Linux File System

1. Linux uses file systems like ext2, ext3, ext4, XFS, Btrfs.
2. **Ext3**-ext3 is standard on disk file system for Linux

- Uses a mechanism similar to that of BSD Fast File System (FFS) for locating data blocks belonging to a specific file
- Supersedes older extfs, ext2 file systems
- Work underway on ext4 adding features like extents

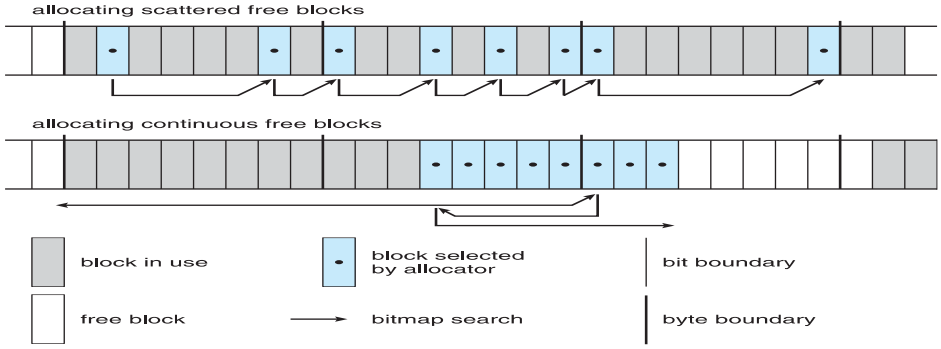
1. Ext2-

- ext2 is a **non-journaling file system** used in early Linux systems.
- It offers **fast performance** due to the absence of journaling overhead.
- Supports **large files and volumes** up to 32TB.
- Still used in **embedded systems** (e.g., USB drives, boot partitions).

2. Ext4-

- ext4 is the **default file system** in most modern Linux distros.
- Supports **extents**, reducing fragmentation and improving speed.
- Offers **journaling**, large volume support, and delayed allocation.
- Backward-compatible with ext3 and ext2.
- Example: Default in **Ubuntu, Debian, and Fedora**.

3. File names are **case-sensitive** (e.g., `data.txt` ≠ `Data.txt`).
4. The structure is **hierarchical**, starting from the root `/`.
5. All devices are represented as files in `/dev` (e.g., `/dev/sda`).
6. File permissions follow the **owner/group/others** model using `rwx` bits.
7. Linux uses **inodes** to store file metadata (e.g., size, owner).
8. **Symbolic and hard links** are supported (`ln -s source link`).
9. External drives are **mounted into the directory tree** (e.g., `/mnt/usb`).
10. Hidden files start with a **dot**, like `.bashrc`.
11. File naming allows most characters except `/` and null.
12. **Journaling** is used in ext3/ext4 for crash recovery.
13. **All files are treated equally**, including programs, text, and devices.
14. Each user has a home directory (e.g., `/home/roshan/`).
15. File system is managed using commands like `ls`, `chmod`, `mount`.
16. Example: A valid path is `/home/user/docs/report.txt`.



Components of linux system

1. Linux Kernel

- The kernel is the **core part** of Linux, managing CPU, memory, and devices.
- It provides **process scheduling**, **virtual memory**, and **file system** control.
- Implements system calls like `fork()`, `exec()`, `read()`, etc.
- Acts as a **bridge between user programs and hardware**.
- It runs in **privileged (kernel) mode** for direct hardware access.
- Example: `/boot/vmlinuz-<version>` is a Linux kernel binary.

2. Loadable Kernel Modules (LKMs)

- LKMs are pieces of code that **extend kernel functionality at runtime**.
- They allow **drivers or filesystems** to be added without rebooting.
- Used for hardware like **USB, network, or GPU support**.
- Commands like `insmod`, `modprobe`, `rmmod` manage modules.
- Modules help keep the kernel **modular and compact**.
- Example: `e1000.ko` is a module for Intel Ethernet driver.

3 System Shared Libraries

- Shared libraries contain **common functions** used by many programs.
- Reduces redundancy and **memory usage** across multiple applications.
- They are loaded dynamically using **dynamic linking**.
- Stored typically in `/lib` or `/usr/lib` directories
- Shared libraries have `.so` extensions (e.g., `libc.so.6`).
- Example: `libm.so` provides math functions like `sqrt()` or `sin()`.

4 Compilers

- Compilers convert **source code into executable binaries**.
- Common Linux compilers include **GCC**, **Clang**, etc.
- They support languages like **C, C++, Fortran**, etc.
- Includes preprocessors, assemblers, and linkers.
- Compilers generate **optimized binaries** for performance.
- Example: `gcc main.c -o main` compiles a C program.

5. User Utility Programs

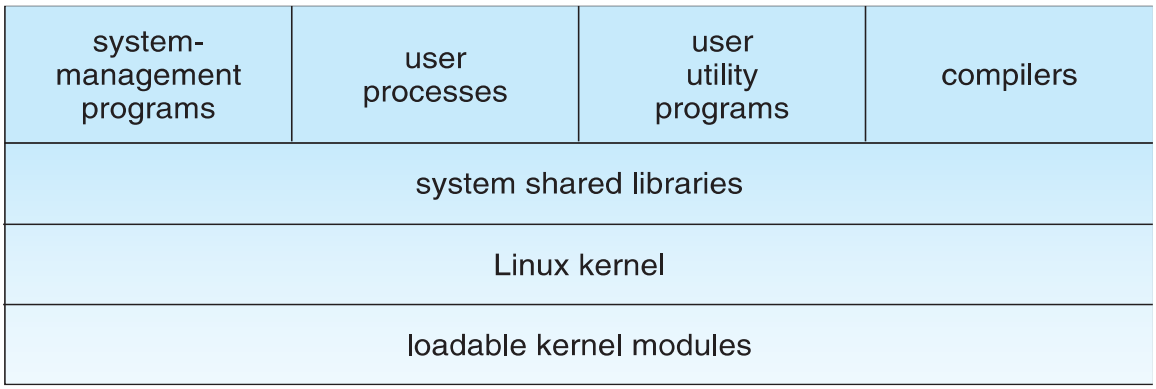
- Utility programs perform **routine user-level tasks**.
- Includes file tools (`cp`, `mv`, `rm`) and text tools (`grep`, `awk`).
- Enhances user productivity via command-line or GUI tools.
- Typically stored in `/bin`, `/usr/bin`, `/sbin`.
- These tools often wrap around **system calls**.
- Example: `tar` is used for archiving and compressing files.

6. System Management Programs

- These are used by **admins to manage the system environment**.
- Includes services, daemons, and configuration tools.
- Manages **users, services, storage, networking**, etc.
- Uses files like `/etc/passwd`, `/etc/fstab`, `/etc/systemd`.
- Often runs at **startup or as background services (daemons)**.
- Example: `systemctl` is used to manage services in `systemd`.

7. User Processes

- These are the **running instances** of user programs.
- Each process has a **unique PID** and runs in user mode.
- Can be created using system calls like `fork()` or `exec()`.
- Interacts with the kernel via system calls for I/O or memory.
- Tools like `top`, `ps`, and `kill` manage user processes.
- Example: A browser or terminal running is a user process (`firefox`, `gnome-terminal`)



Windows File System(any 10) and it components

- ✧ Windows uses **NTFS**, **FAT32**, and **exFAT** file systems.
- ✧ File names are **case-insensitive** (`data.txt` = `Data.txt`).
- ✧ File system structure is **drive-letter based** (e.g., `C:\`, `D:\`).
- ✧ Devices are **not shown as files**, managed through device drivers.
- ✧ File permissions are managed using **Access Control Lists (ACLs)**.
- ✧ NTFS stores metadata in a **Master File Table (MFT)**.
- ✧ Supports **shortcuts**, and symbolic links only via NTFS with admin rights
- ✧ Drives appear as **separate logical volumes**, not mounted to root.
- ✧ Files are hidden using **attributes**, not dot prefixes.
- ✧ Disallows characters like `\ / : * ? " < > |` in filenames.
- ✧ NTFS supports **journaling and self-healing features**.
- ✧ Uses **User Account Control (UAC)** for file access security.
- ✧ User files are stored under `C:\Users\Username\`.
- ✧ File system managed via Explorer and commands like `dir`, `attrib`.
- ✧ Example: A valid path is `C:\Users\Roshan\Documents\report.txt`

Networking -Windows 7 supports both peer-to-peer and client/server networking; it also has facilities for network management.

- ✧ To describe networking in Windows 7, we refer to two of the internal networking interfaces:
- ✧ NDIS (Network Device Interface Specification) — Separates network adapters from the transport protocols so that either can be changed without affecting the other.
- ✧ TDI (Transport Driver Interface) — Enables any session layer component to use any available transport mechanism.

File system

- ✧ All metadata, such as information about the volume, is stored in a regular file
- ✧ NTFS uses clusters as the underlying unit of disk allocation
- ✧ A cluster is a number of disk sectors that is a power of two

File system - internal layout

- ✧ NTFS uses logical cluster numbers (LCNs) as disk addresses
- ✧ A file in NTFS is not a simple byte stream, as in MS-DOS or UNIX
- ✧ Each file on an NTFS volume has a unique ID called a file reference.
- ✧ 64-bit quantity that consists of a 48-bit file number and a 16-bit sequence number

File system-recovery

- ✧ All file system data structure updates are performed inside transactions that are logged.
- ✧ Before a data structure is altered, the transaction writes a log record that contains redo and undo information.

File system - security

- ✧ Security of an NTFS volume is derived from the Windows 7 object model.
- ✧ Each file object has a security descriptor attribute stored in this MFT record.
- ✧ File system-compression
  To compress a file, NTFS divides the file's data into compression units, which are blocks of 16 contiguous clusters.

Windows components.

1. Windows Kernel (NT Kernel)

- ✧ Core component that handles **process, memory, and hardware management**.
- ✧ Supports **multitasking, thread scheduling, and interrupt handling**.
- ✧ Provides interface to Windows Executive via **system calls**.
- ✧ Operates in **kernel mode**, isolated from user applications.
- ✧ Manages **virtual memory** and **I/O operations**.
- ✧ Example: The `ntoskrnl.exe` file is the main kernel image.

2. Windows Executive

- ✧ Layer above the kernel offering high-level OS services.
- ✧ Includes managers like file systems cache management ,device drivers,network drivers
- ✧ Interfaces between **subsystems** and the kernel.
- ✧ Handles **resource allocation and system-wide coordination**.
- ✧ Critical for system services such as **security, memory, processes**.
- ✧ Example: Manages file operations when user accesses documents.

3. System Processes and Services

- ✧ Provide essential OS services in **user or kernel mode**.
- ✧ Examples include `lsass.exe` (security), `csrss.exe` (console), `winlogon.exe`.
- ✧ Automatically started during boot and run in background.
- ✧ Can be managed using `services.msc`.
- ✧ Crucial for **authentication, printing, event logging, etc.**
- ✧ Example: `svchost.exe` hosts multiple system services.

4. User Interface (UI Subsystem)

- ✧ Provides **Graphical User Interface (GUI)** for user interaction.
- ✧ Includes **Windows Shell**, Desktop, Taskbar, File Explorer.
- ✧ Handles user inputs via **mouse, keyboard, touch**.
- ✧ Subsystems like **Win32 API** offer GUI programming support.
- ✧ Supports **multiple desktop environments and themes**.

5. File System (NTFS)

- ✧ NTFS is the **default Windows file system**, supporting security and compression.
- ✧ Supports **file permissions, journaling, encryption**.
- ✧ Allows large volume and file support (up to 16 EB).
- ✧ Integrates with **Windows ACLs (Access Control Lists)**.
- ✧ Supports **file metadata, indexes, and recovery**.
- ✧ Example: Files stored in `C:\Users\Roshan\Documents` are managed via NTFS.

🛠

6. Device Drivers

- ✧ Drivers let OS communicate with **hardware devices**.
- ✧ Managed by the I/O Manager in the Executive.
- ✧ Windows supports **plug and play (PnP)** device drivers.
- ✧ Drivers run in **kernel mode** for direct hardware access.
- ✧ Can be updated via **Device Manager**.
- ✧ Example: `nvlddmkm.sys` is the NVIDIA graphics driver

7. HAL (Hardware Abstraction Layer)

- ✧ Abstracts hardware differences for compatibility across devices.
- ✧ Allows Windows to run on diverse platforms like x86, ARM
- ✧ Provides a consistent interface to access **hardware resources**.
- ✧ Lies between the **kernel and physical hardware**.
- ✧ Ensures portability of OS across machines.
- ✧ Example: Converts generic instructions into hardware-specific I/O.

File I/O