Operating Systems Notes - Chapter 1

June 9, 2025

Contents

1.1 Introduction to Operating Systems

1 Purpose and Core Responsibilities

- **Intermediary:** OS sits between user/applications and hardware.
  - Hides hardware details.
  - Exposes services.
- **Goals:**
  - Run programs conveniently (consistent UI).
  - Run programs efficiently (performance, safe sharing).
  - Manage hardware resources (CPU, memory, I/O, storage).

2 Abstraction Layers (high → low)

1. **User** – GUI, CLI, touch, voice.
2. **Application programs** – browsers, compilers, games.
3. **Operating system** – kernel, system programs, middleware.
4. **Computer hardware** – CPU, memory, devices.

3 User and Kernel Mode Transition

- Hardware mode bit switches CPU from *user* to *kernel* on system call (trap) or interrupt.
- Return-from-trap restores user mode.

4 User View vs. System View

A von Neumann CPU follows a fetch–decode–execute loop, issuing memory addresses the OS must service quickly and safely.

- **User view:** Focus on ease of use; resource details hidden.
- **System view:**
  - **Resource allocator** – arbitrates CPU, memory, I/O.
  - **Control program** – governs execution, manages devices.

5 What Forms an Operating System?

Always	Usually	Optional / Varies
Kernel – always runs after boot.	**Device drivers**, loadable modules.	**Middleware** (graphics, DB), extra utilities.
	System programs (shells, daemons).	

- **Firmware OSs:** microcode, BIOS/UEFI, and many device controllers contain minimal OS.
- Run continuously beneath the main kernel.

6 Why Study Operating Systems?

- All code runs on an OS.
- Knowing OS policies & APIs yields safer, faster, portable software.
- **OS flavours:**
  - Desktop (Windows, macOS).
  - Server (Linux, Windows Server).
  - Mobile (Android, iOS).
  - Embedded/real-time systems (emphasize different goals).
- **Open-source examples:** Linux, FreeBSD, Minix show how textbook concepts map to production kernels.
- OS concepts (processes, memory, I/O, security) recur in servers, clouds, IoT.

7 Glossary Highlights (1.1)

Term	Definition
Operating system	Intermediary between user/applications and hardware; hides details, exposes services; runs programs conveniently/efficiently; manages hardware resources.
Kernel	Always runs after boot; provides services.
System program	Loaded at boot, becomes system daemon; provides services outside kernel.
Middleware	Optional/varies; graphics, DB, extra utilities.
Resource allocator	Part of OS (system view); arbitrates CPU, memory, I/O.
Control program	Part of OS (system view); governs execution, manages devices.
Ease of use	User view goal; focus on convenience, resource details hidden.
Resource utilization	System view goal; focus on efficiency, performance, safe sharing.

1.2 Computer-System Organization

1 System-Level Hardware Layout

- Shared **system bus** connects:
  - CPU cores.
  - Main memory.
  - Device controllers.
- **Device controller:** Logic per device class (disk, GPU, USB) with registers & buffer.
- **Device driver:** Kernel API shielding hardware details.
- **Parallelism:** CPU & controllers run concurrently.
- **Memory controller:** Arbitrates memory access.

2 Interrupts – Hardware-Driven Events

2.1 Lifecycle

1. Controller raises signal on **interrupt-request line**.
2. CPU catches signal, saves state, jumps to handler via **interrupt vector**.
3. Handler services device, restores state, executes `return_from_interrupt`.

2.2 Efficiency Features

- **O(1) dispatch** via vector table.
- **Nonmaskable** vs. **maskable** lines (maskable can be disabled).
- **Priority & interrupt chaining:** High-priority pre-empts low; vector entry may head list of handlers.

2.3 End-to-End I/O Timeline

```
User code → I/O request → controller/DMA busy → finish → interrupt → handler → resume user code
```

3 Storage Structure

3.1 Memory Hierarchy

Layer	Volatile?	Size	Access	Notes
Registers	Yes	bytes	sub-ns	in CPU
Cache	Yes	KB–MB	ns	SRAM
Main memory	Yes	GBs	∼10 ns	DRAM
NVM/SSD	No	10 GB–TB	μs	electrical
HDD	No	100 GB–TB	ms	mechanical
Optical/tape	No	TB–PB	s–min	archival

3.2 Key Units & Terms

- **Bit** → **byte (8 bits)** → **word** (CPU width).
- **KiB / MiB / GiB / TiB / PiB** = powers of 1024.
- **Volatile** memory loses data on power-off.
- **Non-volatile storage (NVS)** persists data.
- NVS divides into:
  - *Mechanical* media (HDD, optical, tape).
  - *Electrical* non-volatile memory (NVM: SSD, flash).
- **Firmware / EEPROM** stores bootstrap program.
- **Interrupt masking:** Kernel may briefly disable maskable interrupts for data coherence in critical sections.

3.3 Why Secondary Storage?

- Main memory is finite and volatile.
- Programs & data live on slower persistent media until loaded.

4 I/O Structure

4.1 Direct Memory Access (DMA)

- Driver configures DMA engine.
- Controller moves block without CPU.
- Raises one interrupt on completion.

4.2 Bus vs. Switched Fabrics

- **Shared bus:** One transfer at a time; common on PCs.
- **Switch fabric:** Concurrent links; used in servers & SoCs.

4.3 Full I/O Cycle

1. Driver starts I/O.
2. Controller activates device/DMA.
3. CPU handles other tasks, checks interrupts.
4. Device finishes, raises interrupt.
5. Handler validates data, wakes waiting process.

5 Glossary Highlights (1.2)

Term	Definition
Bus	Shared path linking CPU, memory, controllers.
Device driver	Kernel interface to a controller.
Interrupt / vector / request line	Hardware signal and its dispatch mechanism.
Maskable / nonmaskable interrupt	Can or cannot be disabled.
DMA	Controller-driven block transfer to/from memory.
Volatile / non-volatile memory	Data lost or retained on power loss.

1.4 Operating-system operations

- OS provides environment for program execution.
- Internal OS variations exist, but commonalities are shared.

1 Initial Program and Kernel Loading

- Computer needs initial program on power-up/reboot.
- **Bootstrap program:** Simple, stored in **firmware**.
- Initializes system: CPU registers, device controllers, memory.
- Locates and loads **operating-system kernel** into memory.
- Kernel provides services once loaded and executing.
- Some services by **system programs** loaded at boot, becoming **system daemons**.
- Example: On Linux, `systemd` starts other daemons.
- After booting, system waits for events (e.g., interrupts or traps).

Hadoop

- Open-source software framework for distributed **big data** processing in clustered systems.
- Scales from single system to thousands of nodes.
- Assigns tasks, manages parallel communication, coalesces results.
- Detects/manages node failures; provides efficient, reliable distributed computing.
- Organized around three components:
  1. Distributed file system (manages data/files across nodes).
  2. **YARN** ("Yet Another Resource Negotiator") framework (resource management, task scheduling).
  3. **MapReduce** system (parallel data processing across nodes).
- Primarily runs on Linux.
- Applications in PHP, Perl, Python, Java (Java popular due to MapReduce libraries).

2 Interrupts and Traps

- Events almost always signaled by an **interrupt**.
- **Trap** (or **exception**): Software-generated interrupt.
  - Caused by error (e.g., division by zero, invalid memory access).
  - Caused by user program request for OS service via **system call**.

3 Multiprogramming and Multitasking

- OS ability to run multiple programs is important.
- Single program cannot keep CPU/I/O devices busy always.
- **Multiprogramming:**
  - Increases CPU utilization and user satisfaction.
  - OS keeps several **processes** (programs in execution) in memory simultaneously.
  - When a process waits (e.g., I/O), OS switches to another.
  - CPU never idle if a process needs execution.
  - *Figure 1.4.1: Memory layout for a multiprogramming system. Diagram shows vertical column representing memory which contains layers of operating system, process 1, process 2, process 3, and process 4.*
- **Multitasking:**
  - Logical extension of multiprogramming.
  - CPU switches frequently among processes for fast **response time**.
  - Handles slow interactive I/O (e.g., user typing).
  - Requires:
    - Memory management (*Main Memory*, *Virtual Memory*).
    - **CPU scheduling** (choosing next process to run).
    - Protection mechanisms (limit process interference).
    - **Virtual memory**: Allows execution of processes not fully in memory; enables programs larger than physical memory; abstracts main memory into large, uniform storage, separating **logical memory** from physical.
    - File system (*File-System Interface*, *File-System Implementation*, *File-System Internals*).

    * Storage management (*Mass-Storage Structure*).
    * Resource protection (*Protection*).
    * Process synchronization and communication (*Synchronization Tools*, *Synchronization Examples*).
    * Deadlock prevention (*Deadlocks*).

4 Dual-mode and Multimode Operation

- Ensures incorrect/malicious programs don't harm others or OS.
- Distinguishes OS code execution from user code.
- Hardware supports differentiation among execution modes.
- At least two separate modes:
  - **User mode**: System executing for user application.
  - **Kernel mode** (also **supervisor mode**, **system mode**, or **privileged mode**): System executing for OS.
- **Mode bit** added to hardware: kernel (0) or user (1).
- *Figure 1.4.2: Transition from user to kernel mode. Diagram shows events such as user process executing, calling system call and return from system call in user mode whereas trapping mode bit 0, execute system call, and returning mode bit 1 in kernel mode.*
- At boot, hardware starts in kernel mode.
- OS loaded, starts user applications in user mode.
- Trap or interrupt: hardware switches from user to kernel mode (mode bit to 0).
- OS always gains control in kernel mode.
- System switches to user mode (mode bit to 1) before passing control to user program.
- *Figure 1.4.3: Windows Performance Monitor tool. Image shows a graph which utilizes an x-axis with time and y-axis with percentages of system usage. It shows in green the user mode which is higher than the red line which is in kernel mode.*
- Dual mode protects OS from errant users, and users from each other.
- **Privileged instructions**: Harmful machine instructions, executable only in kernel mode.
  - Attempt in user mode → hardware treats as illegal, traps to OS.
  - Examples: switch to kernel mode, I/O control, timer management, interrupt management.
- Modes can extend beyond two:
  - Intel processors: four **protection rings** (ring 0 kernel, ring 3 user).
  - ARM v8 systems: seven modes.
  - CPUs supporting virtualization: separate mode for **Virtual Machine Manager (VMM)**.
- Instruction execution lifecycle: OS (kernel) → user app (user) → back to OS via interrupt, trap, or system call.
- Most contemporary OS (Windows, Unix, Linux) use dual-mode for protection.

System Calls

- User program asks OS to perform reserved tasks.
- Invoked via trap to specific interrupt vector location.
- Can use generic 'trap' or specific 'syscall' instruction.
- Hardware treats as software interrupt.
- Control passes to OS service routine; mode bit set to kernel.
- Kernel examines instruction, determines system call, verifies parameters, executes request, returns control.

Program Errors

- Hardware protection detects mode-violating errors (e.g., illegal instruction, invalid memory access).
- Hardware traps to OS.
- OS terminates program abnormally, provides error message, may dump memory for examination.

5 Timer

- Ensures OS maintains CPU control.
- Prevents user programs from infinite loops or failing to return control.
- Can interrupt computer after specified period (fixed or variable).
- Variable timer: fixed-rate clock and counter.
  - OS sets counter.
  - Every clock tick, counter decremented.

  - Counter reaches 0, interrupt occurs.
- OS ensures timer set before turning control to user.
- If timer interrupts, control transfers automatically to OS (may treat as fatal error or give more time).
- Instructions modifying timer content are privileged.

Linux timers

- Kernel config parameter **HZ** specifies timer interrupt frequency (e.g., 250 HZ = 250 interrupts/sec).
- Related kernel variable **jiffies** represents timer interrupts since boot.

6 Section glossary

Term	Definition
Big data	Extremely large data sets; distributed systems suited for it.
MapReduce	Google-created big data programming model for parallel processing across distributed cluster nodes.
System daemon	Service outside kernel by system programs loaded at boot, running continuously.
Trap	Software interrupt from error or user program request for OS service.
Exception	Software-generated interrupt from error or user program request for OS service.
System call	Software-triggered interrupt for process to request kernel service.
Multiprogramming	Increases CPU utilization by organizing jobs so CPU always has one to execute.
Process	Program loaded into memory and executing.
Multitasking	Concurrent performance of multiple jobs; CPU switches frequently for fast response.
Response time	Time for system to respond to user action.
CPU scheduling	System chooses which ready job runs next.
Virtual memory	Allows process execution not completely in memory; enables programs larger than physical memory; separates logical from physical memory.
Logical memory	Memory as viewed by user; large uniform array, not matching physical memory in virtual memory systems.
User mode	CPU mode for user processes; some instructions limited/disallowed.
Kernel mode	CPU mode with all instructions enabled; kernel runs here.
Supervisor mode	CPU mode with all instructions enabled; kernel runs here.
System mode	CPU mode with all instructions enabled; kernel runs here.

1.5 Resource management

- OS is a **resource manager**, managing:
  - CPU.
  - Memory space.
  - File-storage space.
  - I/O devices.

1 Process management

- A program in execution is a **process** (e.g., compiler, word processor, social media app).
- A process needs resources: CPU time, memory, files, I/O devices, allocated during execution.
- Initialization data (input) may be passed to a process (e.g., URL for web browser).
- OS reclaims reusable resources when a process terminates.
- Program is a *passive* entity (like a file on disk); process is an *active* entity.
- **Single-threaded process:** One **program counter** specifies next instruction; sequential execution.
- **Multithreaded process:** Multiple program counters (*Threads & Concurrency*).
- A process is the unit of work in a system.
- System consists of OS processes (execute system code) and user processes (execute user code).
- Processes can execute concurrently (multiplexing on single CPU core) or in parallel (across multiple CPU cores).
- OS responsibilities:
  - Create/delete user and system processes.
  - Schedule processes and threads on CPUs.
  - Suspend/resume processes.
  - Provide mechanisms for process synchronization.
  - Provide mechanisms for process communication.
- (Process management techniques in *Processes* through *Synchronization Examples*).

2 Memory management

- Main memory central to modern computer operation: large array of bytes, each with an address.
- Repository of quickly accessible data shared by CPU and I/O devices.
- CPU reads instructions/data from main memory (von Neumann architecture).
- Main memory generally only large storage CPU accesses directly; disk data transferred here.
- For execution, program mapped to absolute addresses and loaded into memory.
- To improve CPU utilization/system responsiveness, general-purpose computers keep several programs in memory.
- Memory-management schemes vary; effectiveness depends on situation and *hardware design.*
- OS responsibilities:
  - Track memory usage (which parts, by which process).
  - Allocate/deallocate memory space.
  - Decide which processes/data to move into/out of memory.
- (Memory-management techniques in *Main Memory* and *Virtual Memory*).

3 File-system management

- OS provides uniform, logical view of information storage.
- OS abstracts physical storage properties to define logical storage unit: the **file**.
- OS maps files onto physical media, accesses via storage devices.
- Computers store information on various physical media (secondary, tertiary).
- A **file** is a collection of related information defined by its creator (e.g., programs, data).
- Files can be free-form (text files) or rigidly formatted (fixed fields, e.g., mp3).
- OS implements file concept by managing mass storage media and control devices.
- Files normally organized into directories.
- Access control may be needed for multiple users (read, write, append).
- OS responsibilities:
  - Create/delete files.

  - Create/delete directories to organize files.
  - Support primitives for manipulating files/directories.
  - Map files onto mass storage.
  - Back up files on stable (nonvolatile) storage media.
- (File-management techniques in *File-System Interface*, *File-System Implementation*, *File-System Internals*).

4 Mass-storage management

- Computer systems use secondary storage (HDDs, NVM devices) to back up main memory.
- Most programs stored on these devices until loaded into memory.
- Proper secondary storage management is crucial.
- OS responsibilities for secondary storage management:
  - Mounting/unmounting.
  - Free-space management.
  - Storage allocation.
  - Disk scheduling.
  - Partitioning.
  - Protection.
- Tertiary storage (slower, lower cost, higher capacity) for backups, seldom-used data, archival.
  - Examples: magnetic tape, CD, DVD, Blu-ray.
  - OS may manage mounting/unmounting, device allocation, data migration.
- (Techniques in *Mass-Storage Structure*).

5 Cache management

- **Caching** principle: information copied from slower storage (e.g., main memory) to faster temporary storage (**cache**).
- When info needed, check cache first. If not present, use source and copy to cache.
- Internal programmable registers act as high-speed cache for main memory.
- Hardware-only caches (e.g., instruction cache, data caches) outside OS control.
- **Cache management** important due to limited cache size; careful selection of size/replacement policy increases performance.
- *Figure 1.5.1: Characteristics of various types of storage. Table lists level, typical size, implementation technology, access time, bandwidth, managed by, and backed by for various types of storage such as registers, cache, main memory, solid-state disk, and magnetic disk.*
- Information movement between storage levels:
  - Explicit (OS-controlled, e.g., disk to memory).
  - Implicit (hardware function, e.g., cache to CPU).
- Data coherency:
  - Same data may appear in different storage hierarchy levels (e.g., integer A on disk, main memory, cache, internal register).
  - *Figure 1.5.2: Migration of integer A from disk to register. Block diagram shows migration of integer A from magnetic disk to hardware register through main memory and cache.*
  - Multitasking: processes must access most recently updated value.
  - Multiprocessor: update to value in one cache reflected in all other caches (**cache coherency**, usually hardware issue).
  - Distributed: replicas of same file on different computers kept consistent (*Networks and Distributed Systems*).
- (Replacement algorithms for software-controlled caches in *Virtual Memory*).

6 I/O system management

- OS hides peculiarities of specific hardware devices from user.
- **I/O subsystem** consists of:
  - Memory-management component (buffering, caching, spooling).
  - General device-driver interface.
  - Drivers for specific hardware devices.
- Only device driver knows its specific device's peculiarities.
- (Discussed in *I/O Systems*).

7 Section glossary

Term	Definition
Resource manager	OS role in managing computer resources.
Program counter	CPU register indicating main memory location of next instruction to load/execute.
File	Smallest logical storage unit; collection of related info defined by creator.
Caching	Use of temporary data storage areas to improve performance.
Cache management	Management of a cache's contents.
Cache coherency	Coordination of cache contents so update in one cache reflects immediately in others holding that value.
I/O subsystem	I/O devices and kernel part managing I/O.

1.6 Security and protection

- In multi-user, multi-process systems, data access must be regulated.
- Mechanisms ensure processes operate only on authorized resources (files, memory segments, CPU).
- Examples:
  - Memory-addressing hardware (process within its address space).
  - Timer (CPU relinquishing control).
  - Device-control registers (not user accessible).

1 Protection

- **Definition:** Mechanism controlling access of processes/users to system resources.
- Specifies and enforces controls.
- Improves **reliability** by detecting latent errors at subsystem interfaces.
- Prevents contamination of healthy subsystems by malfunctioning ones.
- Unprotected resources cannot defend against unauthorized/incompetent use.
- Distinguishes authorized from unauthorized usage (*Protection* chapter).

2 Security

- **Definition:** Defense of a system from external and internal attacks.
- Attacks include: viruses, worms, **denial-of-service** (DoS) attacks, identity theft, theft of service.
- Prevention can be OS function, policy, or require additional software.
- OS security features are a fast-growing area due to rising incidents (*Security* chapter).

3 User and Group Identification

- Systems must distinguish among all users.
- Most OS maintain list of user names and associated **user identifiers** (**user IDs** or **UIDs**).
- In Windows, this is a **security ID** (**SID**).
- Numerical IDs are unique per user.
- Upon login, authentication determines user ID, associated with all user's processes/threads.
- For readability, user IDs translated back to user names.
- To distinguish among sets of users (e.g., file owner vs. group of readers), systems define **group name** and its users.
- Group functionality implemented as system-wide list of group names and **group identifiers**.
- User can belong to one or more groups, depending on OS design.
- Group IDs included in every associated process and thread.

4 Privilege Escalation

- Users sometimes need to **escalate privileges** for extra permissions (e.g., restricted device access).
- OS provides various methods.
- On UNIX, **setuid** attribute on program runs it with file owner's user ID, not current user's.
- Process runs with this **effective UID** until privileges turned off or terminates.

5 Section glossary

Term	**Definition**
Protection	Mechanism controlling access of processes/users to system resources.
Security	Defense of system from external/internal attacks (e.g., viruses, DoS, identity theft).
User identifier (user ID) (UID)	Unique numerical user identifier.
Security ID (SID)	Windows value to uniquely identify user/group for security.
Group identifier	Identifies group of users for access rights.
Escalate privileges	Gain extra permissions for an activity (e.g., restricted device access).
Setuid	UNIX attribute: program runs with file owner's user ID.
Effective UID	Current UID process uses; can differ from login UID due to privilege escalation.

1.7 Virtualization

- **Definition:** Technology abstracting a single computer's hardware (CPU, memory, disk, network) into multiple execution environments.
- Creates illusion each environment runs on its own private computer.
- Environments can be different OS (e.g., Windows, UNIX) running concurrently.
- User can switch among virtual machines (VMs) like switching among processes.
- Allows OS to run as applications within other OS.
- Industry is vast and growing due to its utility.

1 Emulation vs. Virtualization

Emulation

- Simulates computer hardware in software.
- Used when **source CPU type** differs from **target CPU type**.
- **Example:** Apple's "Rosetta 2" allowed Intel x86 apps on ARM-based Apple Silicon.
- **Drawback:** Heavy performance cost; every machine instruction translated, often to multiple target instructions.
- Emulated code runs much slower than native code.

Virtualization

- OS natively compiled for a specific CPU architecture runs within another OS also native to that CPU.
- **Origin:** First used on IBM mainframes for multiple users to run tasks concurrently on a single-user system.
- **VMware's innovation:** Created virtualization technology as an application on Windows for Intel x86 CPUs.
  - Ran guest copies of Windows or other native x86 OS.
  - Windows acted as the **host operating system**.
  - VMware application was the **virtual machine manager (VMM)**.
  - **VMM** responsibilities: runs guest OS, manages resources, protects guests from each other.
- *Figure 1.7.1: A computer running (a) a single operating system and (b) three virtual machines. Diagram shows a single OS system with hardware, kernel, programming interface, and one process. In contrast, a virtualized system shows hardware, a virtual machine manager, three virtual machines, each with its own kernel, programming interface, and processes.*

2 Growing Importance of Virtualization

- Despite modern OS running multiple applications reliably, virtualization use continues to grow.
- **On laptops/desktops:**
  - Allows users to install multiple OS for exploration.
  - Run applications for OS other than the native host (e.g., macOS x86 running Windows 10 guest).
- **For software development companies:**
  - Run all target OS on a single physical server for development, testing, and debugging.
- **Within data centers:**
  - Common method for executing and managing computing environments.
  - Modern VMMs (e.g., VMware ESX, Citrix XenServer) *are* the host OS, providing services and resource management directly to VM processes.
- **Educational use:** Linux VM provided with this text allows running Linux and development tools regardless of host OS.

3 Section glossary

Term	Definition
Virtualization	Technology abstracting single computer hardware into multiple execution environments; creates illusion each environment runs on private computer.
Virtual machine (VM)	Hardware abstraction allowing virtual computer to execute on physical computer; multiple VMs can run on single physical machine, each with different OS.
Emulation	Methodology enabling process to run when compiled program's source CPU type differs from target CPU type.
Guest	In virtualization, OS running in virtual environment rather than natively on hardware.
Host	In virtualization, VMM location running guest OS; generally, a computer.
Virtual machine manager (VMM)	Computer function managing VM; also **hypervisor**.

1.8 Distributed systems

- Collection of physically separate, possibly heterogeneous computer systems networked together.
- Provides users access to various system resources.
- Increases computation speed, functionality, data availability, and reliability.
- OS may generalize network access as file access (via device driver) or require specific network function invocation.
- Protocols greatly affect utility and popularity.

1 Networks

- **Definition:** Communication path between two or more systems.
- Distributed systems depend on networking.
- Vary by protocols, distances between nodes, and transport media.
- **TCP/IP:** Most common network protocol, fundamental Internet architecture.
  - Supported by most general-purpose OS.
  - Some systems use proprietary protocols.
- OS requires network protocol to have:
  - Interface device (e.g., network adapter) with device driver.
  - Software for data handling.

Network Characterization by Distance

- **Local-area network (LAN):** Connects computers within a room, building, or campus.
- **Wide-area network (WAN):** Links buildings, cities, or countries (e.g., global company offices).
- **Metropolitan-area network (MAN):** Links buildings within a city.
- **Personal-area network (PAN):** Wireless technology (e.g., Bluetooth, 802.11) for communication over several feet (e.g., phone to headset).

Network Media

- Varied media: copper wires, fiber strands, wireless transmissions (satellites, microwave dishes, radios).
- Cellular phone connections create networks.
- Short-range infrared communication can be used.
- Any communication between computers uses or creates a network.
- Networks vary in performance and reliability.

2 Network Operating Systems vs. Distributed Operating Systems

- Some OS extend beyond basic network connectivity.
- **Network operating system:**
  - Provides features like file sharing across network.
  - Includes communication scheme for processes on different computers to exchange messages.
  - Computer runs autonomously but is aware of and communicates with other networked computers.
- **Distributed operating system:**
  - Provides a less autonomous environment.
  - Computers communicate closely enough to create illusion of single OS controlling network.

3 Section glossary

Term	Definition
Network	Communication path between two or more systems.
Transmission Control Protocol/Internet Protocol (TCP/IP)	Most common network protocol; fundamental Internet architecture.
Local-area network (LAN)	Network connecting computers within a room, building, or campus.
Metropolitan-area network (MAN)	Network linking buildings within a city.
Personal-area network (PAN)	Network linking devices within several feet (e.g., on a person).
Wide-area network (WAN)	Network linking buildings, cities, or countries.
Network operating system	OS providing features like network file sharing and inter-process communication; computer runs autonomously but communicates with others.

1.9 Kernel data structures

- Fundamental data structures used extensively in operating systems.

1 Lists, stacks, and queues

- **Array:** Simple data structure; each element accessed directly (e.g., main memory).
- **List:** Collection of data values as a sequence; items accessed in particular order.
  - Commonly implemented as a **linked list**, where items are linked.
  - **Singly linked list:** Each item points to its successor.
  - *Figure 1.9.1: Singly linked list. Diagram shows a sequence of data sets where each data set points to its immediate successor, and the last data set points to null.*
  - **Doubly linked list:** Item can refer to its predecessor or successor.
  - *Figure 1.9.2: Doubly linked list. Diagram shows a sequence of data sets where each data set points to its immediate successor and predecessor.*
  - **Circularly linked list:** Last element refers to the first, not null.
  - Accommodates varying item sizes; allows easy insertion/deletion.
  - **Disadvantage:** Retrieving specified item is $O(N)$ in worst case (linear performance).
  - Used directly by kernel algorithms or for constructing other structures.
- **Stack:** Sequentially ordered data structure using **Last In, First Out (LIFO)** principle.
  - Last item added is first removed.
  - Operations: **push** (insert), **pop** (remove).
  - OS uses stacks for function calls (parameters, local variables, return address pushed/popped).
- **Queue:** Sequentially ordered data structure using **First In, First Out (FIFO)** principle.
  - Items removed in order of insertion.
  - Common in OS (e.g., printer jobs, CPU scheduling tasks).

2 Trees

- Data structure to represent data hierarchically via parent-child relationships.
- **General tree:** Parent may have unlimited children.
- **Binary tree:** Parent has at most two children (left and right).
- **Binary search tree:** Binary tree with ordering: $left\_child \leq right\_child$.
  - *Figure 1.9.4: Binary search tree. Diagram shows a binary search tree with a root item 17, branching into 12 and 35. Item 12 branches into 6 and 14. Item 35 branches into 40, which then branches into 38.*
  - Worst-case search performance is $O(N)$.
  - **Balanced binary search tree:** Contains N items with at most $\lg N$ levels, ensuring $O(\lg N)$ worst-case performance.
  - **Example:** Linux uses a balanced binary search tree (red-black tree) for CPU scheduling.

3 Hash functions and maps

- **Hash function:** Takes data input, performs numeric operation, returns numeric value.
- Numeric value used as index into a table (array) for quick data retrieval.
- Retrieval can be $O(1)$ (constant time), much faster than $O(N)$ for lists.
- Used extensively in OS due to performance.
- **Hash collision:** Two unique inputs result in same output value (same table location).
  - Handled by having a linked list at the table location for all items with same hash value.
  - More collisions reduce efficiency.
- **Hash map:** Associates (maps) [key:value] pairs using a hash function.
  - Apply hash function to key to obtain value.
  - *Figure 1.9.5: Hash map. Diagram shows a hash map with bits numbered 0 to N. A hash function applied to a key on one bit of the hash map obtains a value.*
  - **Example:** User name mapped to password for authentication.

4 Bitmaps

- String of N binary digits representing status of N items.
- **Example:** 0 = resource available, 1 = resource unavailable.
- Value of i^{th} position associated with i^{th} resource.
- **Example bitmap:** ‘0 0 1 0 1 1 1 0 1‘ (resources 2, 4, 5, 6, 8 unavailable; 0, 1, 3, 7 available).
- **Space efficiency:** Significant power; using single bit vs. eight-bit Boolean value saves space.
- Commonly used to represent availability of large number of resources (e.g., disk blocks).

5 Linux kernel data structures

- Kernel source code provides details.
- Linked-list: ‘`<linux/list.h>`‘.
- Queue: known as ‘`kfifo`‘, implementation in ‘`kfifo.c`‘.
- Balanced binary search tree: implemented as **red-black trees**, details in ‘`<linux/rbtree.h>`‘.

6 Section glossary

Term	**Definition**
List	Data structure presenting data values as a sequence.
Linked list	Data structure where items are linked.
Stack	LIFO sequentially ordered data structure.
Queue	FIFO sequentially ordered data structure.
Tree	Data structure representing data hierarchically via parent-child relationships.
General tree	Tree where parent may have unlimited children.
Binary tree	Tree where parent has at most two children.
Binary search tree	Binary tree with ordering: $\textit{left\_child} \leq \textit{right\_child}$.
Balanced binary search tree	Tree with N items, at most $\lg N$ levels, ensuring $O(\lg N)$ worst-case performance.
Red-black tree	Tree with N items, at most $\lg N$ levels, ensuring $O(\lg N)$ worst-case performance.
Hash function	Function taking data input, performing numeric operation, returning numeric value.
Hash map	Data structure mapping [key:value] pairs using a hash function.
Bitmap	String of N binary digits representing status of N items.

1.10 Computing environments

- Operating systems used in various computing environments.

1 Traditional computing

- Lines between traditional computing environments blurred.
- **Typical office environment** evolved:
  - From PCs on network with file/print servers.
  - To **web technologies** and increasing **WAN bandwidth** stretching boundaries.
  - Companies use **portals** for web access to internal servers.
  - **Network computers** (or **thin clients**) for security/easier maintenance.
  - Mobile computers synchronize with PCs for portable use.
  - Mobile devices connect to wireless/cellular networks for web portals.
- **Home computing** evolved:
  - From single PCs with slow modems.
  - To inexpensive fast data connections.
  - Home computers serving web pages and running networks (printers, client PCs, servers).
  - Use of **firewalls** to protect networks from security breaches.
- Historically, systems were either:
  - **Batch** (processed jobs in bulk with predetermined input).
  - **Interactive** (waited for user input).
- **Time-sharing systems:**
  - Multiple users shared time on systems to optimize resource use.
  - Used timer and scheduling algorithms to cycle processes rapidly through CPU.
  - Rare today as traditional time-sharing systems.
  - Same scheduling technique used on desktops, laptops, servers, mobile computers.
  - Processes often owned by single user (or user and OS).
  - User and system processes managed to get a slice of computer time (e.g., multiple windows, web browser processes).

2 Mobile computing

- **Definition:** Computing on handheld smartphones and tablet computers.
- **Physical features:** Portable and lightweight.
- **Functionality evolution:**
  - Historically: sacrificed screen size, memory, functionality for mobile access (email, web browsing).
  - Today: rich features, comparable to laptops; unique/impractical functionality for desktops.
  - Used for music, video, digital books, photos, HD video editing.
- **Unique features leveraged by applications:**
  - **GPS chips:** Determine precise location for navigation apps.
  - **Accelerometers and gyroscopes:** Detect orientation, tilting, shaking (e.g., in games).
  - **Augmented-reality applications:** Overlay info on display of current environment.
- **Connectivity:** Typically use **IEEE 802.11 wireless** or **cellular data networks**.
- **Limitations compared to PCs:**
  - More limited memory capacity and processing speed (e.g., 256 GB storage vs. 8 TB on desktop).
  - Smaller, slower processors with fewer cores due to power consumption.
- **Dominant mobile operating systems:**
  - **Apple iOS:** Designed for Apple iPhone and iPad.
  - **Google Android:** Powers smartphones and tablets from many manufacturers.

3 Client-server computing

- **Definition:** Network architecture where **server systems** satisfy requests from **client systems**.
- A form of specialized distributed system.
- *Figure 1.10.1: General structure of a client-server system. Diagram shows server connected through network to client desktop, client laptop, and client smartphone.*
- **Server system categories:**
  - **Compute-server system:** Client sends request for action (e.g., read data); server executes and sends results (e.g., database server).
  - **File-server system:** Provides file-system interface for clients to create, update, read, delete files (e.g., web server delivering files).

4 Peer-to-peer computing

- **Definition:** Distributed system model where clients and servers are not distinguished; all nodes are **peers**.
- Each node can act as either client (requesting service) or server (providing service).
- **Advantage:** Services provided by several distributed nodes, avoiding server bottlenecks.
- **Node participation:** Must first join the network of peers.
- **Service discovery methods:**
  - **Centralized lookup service:** Node registers service; clients contact lookup service to find providers. Communication then direct. (e.g., Napster).
  - **No centralized lookup service:** Client broadcasts request to all nodes; providers respond directly. Requires a **discovery protocol** (e.g., Gnutella).
  - *Figure 1.10.2: Peer-to-peer system with no centralized service. Image shows a client broadcasting a request to other nodes, and a service provider node responding directly to the client.*
- **Examples:**
  - **Napster** (late 1990s): Centralized server indexed files, actual exchange was peer-to-peer. Shut down due to copyright.
  - **Gnutella:** Decentralized, clients broadcasted requests.
  - **Skype:** Hybrid approach with centralized login server but decentralized peer communication for VoIP calls/messages.

5 Cloud computing

- **Definition:** Delivers computing, storage, and applications as a service across a network.
- Logical extension of **virtualization**, using it as base.
- **Example: Amazon Elastic Compute Cloud (EC2)** with thousands of servers, millions of VMs, petabytes of storage. Users pay for resources.
- **Types of cloud computing:** (not discrete, often combined)
  - **Public cloud:** Available via Internet to anyone willing to pay.
  - **Private cloud:** Run by a company for its own use.
  - **Hybrid cloud:** Includes both public and private components.
  - **Software as a service (SaaS):** Applications (e.g., word processors) available via Internet.
  - **Platform as a service (PaaS):** Software stack ready for application use via Internet (e.g., database server).
  - **Infrastructure as a service (IaaS):** Servers or storage available over Internet (e.g., backup storage).
- **Underlying infrastructure:**
  - Traditional operating systems within cloud infrastructure.
  - **VMMs** (Virtual Machine Managers) manage virtual machines.
  - **Cloud management tools** (e.g., VMware vCloud Director, Eucalyptus) manage VMMs and cloud resources, acting as a new type of OS.
- *Figure 1.10.3: Cloud computing. Diagram shows a public cloud providing IaaS, with cloud services and user interface protected by a firewall.*

6 Real-time embedded systems

- **Embedded computers:** Most prevalent form of computers (car engines, robots, optical drives, microwaves).
- Tend to have very specific tasks.
- Systems usually primitive; OS provides limited features, often little/no user interface.
- Focus on monitoring and managing hardware devices.
- **Variations:**
  - General-purpose computers running standard OS (e.g., Linux) with special applications.
  - Hardware devices with special-purpose embedded OS.

  - Hardware devices with **application-specific integrated circuits (ASICs)** that perform tasks without an OS.
- Expanding use: computerized houses controlling heating, lighting, alarms, appliances; potential for smart refrigerators.
- **Real-time operating systems (RTOS):**
  - Used when rigid time requirements on processor operation or data flow.
  - Often used as control devices in dedicated applications (e.g., scientific experiments, medical imaging, industrial control, weapon systems, fuel-injection).
  - Have well-defined, fixed time constraints.
  - Processing *must* be done within defined constraints, or system fails (e.g., robot arm halting after smashing).
  - Functions correctly only if it returns correct result within time constraints.
  - Contrasts with traditional systems where quick response is desirable but not mandatory.

7 Section glossary

Term	Definition
Portals	Gateways between requestors and services on provider computers.
Network computer	Limited computer understanding only web-based computing.
Thin client	Limited computer (terminal) for web-based computing.
Wireless network	Communication network of radio signals.
Firewall	Computer/appliance/process/router protecting network from security breaches by managing/blocking communications.
Mobile computing	Computing on handheld smartphones/tablets.
Apple iOS	Mobile OS by Apple Inc.
Google Android	Mobile OS by Google Inc.
Server system	System providing services to other computers.
Client system	Computer using services from other computers.
Client-server model	Server provides services to one or more clients.
Compute-server system	Server providing interface for client to request action; executes and sends results.
File-server system	Server providing file-system interface for clients to manipulate files.
Cloud computing	Delivers computing, storage, applications as a service across network.
Amazon Elastic Compute Cloud (EC2)	Amazon's cloud computing instance.
Public cloud	Cloud computing available via Internet to anyone willing to pay.
Private cloud	Cloud computing run by a company for its own use.
Hybrid cloud	Includes both public and private components.
Software as a Service (SaaS)	Applications (e.g., word processors) available as service via Internet.
Platform as a Service (PaaS)	Software stack ready for application use via Internet (e.g., database server).
Infrastructure as a Service (IaaS)	Servers/storage available over Internet (e.g., backup storage).
ASIC	Application-specific integrated circuit (hardware chip) performing tasks without OS.
Real-time operating systems (RTOS)	Systems with rigid time requirements on processor operation/data flow; often control devices in dedicated applications.

1.11 Free and open-source operating systems

- Study of OS made easier by free software and open-source releases.
- Both available in source-code format (not compiled binary).
- **Free software** (*free/libre software*):
  - Source code available.
  - Licensed for no-cost use, redistribution, and modification.
- **Open-source software**:
  - Source code available.
  - Does not necessarily offer free licensing.
- All free software is open source, but some open-source software is not "free."
- **GNU/Linux:** Most famous open-source OS; some distributions are free, others open source only.
- **Microsoft Windows:** Closed-source, proprietary software (Microsoft owns, restricts use, protects source code).
- **Apple macOS:** Hybrid approach (open-source **Darwin** kernel, proprietary closed-source components).
- Starting with source code allows programmers to produce executable binary code.
- Examining source code is an excellent learning tool: students can modify, compile, and run changes.
- **Benefits of open-source operating systems:**
  - Community of programmers contribute (write, debug, analyze, support, suggest changes).
  - Arguably more secure (many eyes viewing the code).
  - Bugs found and fixed faster due to number of users/viewers.
  - Commercial companies (e.g., Red Hat) benefit from open-sourcing code (revenue from support contracts, hardware sales).

1 History

- Early computing (1950s): software generally came with source code (e.g., MIT's Tech Model Railroad Club, Homebrew user groups, DECUS).
- 1970: Digital's OS distributed as source code with no restrictions/copyright.
- Companies later limited software use to authorized/paying customers.
- Releasing only binary files (not source code) protected code/ideas.
- By 1980: proprietary software was usual case.

2 Free operating systems

- To counter proprietary trend, **Richard Stallman** (1984) started developing **GNU** ("GNU's Not Unix!"): a free, UNIX-compatible OS.
- "Free" refers to freedom of use, not price.
- Free-software movement holds users entitled to four freedoms:
  1. To freely run the program.
  2. To study and change the source code.
  3. To give or sell copies.
  4. To give or sell copies with or without changes.
- 1985: Stallman published **GNU Manifesto** (argues all software should be free).
- Formed **Free Software Foundation (FSF)** to encourage free software use/development.
- FSF uses "copyleft" (Stallman invention): licensing giving four essential freedoms, conditioned on preserving them in redistribution.
- **GNU General Public License (GPL):** Common free software license; requires source code distribution with binaries, and all copies (modified) released under same GPL.
- Creative Commons "Attribution Sharealike" license is also a copyleft license.

3 GNU/Linux

- By 1991: GNU OS nearly complete (compilers, editors, utilities, libraries, games), but kernel not ready.
- 1991: **Linus Torvalds** (Finland) released rudimentary UNIX-like kernel using GNU compilers/tools, invited worldwide contributions.
- Internet enabled rapid "Linux" growth (weekly updates, thousands of programmers).
- 1991: Linux not free (noncommercial redistribution only).

- 1992: Torvalds rereleased Linux under GPL, making it free software (and "open source").
- **GNU/Linux:** Full OS (kernel properly Linux, includes GNU tools).
- Spawned hundreds of unique **distributions** (custom builds): Red Hat, SUSE, Fedora, Debian, Slackware, Ubuntu.
- Distributions vary in function, utility, installed applications, hardware support, user interface, and purpose.
- Example: Red Hat Enterprise Linux for large commercial use.
- Example: **PCLinuxOS** is a **Live CD** (or **LiveDVD**) – OS bootable/run from CD-ROM/DVD without installation.
- Running Linux on Windows (or other) system using virtualization:
  1. Download/install free **Virtualbox VMM** tool (`https://www.virtualbox.org/`).
  2. Install OS from scratch (CD image) or use pre-built images (`http://virtualboxes.org/images/`).
  3. Boot the virtual machine within Virtualbox.
- Alternative: free program **Qemu** (`http://wiki.qemu.org/Download/`), includes `qemu-img` for converting Virtualbox images.
- This text provides a virtual machine image of GNU/Linux running Ubuntu (includes source code, dev tools).

4 BSD UNIX

- Longer, more complicated history than Linux.
- Started 1978 as derivative of AT&T's UNIX.
- UCB releases in source/binary form, but not open source (AT&T license required).
- Development slowed by AT&T lawsuit.
- 1994: Fully functional, open-source **4.4BSD-lite** released.
- Many distributions: **FreeBSD**, **NetBSD**, **OpenBSD**, **DragonflyBSD**.
- To explore FreeBSD source code: download VM image (Virtualbox), source in `/usr/src/`, kernel in `/usr/src/sys`, VM code in `/usr/src/sys/vm`.
- View source code online: `https://svnweb.freebsd.org`.
- Source code controlled by a **version control system** (e.g., "subversion" `https://subversion.apache.org/source-code`).
  - Allows user to "pull" entire source code tree and "push" changes.
  - Provides file history and conflict resolution.
- Another version control system: **git** (`http://www.git-scm.com`), used for GNU/Linux and other programs.
- **Darwin:** Core kernel component of macOS, based on BSD UNIX, open-sourced (`http://www.opensource.apple.com/`).
  - Every macOS release has open-source components posted there.
  - Package containing kernel begins with "xnu."
  - Apple provides developer tools, documentation, support (`http://developer.apple.com`).

5 The study of operating systems

- Never been easier/more interesting to study OS.
- Open-source movement: many OS available in source and binary (Linux, BSD UNIX, Solaris, part of macOS).
- Source code allows studying OS from inside out (examining code vs. documentation/behavior).
- OS no longer commercially viable have been open-sourced (study how systems operated with fewer CPU, memory, storage resources).
- Rise of **virtualization** (mainstream, often free) allows running many OS on one core system.
  - Example: VMware (`http://www.vmware.com`) provides free "player" for Windows; hundreds of free "virtual appliances" can run.
  - Example: Virtualbox (`http://www.virtualbox.com`) provides free, open-source VMM on many OS.
  - Students can try hundreds of OS without dedicated hardware.
- Simulators of specific hardware also available (run OS on "native" hardware within modern computer/OS).
  - Example: DECSYSTEM-20 simulator on macOS can boot TOPS-20, load source tapes, modify/compile new TOPS-20 kernel.
- Open-source OS make it easier to move from student to OS developer (create new OS distribution).
- Access to source code limited only by interest, time, and disk space.

6 Solaris

- Commercial UNIX-based OS of Sun Microsystems.
- Originally Sun's **SunOS** based on BSD UNIX.
- 1991: Sun moved to AT&T's System V UNIX as its base.
- 2005: Sun open-sourced most Solaris code as the **OpenSolaris** project.
- 2010: Oracle's purchase of Sun left project state unclear.
- **Project Illumos:** Groups interested in OpenSolaris expanded features; basis for several products (`http://wiki.illumos.org`).

7 Open-source systems as learning tools

- Free-software movement drives programmers to create thousands of open-source projects (including OS).
- Portals: `http://freshmeat.net/`, `http://distrowatch.com/`.
- Students use source code as learning tool: modify/test programs, help find/fix bugs, explore mature systems (OS, compilers, tools, UIs).
- Availability of source code for historic projects (e.g., Multics) helps understanding and building knowledge for new projects.
- Diversity of open-source OS (e.g., GNU/Linux, BSD UNIX) with different goals, utility, licensing, purpose.
- Cross-pollination occurs (e.g., major OpenSolaris components ported to BSD UNIX), allowing rapid improvements.
- Advantages of free software and open sourcing likely to increase number/quality of open-source projects and their adoption.

8 Section glossary

Term	Definition
Free operating system	OS released under license making source code available, allowing no-cost use, redistribution, modification.
Open-source operating system	OS/program available in source-code format (not compiled binary).
Closed-source	OS/program available only in compiled binary code format.
Reverse engineering	Converting compiled binary file into human-readable format.
GNU General Public License (GPL)	License codifying copylefting; common free software license.
GNU/Linux (aka Linux)	Open-source OS: GNU foundation components + Linus Torvalds' kernel.
Distribution	Release of an OS version.
LiveCD	OS bootable/run from CD-ROM (or any media) without installation.
LiveDVD	OS bootable/run from DVD (or any media) without installation.
UnixBSD	UNIX derivative based on UCB work.
Version control system	Software managing distributions; allows ”pulling” source tree, ”pushing” changes.
Git	Version control system for GNU/Linux and other programs.
Solaris	Commercial UNIX-based OS by Sun Microsystems (now Oracle); active open source version called Illumos.
SunOS	Predecessor of Solaris by Sun Microsystems Inc.