decides who all have what all kinds of access (rwx) on his object. This is the default for most Operating Systems.
  - Mandatory Access Control (MAC): In MAC, system decides who all have what all kinds of access (rwx) on the objects. Every object has a security classification associated with it (e.g, top secret, secret, confidential, restricted, unclassified). Every user has a security clearance to access a specific class of object.
  - Role-Based Access Control (RBAC): Users are assigned roles, and roles have specific permissions, simplifying management and enhancing security.
- **Access Control:**
  - UNIX Traditional privilege Scheme: Defines what actions a user or process can perform on files, directories, and other system resources. Permissions are usually categorized as read, write, and execute. The UID is compared with the object owner ID, group ID and the permissions are granted accordingly else other permissions are granted to the user.
  - Access Control Lists (ACLs): Provide a more granular control over access permissions by associating specific permissions with individual users or groups.
- **Encryption:**
  - Data Encryption: Protects data at rest (stored data) and in transit (data being transferred across networks) using algorithms like AES (Advanced Encryption Standard).
  - File System Encryption: Encrypts the entire file system to protect data from unauthorized access.
- **Kernel Security:**
  - Privilege Separation: Ensures that the kernel and user space are separated to prevent unauthorized access to kernel-level operations.
  - Secure Boot: Verifies the integrity of the operating system at startup to prevent boot-level malware.
  - Data Execution Prevention (DEP): Prevents code from being executed in certain regions of memory.
    - Address Space Layout Randomization (ASLR): Randomizes memory addresses used by system and application processes to make it harder for attackers to predict targets.
- **Auditing and Logging:**
  - Activity Logs: Record events and user actions to monitor and detect suspicious activities.
    - Audit Trails: Provide a detailed history of system access and changes to help in forensic investigations and compliance.
- **Firewalls and Network Security:**
  - Firewalls: Control incoming and outgoing network traffic based on security rules.
  - Intrusion Detection Systems (IDS): Monitor network and system activities for malicious activities or policy violations.

- **Isolation and Sandboxing:**
    - Process Isolation: Ensures that processes run in their own memory space and cannot interfere with each other.
        - Sandboxing: Restricts the execution of code within a controlled environment to prevent it from affecting the rest of the system.
- **Virtualization and Containerization:**
    - Virtual Machines (VMs): Isolate applications and services in separate VMs to reduce the risk of compromise affecting the entire system.
        - Containers: Offer a lightweight form of isolation for running applications in segregated environments.

How and Where Linux OS Store Passwords?

- Across different operating systems, password security is primarily handled through hashing algorithms and encryption techniques to protect against unauthorized access. While the specifics can vary—such as the use of NTLM in Windows, SHA-512 in Linux, or AES-256 in macOS—the common goal is to ensure that password data is securely managed and protected from compromise. On almost all modern Linux distributions, user passwords are hashed and stored in the **`/etc/shadow`** file. Access to this file is restricted to root users, and additional security measures like disk encryption and Pluggable Authentication Modules (PAM) modules help to protect and manage authentication securely.
- The early version of UNIX in 1971, used to store the hashed value of passwords in the second field of the world readable **`/etc/passwd`** file. This was because this file contains user related information other than passwords and many applications require that information to function properly. In the later versions of UNIX, and in today's Linux distros, the hashed password is saved in the **`/etc/shadow`** file, which is readable only by super users.
- The screenshot of the contents of the `/etc/passwd` file on my Kali Linux machine, with its seven self-explanatory fields is shown below:

```
loginname:en_passwd:UID:GID:GECOS:homedir:shell
```

```
┌──(kali㉿kali)-[~]
└─$ cat /etc/passwd | grep kali
kali:x:1000:1000:,,,:/home/kali:/usr/bin/zsh
```

- Similarly, the screenshot of the contents of the **`/etc/shadow`** file on my Kali Linux machine, with its nine fields is also shown below. Every row contains one record having nine colon separated fields:

```
┌──(kali㉿kali)-[~]
└─$ sudo cat /etc/shadow | grep kali
[sudo] password for kali:
kali:$y$j9T$hRLNczNaMnWjaIlEM3Gu51$AX7a7GZxp59zDXxaYo7Wb8f9C7FSTNTMc552LkSYDz/:19870:0:99999:7:::
```

Every row contains one record having nine colon separated fields. Here is the breakdown of each field:

`user:$y$salt$hash:lastchanged:min:max:warn:inactive:expire:`

1. **`kali:`** Username
2. **`$ID$salt$hash:`** Indicates the hashing algorithm used, salt and hash
   - **`ID:`** This is a string that indicates the hashing algorithm used. Common values are:
     - `$1$` is MD5.
     - `$2a$` or `$2y$` is blowfish.
     - `$5$` for SHA-256.
     - `$6$` for SHA-512.
     - `$y$` for yescrypt.
   - **`Salt:`** It prevent two users with the same password from having duplicate entries in the /etc/shadow file. If user1 and user2 both has set their passwords as 'pucit', their encrypted passwords will be different because their salts will be different.
   - **`Hash:`** The salt and the un-encrypted password are combined and encrypted using the specific algorithm to generate the encrypted hash of the password that is saved here.
3. **`Lastchanged (19870):`** The date of the last password change, expressed as the number of days since Jan 01 1970 (UNIX epoch). If this field is empty, it might imply that the user hasn't changed their password since the account was created.
4. **`Minimum (0):`** Minimum password age specifies the minimum number of days required between password changes. An empty field or a zero means that password aging features are disabled.
5. **`Maximum (99999):`** Maximum password age specifies the maximum number of days that a password is valid. After this period, the user will be prompted to change their password.
6. **`Warn (7):`** Password warning period specifies the number of days before the password expires that the user will start receiving warnings about the upcoming expiration.
7. **`Inactive:`** Password inactivity period specifies the number of days after a password expires during which the account remains usable. After this period, the account will be disabled if the password isn't changed.
8. **`Expire:`** Account expiration date specifies the number of days since January 1, 1970, when the user account will expire. After this date, the user account will be disabled. If this field is empty, the account does not have an expiration date.
9. **Reserve:** This field is reserved for future use and is generally left empty in most implementations.

- The login process in the early UNIX versions was as follows:

```
if (md5($submitted_passowrd) == $stored_password_hash)
        login()
else
        display_msg("Wrong Password")
```

Note: Many websites and online services also require users to login using passwords. They typically store their passwords in their databases and may be using non-standardized ways.

Some Basic Cryptographic Terms and Linux Tools to use them https://cyberchef.io/

A **Man-in-the-Middle (MitM) attack** is a type of cyberattack where an attacker secretly intercepts and possibly alters the communication between two parties who believe they are directly communicating with each other. The attacker sits between the two parties, relaying messages and potentially capturing sensitive information like login credentials, personal data, or financial details. In a typical MitM attack, the attacker can:

1. **Eavesdrop** on the communication to intercept data and attacks Confidentiality.
2. **Alter** the communication by modifying messages and attacks Integrity.
3. **Impersonate** one or both of the communicating parties, also known as masquerading/spoofing.

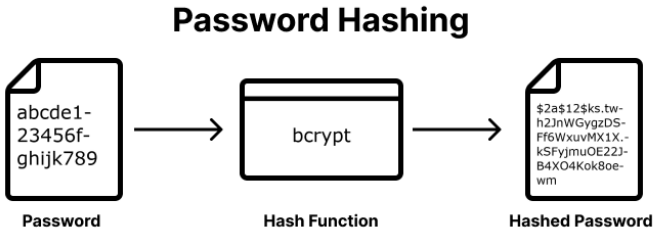
Before we see how hashes are generated, let me first describe some basic cryptographic terms:

- **Encoding:** It is a process of converting data to another format to be used on different device or system using a publicly known algorithm and without the need of a key to encode or decode. The main goal is not security but ensuring that data can be correctly interpreted or transferred. One of the main characteristics of encoding is its reversibility, i.e., the encoded data can be easily decoded back to its original form if you know the encoding scheme used. Some examples of different encoding schemes are octal, hex, base64, ASCII, UTF-8, and so on

    - **Base 64 Encoding:** Encodes and decodes data in Base64 format, which is commonly used to encode binary data as ASCII text.
        - Encode a string to base64
          ```
          echo -n "Hello World" | base64
          ```
        - Decode a string from base64
          ```
          echo "SGVsbG8gV29ybGQ=" | base64 -d
          ```
        - Encode a file to base64
          ```
          base64 inputfile.txt > encodedfile.txt
          ```
        - Decode a Base64 file
          ```
          base64 -d encodedfile.txt > decodedfile.txt
          ```

    - **xxd Encoding:** Creates a hex dump of a file or standard input, or converts a hex dump back into binary
        - Create hex dump of a string and display on screen
          ```
          echo -n "Hello World" | xxd
          ```
        - Create hex dump of a file and save the output in a file hello.dump
          ```
          xxd hello.c > hello.dump
          ```
        - Convert a hexdump back to its ASCII representation
          ```
          xxd -r hello.dump
          ```

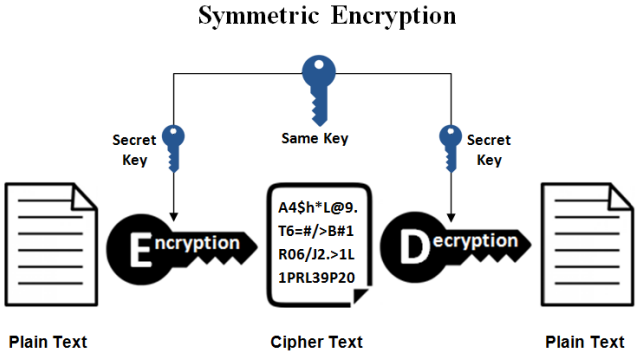
- **Hashing:** Hashing is used to verify data integrity. It transforms data into a fixed-size string of characters, which is typically a digest. Hashing is used to store passwords and also ensuring data integrity while downloading files from websites. The two main characteristics of good hashing algorithms are that they are irreversible (one way function) and deterministic (same input will always produce the same hash output)
    - **Hashing using OpenSSL:** OpenSSL is a cryptography toolkit implementing the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) network protocols and related cryptography standards required by them. The **`openssl`** is a command line program for using the various cryptography functions of OpenSSL's crypto library from the shell. It has many uses, but most common are:
        - Calculation of Hashes
        - Symmetric Encryption
        - Public/Private key generation
        - Asymmetric Encryption.




```
$ openssl version
OpenSSL 3.3.3 4 Jun 2024 (Library:OpenSSL 3.2.2 4 Jun 2024

$ openssl dgst -help
$ echo "password" > file.txt

$ openssl dgst -md5 ./file.txt
MD5(file.txt)= 286755fad04869ca523320acce0dc6a4      (128 bits hash)

$ echo "password" | openssl dgst -md5
MD5(stdin)= 286755fad04869ca523320acce0dc6a4         (128 bits hash)
```

The `openssl` command has many sub-commands like `version`, `base64`, `enc`, `dgst`, `ciphers`, `dsa`, `rsa`, and so on. The `openssl dgst` command (Message Digest Calculation), we can use many different hashing algorithms. In the above examples I have used the algorithm message digest 5 (`md5`) that will create a 128 bit hash. You can use the `sha1` (160), `sha256` (256 bit hash), `sha384` (384 bit hash), `sha512` (512 bit hash), and so on.

- **Encryption:** It is used to protect data confidentiality. It is a process of scrambling data to make it decipherable only by the intended recipient. The goal is to ensure that only those with the correct decryption key can read or access the original data. One of the main characteristics of encoding is its reversibility, i.e., the encrypted data can be decrypted back to its original form if you have the decryption key. Some examples of different encoding schemes are octal, hex, base64, ASCII, UTF-8, and so on. There are two major categories of encryption that are taught in cryptography, namely symmetric encryption and asymmetric encryption.
    - **Symmetric Encryption:** Symmetric encryption uses the same key for both encryption and decryption. This means that both parties must share the same secret key. The security of symmetric encryption depends on keeping this key secret. Some famous symmetric encryption algorithms are:
        - Data Encryption Standard (DES with 56 bits key)
        - Triple Data Encryption Standard (3DES with 168 bits key)
        - Blowfish (32 bit to 448 bit)
        - Advance Encryption Standard (AES with 128, 192, 256 bits key)

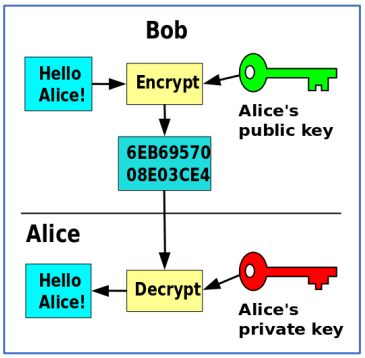



```
$ openssl enc -help

$ openssl enc -list

$ echo "hello world" > f1.txt

$ openssl enc -des -in f1.txt -out encrypted_des -K 0123456789abcdef -iv
a0b0c1d4e5f27891

$ cat encrypted_des

$ openssl enc -des -d -in encrypted_des -out decrypted_des -K
0123456789abcdef -iv a0b0c1d4e5f27891

$ cat decrypted_des
```

Note:

- The **-K** argument specifies the encryption key as a hexadecimal string. DES requires a key that is exactly 8 bytes long (16 hexadecimal characters). If you provide a key that is longer or shorter, OpenSSL will either truncate it or pad it, potentially compromising security.
- The **-iv** argument specifies the initialization vector (IV) for certain modes of operation (like CBC - Cipher Block Chaining). It ensures that the same plaintext block will encrypt to different ciphertext blocks, providing additional security. For DES, the IV must also be 8 bytes long (16 hexadecimal characters).
- For the decryption process, one has to provide the same key and initialization vector that was used during the encoding process
- Limitation of symmetric encryption is, "How the sender and receiver share the encryption key?"

- **Asymmetric Encryption:** Asymmetric encryption uses a pair of keys: a public key for encryption and a private key for decryption. The keys are mathematically linked, but knowing the public key does not help in deriving the private key. Some famous asymmetric encryption algorithms are Rivest-Shamir-Adleman (RSA), Digital Signature Standard (DSS), and Pretty Good Privacy (PGP).



    - **Rivest-Shamir-Adleman (RSA):** is the first public key cryptosystem and widely used for secure data transmission. The key length is typically 2048 4096 bits for security. It can be used for both encryption and digital signature.
        - **Step 1: (Generate an RSA Key pair)**
            - Choose Two Secret Numbers**:** Start by selecting two large, secret prime numbers. These will be the foundation of the keys.
            - Create a Public Key**:** From these two numbers, generate a public key that anyone can use to encrypt messages meant for you. This public key consists of a large number (the product of the two primes) and another number that helps with encryption.
            - Create a Private Key**:** Also generate a private key, which is kept secret. This key is used to decrypt messages that were encrypted with your public key.

```
$openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:2048 -out private_key.pem
$openssl rsa -pubout -in private_key.pem -out public_key.pem
```

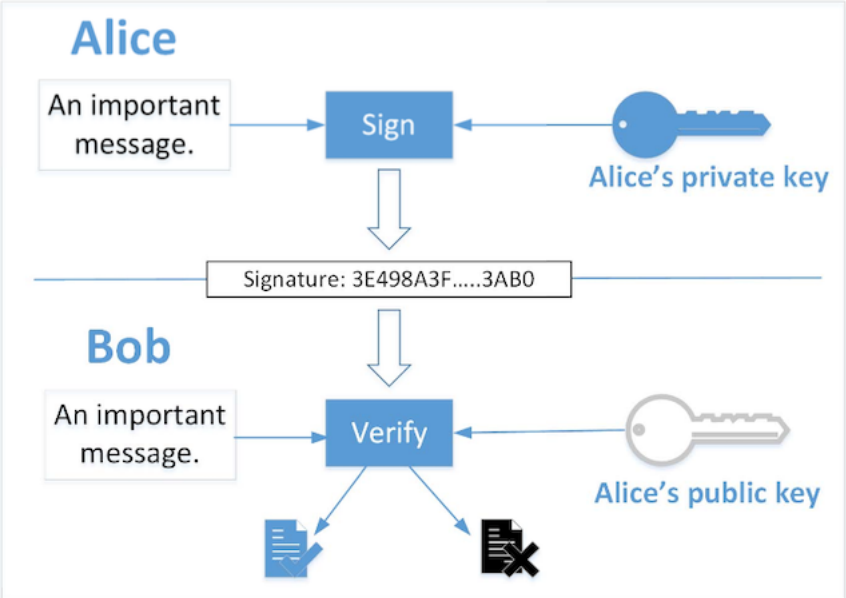
- **Step 2: (Sender Encrypts with the Public Key of Receiver)**

```
$openssl pkeyutl -encrypt -inkey public_key.pem -pubin -in f1.txt -out encrypted.bin
```

- **Step 3: (Receiver Decrypts with the his/her Private Key)**

```
$openssl pkeyutl -decrypt -inkey private_key.pem -in encrypt.bin -out decrypted.txt
```

Digital Signatures: Ensures that MiTM cannot do impersonation. It is an electronic mean of verifying some ones identity. It also use two keys. The sender signs the document using his/her private key and the recipient verifies it using the sender public key which answers in yes or No.



- **Step 1: (Generate an RSA Key pair)**
- **Step 2: (Sender Generate Signature with his/her Private Key)**
- **Step 3: (Receiver Verifies Signature. With sender's Public Key)**

- Limitation is what if the sender or receiver denies that he/she has received the message. The sender may change his/her private:public key pair. Solution is Digital Certificates.

Digital Certificates: It is similar to Digital Signatures, but involves 3rd parties (Certificate Authorities) like GoDaddy, DigiCert, GlobalSign. Digital certificates are used to authenticate the identity of (individual, organization, webservers) and facilitate key exchange for encryption and verification of digital signature. You get your Digital Certificate from a CA and make it public. Rom this digital certificate a browser can extract the public key and use it to encrypt the data to be sent to server. Now you cannot say that I have not sent that message.

Shell scripting

Shells are typically interactive, meaning they accept and execute commands from users in real-time. However, there are situations where you might need to execute a series of commands routinely. To avoid the repetitive task of typing these commands each time, you can write them in a file and execute that file in the shell. These files are called Shell Scripts or Shell Programs, like batch files in MS-DOS. Shell scripts typically have a .sh file extension, such as `myscript.sh`. A shell script consists of the following elements:

- **Shell Keywords**: Keywords like if, else, break, etc.
- **Shell Commands**: Commands such as cd, ls, echo, pwd, touch, etc.
- **Functions**: Groupings of commands that can be executed as a unit.
- **Control Flow**: Constructs for managing the flow of execution, including if..then..else, case, and loops like for, while, and until.

- **Basic Structure of a Shell Script**

```
#!/bin/bash
echo "Hello, World!"
```

- **Variables**

```
NAME="John"
echo "Hello, $NAME"
```

- **Reading input from the user**:

```
echo "Enter your name:"
read NAME
echo "Hello, $NAME"
```

- **If…else Structure**

```
if [ "$NAME" == "John" ]; then
  echo "Your name is John"
else
  echo "Your name is not John"
fi
```

- **Loops**:

```
for i in 1 2 3; do
  echo "Number $i"
done

COUNT=1
while [ $COUNT -le 3 ];
do
  echo "Count is $COUNT"
  COUNT=$((COUNT + 1))
done
```

- **Functions**: Reusable blocks of code.

```
my_function() {
  echo "This is a function"
}
my_function
```

Disclaimer

The series of handouts distributed with this course are only for educational purposes. Any actions and or activities related to the material contained within this handout is solely your responsibility. The misuse of the information in this handout can result in criminal charges brought against the persons in question. The authors will not be held responsible in the event any criminal charges be brought against any individuals misusing the information in this handout to break the law.