Lab02
Chapter 1 and Linux OS structure and commands

Section 1: Summary – Self-study before entering the lab

What Operating Systems Do?

An operating system (OS) is a software that manages a computer's hardware and provides the basis for application programs. The OS is viewed from two perspectives.

- Resource Allocator: The OS manages and allocates system resources (CPU time, memory, file storage, I/O devices) to various programs and users.
- Control Program: The OS controls program execution and manages I/O devices to prevent errors and improper use of the computer.

Computer-System Organization and Architecture

A modern computer system consists of CPU and device controllers connected by a system bus with shared main memory.

- Interrupts are the main mechanism for hardware to communicate with the CPU. An interrupt stops the CPU's current execution and transfers control to a specific interrupt handler routine.
- Storage is arranged in a hierarchy from fastest (but smallest and most volatile) to slowest (but largest and non-volatile): registers, cache, main memory (RAM), Non-volatile Memory (NVM/SSD), Hard Disk Drives (HDDs), and tertiary storage (e.g., tape).
- Direct Memory Access (DMA) is used by device controllers to transfer data blocks directly to/from memory without continuous CPU intervention, reducing overhead for bulk data movement.
- Modern systems are dominated by multiprocessor systems. Symmetric Multiprocessing (SMP) uses multiple peer processors that share physical memory. Multicore systems place multiple computing cores on a single chip. Non-Uniform Memory Access (NUMA) provides each CPU with local, faster memory, connecting CPUs via a system interconnect. Clustered systems join multiple independent nodes for high availability or high-performance computing.

Operating-System Operations

- Bootstrap Program: The system starts with a bootstrap program (stored in firmware) that initializes the system and loads the OS kernel. Multiprogramming and Multitasking: Multiprogramming keeps multiple processes in memory to maximize CPU utilization. Multitasking is an extension that switches the CPU among processes frequently, providing a fast response time to users.
- Dual-Mode Operation: Hardware uses a mode bit to distinguish between user mode and kernel mode. Privileged instructions (like I/O control or timer management) can be executed only in kernel mode, protecting the OS from user programs.
- System Calls: A system call is the mechanism by which a user program can request a protected OS service (transferring execution from user mode to kernel mode).

Resource Management and Protection

- Process Management: The OS manages processes (the unit of work) by creating, deleting, scheduling, and synchronizing them.
- Memory and Storage Management: The OS tracks memory use and allocates/deallocates space for processes. It also manages storage on secondary devices using file systems.
- Protection and Security: Protection controls authorized access to resources, while security defends the system from attacks.
- Virtualization: A technology that creates the illusion of separate, private computers on a single hardware system.

Data Structures and Environments

- Kernel Data Structures: The Linux kernel uses fundamental data structures such as lists, stacks, queues, trees, hash functions, maps, and bitmaps.
- Computing Environments: OSs are found in various environments, including traditional PCs, mobile devices (iOS, Android), client-server systems, peer-to-peer (P2P) systems, cloud computing (IaaS, PaaS, SaaS), and real-time embedded systems.
- Open-Source Systems: The chapter highlights free and open-source operating systems such as GNU/Linux, BSD UNIX, and Solaris.

Section 2: Discussion – selected questions only (30 Minuts)

Answer all of the following questions before entering the lab based on chapter 1 of the textbook. Selected questions will be discussed in the first half an hour of the lab.

1. **What is the purpose of booting in a computer? When does it starts?**
2. What are the three main purposes of an operating system?
3. **What is the purpose of interrupts? What are the differences between a trap and an interrupt? Can traps be generated intentionally by a user program? If so, for what purpose?**
4. **How does the distinction between kernel mode and user mode function as a rudimentary form of protection (security)?**
5. Which of the following instructions should be privileged?

   - Set value of timer.
   - Read the clock.
   - Clear memory.
   - Issue a trap instruction.
   - Turn off interrupts.
   - Modify entries in device-status table.

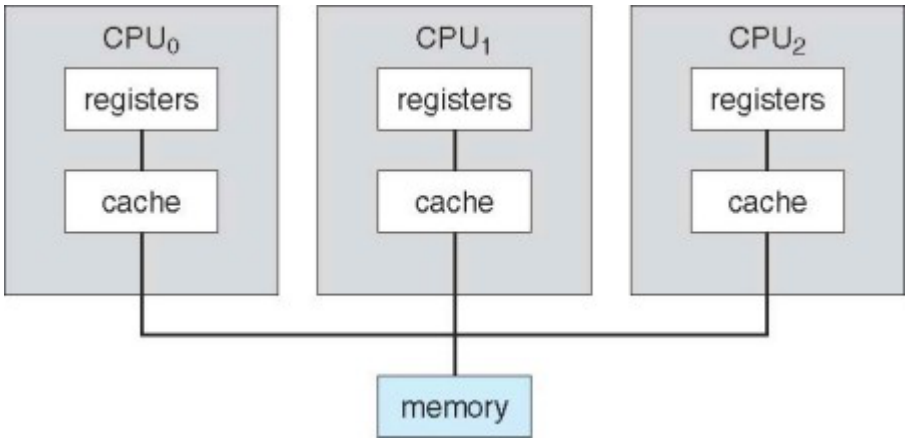
- Switch from user to kernel mode.
- Access I/O device.

6. Describe the differences between symmetric and asymmetric multiprocessing. What are three advantages and one disadvantage of multiprocessor systems?

7. **Direct memory access is used for high-speed I/O devices in order to avoid increasing the CPU's execution load.**
   - **How does the CPU interface with the device to coordinate the transfer?**
   - **How does the CPU know when the memory operations are complete?**

8. Give two reasons why caches are useful. What problems do they solve? What problems do they cause? If a cache can be made as large as the device for which it is caching (for instance, a cache as large as a disk), why not make it that large and eliminate the device

9. **Consider an SMP system similar to what is shown below. Illustrate with an example how data residing in memory could in fact have two different values in each of the local caches.**



10. Rank the following storage systems from slowest to fastest:
    - Hard-disk drives
    - Registers
    - Optical disk
    - Main memory
    - Non-volatile memory
    - Magnetic tapes
    - Cache

Section 3: Linux Structure and basic commands (45 Minutes)

This exercise is designed for a 45 session to introduce the fundamental structure and navigation of the Linux File System Hierarchy.

Part 1: The Linux File Tree and Navigation

Concept: The Root and Paths

In Linux, the entire system - all files, programs, devices, and configurations - are organized under a single inverted tree structure that starts at the **Root Directory**, represented by a single forward slash: `/`.

All paths are:

- Absolute:** Start from the root (`/`) (e.g., `/home/user/Documents`).
- Relative:** Start from your current location (e.g., `Documents/lab_file.txt`).

Command	Purpose	Example	Output/Action
pwd	**P**rint **W**orking **D**irectory (Show current location).	pwd	/home/student
ls	**L**ist directory contents.	ls	Lists files and folders.
ls -l	List in **l**ong format (details like permissions, size, owner).	ls -l	-rw-r--r-- 1 user group 0 Sep 15 10:00 file1.txt
cd	**C**hange **D**irectory.	cd Documents	Moves into the Documents folder.

Hands-On Steps

1. Start and Locate:
   i) Open your terminal.
   ii) Confirm your starting location (your Home Directory) using the command: **pwd**
   iii) View the contents of your home directory: **ls**

2. Navigate to the Root (`/`):
   i) Use the absolute path to move to the very top of the file system: cd /
   ii) List the contents of the entire root directory (these are the main system folders): ls

3. Explore Special Path Shortcuts:
   i) Move back to your home directory using the special shortcut `~`:
   cd ~
   pwd

ii) Create a temporary folder and move into it:
mkdir temp_folder
cd temp_folder
pwd

iii) Move back up one level to the **parent directory** using the `..` shortcut:
cd ..
pwd

Part 2: Understanding Key Directories

Every directory under the root (`/`) has a specific, standardized purpose defined by the Filesystem Hierarchy Standard (FHS).

Directory	Name/Etymology	Primary Purpose	Example Content
/	**Root**	The base of the entire file system tree.	(/etc, /home, /usr, etc.)
/home	Home	Contains the personal directories for all regular users.	/home/your_username/Documents
/etc	**Etc**etera	**Configuration** files for the entire system and its services.	passwd, fstab, network configs.
/bin	**Bin**ary	**Essential** executable programs (commands) used by *all* users.	ls, cp, mv, cat.
/sbin	**S**ystem Binary	Executable programs for **System Administration** (usually only for the root user).	reboot, fdisk, route.
/usr	**U**nix **S**ystem **R**esources	Secondary hierarchy. Contains most user-land programs, libraries, and documentation.	/usr/bin/firefox, /usr/lib/.
/var	**Var**iable	Stores data that is **variable** and expected to change frequently (e.g., logs,	/var/log/ (log files), /var/cache/.

Directory	Name/Etymology	Primary Purpose	Example Content
		caches, spool files).	
/tmp	**Temp**orary	Holds temporary files; contents are usually deleted on reboot.	Application temp files.

Hands-On Steps

1. Examine System Configuration:
   - **i)** Navigate to the configuration directory: **cd /etc**
   - ii) List the contents of `/etc` to see the many configuration files: **ls**
   - iii) What type of files are in here?

2. Examine Essential Binaries:
   - i) Navigate to the directory holding basic commands: cd /bin
   - ii) List the contents and notice the familiar command names (`ls`, `mv`, `cp`): **ls | head -n 10**
   - iii) Could you run a command like `ls` if the `/bin` folder was missing?

3. Review Variable Data Location:
   - **i)** Move to the directory where system logs are kept: **cd /var/log**
   - **ii)** List the contents. You should see many `.log` files: **ls**
   - iii) Why are these logs stored in `/var` and not `/etc`?

Part 3: Basic File Management

Use your existing `temp_folder` or create a new one to practice making and moving files.

Command	Purpose	Example
mkdir	**Make dir**ectory.	mkdir project_alpha
touch	Create an empty file or update timestamp.	touch notes.txt
cp	**Cop**y file or directory.	cp notes.txt backup.txt
mv	**Move** or **Rename** file/directory.	mv notes.txt README.md
rm	**Rem**ove file.	rm backup.txt
rm -r	**R**emove directory **r**ecursively (CAUTION!).	rm -r project_alpha

Hands-On Steps

1. Prepare the Workspace:

   - Ensure you are in your home directory: **cd ~**
   - Create a workspace for this part:
     mkdir file_lab
     cd file_lab

2. Create and Rename a File:

   - Create an empty file named `draft.txt`
     touch draft.txt
     ls

3. Rename `draft.txt` to `final_report.txt` using the `mv` command:
   mv draft.txt final_report.txt
   ls

4. Copy and Organize:

   - Create a subdirectory named `backups`:
     mkdir backups

   - Copy the report into the `backups` folder:
     cp final_report.txt backups/
     ls backups

5. Clean Up:

   - Move back up one level:
     cd ..

   - CAUTION: Remove the entire `file_lab` directory and its contents recursively:
     rm -r file_lab
     ls

Section 4: Explore the linux OS (45 Minuts)

PART 1: Linux Processes

Process management is a core function of the Linux operating system, which is based on the concept of a process—an instance of a program being executed. Linux manages the entire lifecycle of a process, from its creation and scheduling to its execution and termination.

In Linux, a process can be:

- A Program in Execution: This is the primary definition.
- A Parent/Child Relationship: Processes are created via the fork() system call, resulting in a parent process and a child process. All processes, except the initial init or systemd process (PID 1), have a parent.
- A Thread: Linux implements threads (lightweight processes) as standard processes that share resources, simplifying kernel scheduling.

Concept	Explanation	Related Command
PID (Process ID)	A unique, non-reusable integer assigned to every active process.	ps, top
PPID (Parent Process ID)	The PID of the process that created the current process.	ps -o pid,ppid,cmd
States	A process moves through states: **Running**, **Waiting** (for resources/I/O), **Stopped** (paused), and **Zombie** (terminated, but waiting for parent to collect its status).	ps aux
Job Control	The ability to manage foreground and background processes in a shell.	&, Ctrl+Z, bg, fg

1. Viewing Active Processes (`ps`, `top`)

These commands show what's running on the system.

Action	Command	Expected Output/Explanation
View your own processes	ps	Lists processes owned by you in the current terminal session.
View all user processes	ps aux	Lists all running processes system-wide (A), including processes of other users (u), and

Action	Command	Expected Output/Explanation
		processes not attached to a terminal (x). **PID** and **PPID** columns are visible here.
View processes interactively	top	Provides a real-time, dynamic view of running processes, ranked by CPU usage. Press `q` to exit.
View a process tree	pstree	Displays processes in a hierarchical, indented list, showing parent-child relationships clearly.

2. Launching and Controlling Processes (Job Control)

Job control allows you to run long-running tasks without blocking your terminal.

A. Launch a long-running process in the foreground: sleep 300
B. Move the process to the background:

- While the `sleep 300` command is running, press **`Ctrl+Z`** to **stop** the process.
- Then, enter the `bg` command to continue the process in the **background**.

 # (press ctrl+z)
^z
[1]+ stopped sleep 300
bg
[1]+ sleep 300 &
 # the process is now running in the background, and your terminal is free.

C. **Launch a process directly into the background:**
- Use the ampersand (`&`) operator at the end of the command.

sleep 60 &
 # Output will show the Job ID ([2]) and the PID (e.g., 54321).

D. View background jobs:

jobs
 # Lists all jobs currently running or stopped in the background.

E. Move a background process to the foreground: **fg %1**

3. Terminating process (kill)

The `kill` command is used to send signals to a process, typically to end it.

Action	Command	Explanation
Find the PID	pgrep sleep	Find the Process ID (PID) of the sleep commands you ran in the previous step. (Let's assume the PID is 54321).
Graceful Termination (SIGTERM)	kill 54321	This is the default signal (Signal 15: SIGTERM) that requests the process to shut down cleanly, allowing it to save its state.
Immediate Termination (SIGKILL)	kill -9 54321	This sends Signal 9 (SIGKILL), which forces the process to terminate immediately without giving it a chance to clean up. Use this only if the graceful kill fails.

Note on Zombies: A **Zombie** process appears when a child process terminates but its parent has not yet collected its exit status using the `wait()` system call. It is a dead process that still occupies a slot in the process table. Zombies cannot be killed using `kill`, as they are already dead. They must be reaped by their parent or adopted by the `init`/`systemd` process.

The Process Hierarchy

Every process traces its lineage back to **PID 1**, the first process launched by the Linux kernel.

- **PID 1 (init/systemd):** The mother of all processes. It starts all other services and processes, and it adopts any orphaned processes (parents that terminated before their children) to prevent them from becoming permanent zombies.
- **Daemons:** Long-running processes (often ending with `d` like `sshd` or `httpd`) that run in the background to provide services. They are typically created early in the boot process.
- **Shells:** When you open a terminal, your shell (e.g., `bash`) is a child of the terminal emulator, and any command you run is a child of the shell.

This structure ensures that system resources are always managed and that no process, once started, is truly left abandoned.

PART 2: Linux Device Management

Linux abstracts nearly all hardware and system components as **files**, which provides a simple and consistent interface for processes to interact with devices. This approach is rooted in the "everything is a file" philosophy of Unix.

The `/dev` File Structure

The primary directory for Linux devices is **`/dev`** (short for **dev**ices). Files in this directory are not regular files (they don't store data in the conventional sense); instead, they are **device nodes** or **special files** that act as interfaces to hardware or pseudo-devices managed by the kernel.

The devices are categorized into two main types:

Device Type	Description	File Designation	Example Commands/Files
1. Block Devices	Devices that transfer data in fixed-size blocks (e.g., 512 bytes, 4096 bytes). They are primarily used for random-access storage.	Starts with `b` (e.g., `brw-r--r--`).	`/dev/sda` (entire hard drive), `/dev/sdb1` (partition).
2. Character Devices	Devices that transfer data one character (byte) at a time (sequential access). They are used for streams of data.	Starts with `c` (e.g., `crw-rw-rw-`).	`/dev/tty` (terminals), `/dev/null` (the "black hole"), `/dev/random` (random numbers).

Device Node	Purpose	Type
/dev/null	The **"black hole."** Any data redirected to this device is discarded. Reading from it immediately returns an end-of-file.	Character
/dev/zero	A source of infinite **null characters** (zeros). Reading from it provides a stream of zero bytes.	Character
/dev/random	Provides high-quality **cryptographic random numbers** by gathering environmental noise (entropy). May block if entropy pool is empty.	Character
/dev/sda	Represents the first SCSI, SATA, or USB disk (A for first, B for second, etc.). Partitions are numbered (e.g., `/dev/sda1`).	Block

Device Node	Purpose	Type
/dev/tty*	Represents terminals or console devices.	Character

Device Management Commands

Interacting with devices usually involves administrative tasks like checking disk space, partitioning, or mounting filesystems.

1. Viewing Devices and Disk Usage

Command	Purpose	Example Output/Explanation
ls -l /dev	Lists the contents of the /dev directory to show all device nodes.	Look for the first letter (b or c) to determine the device type.
lsblk	**Lis**ts **Bl**oc**k** devices in a tree-like format, showing disks and their partitions.	Shows hard drives (sda), partitions (sda1), and mount points.
df -h	**D**isk **F**ree. Reports on filesystem disk space usage, showing **mount points** and sizes in **h**uman-readable format.	Shows how much space is used/available on mounted partitions.
fdisk -l	Lists disk partition tables (requires elevated privileges like sudo).	Detailed, low-level view of disks and partition sizes.

2. Mounting and Interacting with Devices

Mounting makes a device's filesystem accessible under a specific directory in the single Linux file tree.

Command	Purpose
mount	Attach (mount) a device/filesystem to a mount point (a directory).
umount	Detach (unmount) a mounted device. **Crucial** before physically removing external media.
cat /dev/zero	(Demo only) Continuously output zero bytes (you must press Ctrl+C to stop it).

Command	Purpose
echo "test" > /dev/null	Demonstrate discarding output by redirecting text to the null device.

Example: Checking a New Drive

If you plug in a new USB drive, you would typically use these commands:

Check the device node: **lsblk**

Mount the filesystem (if not automatic): **sudo mount /dev/sdc1 /mnt**

View contents: **ls /mnt**

Unmount when done: **sudo umount /mnt**

PART3: Memory Management

Linux memory management is the operating system's process of allocating, tracking, and protecting the computer's **main memory (RAM)**. It creates a seamless experience for users and programs by handling two primary tasks: managing the limited physical RAM and providing a vast, protected **virtual memory** space to every process.

Key Memory Management Concepts

Concept	Explanation
Virtual Memory	A technique that separates the memory addresses used by a process (logical addresses) from the actual physical addresses. This allows processes to use more memory than is physically available and isolates them from each other.
Paging	The main mechanism for implementing virtual memory. Memory is divided into fixed-size blocks: **pages** (logical) and **frames** (physical). The OS moves pages between physical RAM and **Swap Space** on the disk.
Swap Space	A dedicated area on a disk (partition or file) used as an extension of RAM. When physical memory is full, the OS moves less-used pages from RAM to swap (swapping out) to free up frames for active processes.
Kernel	Memory is split into two regions: **Kernel Space** (for the OS kernel and

Concept	Explanation
Space vs. User Space	its data, protected and privileged) and **User Space** (for application programs, restricted access).
Buffers and Cache	Areas of memory used to temporarily store data for I/O operations. **Buffer** stores data to/from block devices (like disks), while **Cache** stores recently accessed files to speed up subsequent requests.

Memory Management Commands

Linux provides several commands to inspect the system's current memory status, usage by processes, and swap configuration.

1. Overall System Memory (`free`)

The `free` command is the standard way to check the total, used, and free amounts of physical and swap memory.

Action	Command	Key Columns	Explanation
Display memory statistics	free -h	**Total**, **Used**, **Free**, **Shared**, **Buff/Cache**, **Available**	The -h flag displays values in **h**uman-readable format (KB, MB, GB). **Available** is the most accurate measure of memory instantly usable by new applications.
Display in Megabytes	free -m	*Same as above*	Displays all values in Megabytes (MB).

2. Process-Specific Memory Usage (`top` / `ps`)

These tools show how much memory individual processes are consuming.

Action	Command	Key Metrics	Explanation
Interactive monitor	top	**VIRT**, **RES**, **SHR**	Press M (Shift+M) while in top to sort processes by memory usage.
Static report	ps aux	**%MEM**	The %MEM column shows the percentage of physical RAM used by the process.

Action	Command	Key Metrics	Explanation
Detailed process memory	cat /proc/<PID>/status	**VmSize**, **VmRSS**	Replace <PID> with a process ID (found using ps or top). **VmSize** is total Virtual Memory size, and **VmRSS** is the **R**esident **S**et **S**ize (actual physical RAM used).

3. Swap Space Management

These commands are used to check and configure swap space.

Action	Command	Explanation
Check configured swap	swapon --show	Lists all active swap devices (partitions or files) and their sizes.
Check disk usage (for swap partition)	df -h	If a swap partition is mounted, it will show up here, although swapon is more specific.
Displaying system-wide virtual memory stats	vmstat	Provides detailed reports on memory usage, swap activity, I/O, and CPU activity. Key columns include si (swap in) and so (swap out).

Example: Observing Buffer/Cache

When using the `free -h` command, you often see that the `Used` memory is very high, but the `Available` memory is also high. This is primarily due to the **Buffer/Cache**:

Example Output of 'free -h'

	total	used	free	shared	buff/cache	available
Mem:	7.8G	4.2G	500M	100M	3.1G	3.0G
Swap:	2.0G	0B	2.0G			

In this example:

- The system is using **4.2 GB** for running applications (`Used`).
- It is using **3.1 GB** for disk caching/buffers (`Buff/Cache`).
- The cache memory can be instantly reclaimed by applications, so **3.0 GB** is immediately **Available**.
- **Linux uses this cache aggressively** to improve performance, treating "free" memory as wasted resources. The `Used` column should *not* be interpreted as critically low memory.