Operating Systems Notes

May 31, 2025

Contents

1.1 Introduction to Operating Systems

1 Purpose and Core Responsibilities

- **Intermediary:** OS sits between user & applications and hardware, hiding details while exposing 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 a 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 a stream of memory addresses that 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, etc.
  - **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 their own minimal operating systems that run continuously beneath the main kernel.

6 Why Study Operating Systems?

- All code runs on an OS; knowing its policies & APIs yields safer, faster, portable software.
- **OS flavours:** desktop (Windows, macOS), server (Linux, Windows Server), mobile (Android, iOS), and embedded/real-time systems each emphasise different goals.
- **Open-source examples:** Linux, FreeBSD, and 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 and efficiently; manages hardware resources.
Kernel	Always runs after boot; provides services.
System program	Loaded at boot time, becomes system daemon; provides services outside the kernel.
Middleware	Optional/varies; graphics, DB, extra utilities.
Resource allocator	Part of OS (system view); arbitrates CPU, memory, I/O, etc.
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.

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.
- NVS divides into *mechanical* media (HDD, optical, tape) and *electrical* non-volatile memory (NVM: SSD, flash).
- **Firmware / EEPROM** stores bootstrap program.
- **Interrupt masking:** the kernel may briefly disable maskable interrupts while executing critical sections to maintain data coherence.

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.

1.3 Computer-system architecture

- Computer systems are organized based on the number of general-purpose processors.

1 Single-processor systems

- **Definition:** One general-purpose CPU with a single processing core.
- Historically common, but few contemporary systems fit this definition.
- **CPU Core:** Executes instructions, contains registers for local data.
- Main CPU core executes general-purpose instruction sets (including processes).
- May include **special-purpose processors** (e.g., disk, keyboard, graphics controllers).
  - Run limited instruction sets; do not run processes.
  - Can be OS-managed (OS sends tasks, monitors status).
  - Example: Disk-controller microprocessor handles disk queue/scheduling, reducing main CPU overhead.
  - Example: Keyboard microprocessor converts keystrokes to codes for CPU.
  - Can be low-level hardware components, operating autonomously without OS communication.
- Use of special-purpose microprocessors does not classify a system as multiprocessor.

2 Multiprocessor systems

- Dominant in modern computing (mobile devices to servers).
- Traditionally: Two or more processors, each with a single-core CPU.
- Share computer bus, sometimes clock, memory, and peripheral devices.
- **Primary advantage:** Increased throughput (more work in less time).
- Speed-up ratio with N processors is less than N due to overhead and contention for shared resources.
- **Symmetric Multiprocessing (SMP):**
  - Each peer CPU processor performs all tasks (OS functions, user processes).
  - Each CPU has its own registers and private/local cache.
  - All processors share physical memory via the system bus.
  - *Figure 1.3.1: Symmetric multiprocessing architecture. Diagram shows main memory connected to two processors, each with its own CPU containing registers and cache.*
  - Benefit: N processes can run simultaneously on N CPUs without significant performance degradation.
  - Inefficiencies from idle/overloaded CPUs are mitigated by sharing data structures.
  - Allows dynamic sharing of processes and resources (e.g., memory) among processors, reducing workload variance.
  - Requires careful design (e.g., CPU Scheduling, Synchronization Tools).
- **Multicore systems:**
  - Modern evolution: Multiple computing cores reside on a single chip.
  - More efficient than multiple single-core chips due to faster on-chip communication.
  - Uses significantly less power than multiple single-core chips (critical for mobile/laptops).
  - Example: Dual-core design (Figure 1.3.2)
    * Each core has its own register set and L1 cache.
    * Shared L2 cache is local to the chip.
    * Combines local (smaller, faster) and shared (higher-level) caches.
    * *Figure 1.3.2: A dual-core design with two cores on the same chip. Diagram shows main memory connected to L2 cache of processor. Processor has two CPU cores, each core has its own registers and L1 cache and shares L2 cache of processor.*
    * An N-core processor appears as N standard CPUs to the OS.
    * Demands efficient use of processing cores from OS/application designers (Threads & Concurrency).
    * Supported by virtually all modern OS (Windows, macOS, Linux, Android, iOS).

Definitions of computer system components

- **CPU:** Hardware that executes instructions.
- **Processor:** A physical chip containing one or more CPUs.
- **Core:** Basic computation unit of the CPU.
- **Multicore:** Multiple computing cores on the same CPU.
- **Multiprocessor:** Multiple processors within the same CPU chip or system.

- General term *CPU* refers to a single computational unit.
- *Core* and *multicore* refer specifically to one or more cores on a CPU.

Non-Uniform Memory Access (NUMA)

- An alternative to scaling multiprocessor systems when bus contention becomes a bottleneck.
- Each CPU (or group of CPUs) has its own **local memory** accessed via a small, fast local bus.
- CPUs are connected by a **shared system interconnect**; all CPUs share one physical address space.
- *Figure 1.3.3: NUMA multiprocessing architecture. Diagram shows NUMA architecture which includes four interconnected CPUs, each attached with its own local memory.*
- **Advantage:** Fast local memory access with no contention over the system interconnect.
- **Result:** NUMA systems scale more effectively with additional processors.
- **Potential drawback:** Increased latency for remote memory access across the system interconnect (e.g., $CPU0$ accessing $CPU3$'s local memory is slower).
- OS minimizes NUMA penalty through careful CPU scheduling and memory management.
- Increasingly popular on servers and high-performance computing systems due to scalability.

Blade servers

- Multiple processor boards, I/O boards, and networking boards placed in the same chassis.
- Differ from traditional multiprocessor systems: each blade-processor board boots independently and runs its own OS.
- Can consist of multiple independent multiprocessor systems.

3 Clustered systems

- Another type of multiprocessor system, gathering multiple CPUs.
- Composed of two or more individual systems (**nodes**) joined together; each node is typically a multicore system.
- Considered **loosely coupled**.
- Definition: Clustered computers share storage and are closely linked via LAN or faster interconnect (e.g., InfiniBand).
- **Primary use:** Provide **high-availability service** (service continues even if one or more systems fail).
  - Achieved by adding redundancy.
  - Cluster software runs on nodes; each node monitors others.
  - If a monitored machine fails, the monitoring machine takes ownership of its storage and restarts applications.
  - Users/clients experience only a brief interruption.
- High availability increases reliability.
- **Graceful degradation:** System's ability to continue providing service proportional to surviving hardware.
- **Fault-tolerant systems:** Can suffer a single component failure and continue operation (requires detection, diagnosis, correction).
- **Clustering structures:**
  - **Asymmetric clustering:** One machine in **hot-standby mode** (monitors active server); if active fails, hot-standby becomes active.
  - **Symmetric clustering:** Two or more hosts run applications and monitor each other; more efficient as it uses all hardware, but requires multiple applications.
- **PC motherboard example:**
  - A fully functioning computer once slots are populated (processor socket, DRAM sockets, PCIe bus slots, I/O connectors).
  - Lowest-cost general-purpose CPUs have multiple cores.
  - Some motherboards have multiple processor sockets (for NUMA systems).
  - *Image shows a computer processor with arrows pointing to: DRAM slots, processor socket, PCI bus slots, and various I/O and power connectors.*
- **High-Performance Computing (HPC) environments:**
  - Clusters provide significantly greater computational power than single-processor/SMP systems.
  - Run applications concurrently on all computers in the cluster.
  - Requires applications to be specifically written for clusters (e.g., **parallelization**).
  - **Parallelization:** Divides a program into separate components that run in parallel on individual cores/computers.
  - Results from nodes are combined into a final solution.
- **Other cluster forms:**
  - **Parallel clusters:** Multiple hosts access the same data on shared storage.
  - Often requires special software/application releases (e.g., Oracle Real Application Cluster).

- Uses **distributed lock manager (DLM)** for access control and locking.
- Clustering over Wide-Area Network (WAN).

- Cluster technology is rapidly evolving (supporting thousands of systems, separated by miles).
- Enabled by **Storage-Area Networks (SANs)**: allow many systems to attach to a pool of storage.
- If applications/data are on the SAN, cluster software can assign the application to any host attached to the SAN.
- *Figure 1.3.4: General structure of a clustered system. Diagram shows structure of clustered system in which storage-area network is connected with set of interconnected computers.*

4 Section glossary

Term	**Definition**
Core	Within a CPU, the component that executes instructions.
Multiprocessor systems	Systems with two or more hardware processors (CPU cores) in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices.
Symmetric multiprocessing (SMP)	Multiprocessing where each processor performs all tasks, including OS tasks and user processes.
Multicore	Multiple processing cores within the same CPU chip or within a single system.
Multiprocessor	Multiple processors within the same CPU chip or within a single system.
Shared system interconnect	A bus connecting CPUs to memory such that all CPUs can access all system memory; basis for NUMA systems.
Non-Uniform Memory Access (NUMA)	Architecture where memory access time varies based on which core the thread is running on (e.g., core interlink slower than direct DIMM access).
Blade server	Computer with multiple processor, I/O, and networking boards in same chassis; each board boots independently and runs its own OS.
Clustered system	Gathers multiple CPUs; composed of two or more individual systems (nodes) joined together.
High-availability	Service continues even if one or more systems in the cluster fail.
Graceful degradation	System's ability to continue providing service proportional to surviving hardware.
Fault-tolerant system	System that can suffer a single component failure and still continue operation.
Asymmetric clustering	One machine in hot-standby mode monitors active server; takes over if active fails.
Hot-standby mode	Computer in cluster monitors active server, becomes active if it fails.
Symmetric clustering	Two or more hosts run applications and monitor each other.
High-Performance Computing (HPC)	Computing facility for large number of resources used by software designed for parallel operation.
Parallelization	Dividing a program into separate components that run in parallel on individual cores/computers.
Distributed lock manager (DLM)	Function used by clustered system for access control and locking to prevent conflicting operations.
Storage-Area Network (SAN)	Local-area storage network allowing multiple computers to connect to one or more storage devices.

1.4 Operating-system operations

- Provides the environment within which programs are executed.
- Internally, OS vary greatly, but share commonalities.

1 Initial Program and Kernel Loading

- Computer needs an initial program to run when powered up or rebooted.
- This **bootstrap program** is typically simple and stored in **firmware**.
- Initializes all system aspects: CPU registers, device controllers, memory contents.
- Must locate and load the **operating-system kernel** into memory.
- Once kernel is loaded and executing, it provides services.
- Some services are provided by **system programs** loaded at boot time, becoming **system daemons**.
- Example: On Linux, `systemd` starts many other daemons.
- After booting, the system waits for events (e.g., interrupts or traps).

Hadoop

- Open-source software framework for distributed processing of **big data** in clustered systems.
- Designed to scale from a single system to thousands of computing nodes.
- Assigns tasks to nodes, manages communication for parallel computations, and coalesces results.
- Detects and manages node failures, providing efficient and reliable distributed computing.
- Organized around three components:
  1. A distributed file system for managing data and files across nodes.
  2. The **YARN** (”Yet Another Resource Negotiator”) framework for resource management and task scheduling.
  3. The **MapReduce** system for parallel data processing across nodes.
- Primarily runs on Linux systems.
- Applications can be written in PHP, Perl, Python, and Java (popular due to Java libraries for MapReduce).

2 Interrupts and Traps

- Events are almost always signaled by the occurrence of an **interrupt**.
- A **trap** (or an **exception**) is a software-generated interrupt.
  - Caused by an error (e.g., division by zero, invalid memory access).
  - Caused by a specific request from a user program for an operating-system service, by executing a **system call**.

3 Multiprogramming and Multitasking

- Important aspects of OS: ability to run multiple programs.
- Single program cannot keep CPU or I/O devices busy at all times.
- **Multiprogramming:**
  - Increases CPU utilization and user satisfaction.
  - OS keeps several **processes** (programs in execution) in memory simultaneously.
  - When a process waits for a task (e.g., I/O), OS switches to another process.
  - CPU is never idle as long as at least one process needs to execute.
  - *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, providing fast **response time**.
  - Handles slow interactive I/O (e.g., user typing speed).
  - Requires:
    - Memory management (*Main Memory*, *Virtual Memory* chapters).
    - **CPU scheduling** (choosing which process runs next).
    - Protection mechanisms to limit processes’ ability to affect one another.
    - **Virtual memory**: allows execution of processes not completely in memory; enables running programs larger than physical memory; abstracts main memory into a large, uniform array of storage, separating **logical memory** from physical memory.
    - File system (*File-System Interface*, *File-System Implementation*, *File-System Internals* chapters).

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

4 Dual-mode and Multimode Operation

- Ensures incorrect/malicious programs cannot cause other programs or the OS to execute incorrectly.
- Distinguishes between execution of operating-system code and user-defined code.
- Hardware support provides differentiation among various modes of execution.
- At least two separate modes of operation:
  - **User mode**: System executing on behalf of a user application.
  - **Kernel mode** (also called **supervisor mode**, **system mode**, or **privileged mode**): System executing on behalf of the operating system.
- A **mode bit** is added to hardware to indicate current mode: 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 system boot time, hardware starts in kernel mode.
- OS is loaded and starts user applications in user mode.
- Whenever a trap or interrupt occurs, hardware switches from user mode to kernel mode (mode bit to 0).
- OS always gains control in kernel mode.
- System always switches to user mode (mode bit to 1) before passing control to a 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 provides protection for OS from errant users and users from one another.
- **Privileged instructions**: Machine instructions that may cause harm, executable only in kernel mode.
  - Attempt to execute in user mode → hardware treats as illegal and traps to OS.
  - Examples: switch to kernel mode, I/O control, timer management, interrupt management.
- Modes can be extended beyond two:
  - Intel processors: four separate **protection rings** (ring 0 kernel, ring 3 user).
  - ARM v8 systems: seven modes.
  - CPUs supporting virtualization: separate mode for **Virtual Machine Manager (VMM)**.
- Life cycle of instruction execution: OS (kernel mode) → user application (user mode) → back to OS via interrupt, trap, or system call.
- Most contemporary OS (Microsoft Windows, Unix, Linux) use dual-mode for protection.

System Calls

- Provide means for a user program to ask the OS to perform tasks reserved for the OS.
- Invoked via a trap to a specific location in the interrupt vector.
- Can be executed by a generic 'trap' instruction or a specific 'syscall' instruction.
- Treated by hardware as a software interrupt.
- Control passes to a service routine in the OS, and mode bit is set to kernel mode.
- Kernel examines interrupting instruction, determines system call, verifies parameters, executes request, and returns control.

Program Errors

- Hardware protection detects errors that violate modes (e.g., illegal instruction, accessing memory not in user's address space).
- Hardware traps to the OS.
- OS terminates the program abnormally, provides an error message, and may dump program memory to a file for examination/correction.

5 Timer

- Ensures OS maintains control over the CPU.
- Prevents user programs from getting stuck in infinite loops or failing to return control to the OS.
- Can be set to interrupt the computer after a specified period (fixed or variable).
- Variable timer implemented by a fixed-rate clock and a counter.
  - OS sets the counter.
  - Every clock tick, counter is decremented.
  - When counter reaches 0, an interrupt occurs.
- OS ensures timer is set before turning control to user.
- If timer interrupts, control transfers automatically to OS (may treat as fatal error or give more time).
- Instructions that modify the content of the timer are privileged.

Linux timers

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

6 Section glossary

Term	Definition
Big data	Extremely large sets of data; distributed systems are well suited to working with big data.
MapReduce	Google-created big data programming model and implementation for parallel processing across nodes in a distributed cluster.
System daemon	Service provided outside the kernel by system programs loaded at boot time and running continuously.
Trap	Software interrupt caused by an error or a specific request from a user program for an operating-system service.
Exception	Software-generated interrupt caused by an error or a specific request from a user program for an operating-system service.
System call	Software-triggered interrupt allowing a process to request a kernel service.
Multiprogramming	Technique that increases CPU utilization by organizing jobs so that the CPU always has a job to execute.
Process	A program loaded into memory and executing.
Multitasking	Concurrent performance of multiple jobs; CPU switches frequently among them for fast response time.
Response time	Amount of time it takes the system to respond to user action.
CPU scheduling	Process by which the system chooses which job will run next if several jobs are ready to run.
Virtual memory	Technique that allows execution of a process not completely in memory; also, separation of computer memory address space from physical into logical.
Logical memory	Memory as viewed by the user; usually a large uniform array, not matching physical memory in virtual memory systems.
User mode	CPU mode for executing user processes in which some instructions are limited or not allowed.
Kernel mode	CPU mode in which all instructions are enabled; the kernel runs in this mode.
Supervisor mode	CPU mode in which all instructions are enabled; the kernel runs in this mode.
System mode	CPU mode in which all instructions are enabled; the kernel runs in this mode.