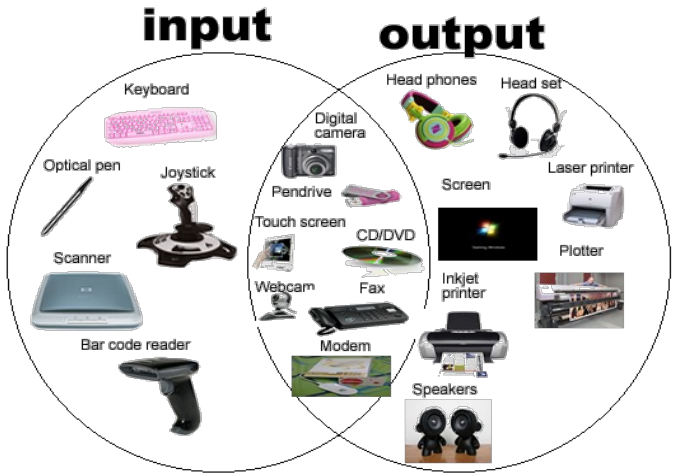
Week5

I/O systems

I/O Systems

- The I/O (Input/Output) System is a subsystem of the operating system that manages communication between the computer and external devices.
- It allows the CPU and memory to interact with devices such as disks, keyboards, printers, displays, and network cards.
- The I/O system includes both hardware components (devices, controllers, ports) and software components (drivers, interrupt handlers, buffers, and kernel services).

Example: When a user saves a file, the I/O system coordinates between the application, the file system, and the disk hardware — ensuring data is correctly written and confirmed.



Role of I/O System

- **Device Management:** Controls and coordinates all input and output devices.
- **Abstraction:** Provides a uniform interface to access various types of devices, hiding hardware details from users and applications.
- **Efficiency:** Maximizes device performance and system throughput while minimizing CPU involvement.
- **Error Handling:** Detects, reports, and recovers from I/O errors.
- **Scheduling and Buffering:** Organizes access to devices and manages data temporarily to ensure smooth operation.
- **Protection and Security:** Ensures controlled access to devices and data.

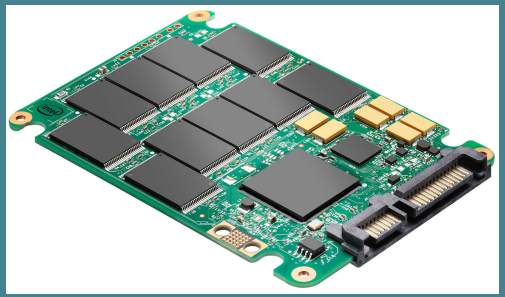
Types of I/O Devices

1. **Block Devices**

- Store data in fixed-size blocks (usually 512 bytes or more).
- Allow random access to any block on the device.
- Commonly used for secondary storage.

Examples: Hard Disk Drives (HDDs), Solid-State Drives (SSDs), USB Flash Drives, CD/DVD drives.

OS Interface: Reads and writes are done in blocks through the file system.



Types of I/O Devices

2. Character Devices

- Handle data as a stream of characters, one at a time.
- Do not support random access; data must be read/written sequentially.
- Typically used for human-interface or communication devices.

Examples: Keyboard, Mouse, Printer, Serial Ports, Terminals.

OS Interface: Uses system calls like read() and write() for character streams.

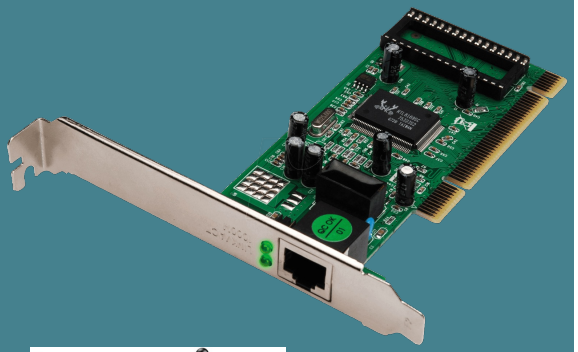
Types of I/O Devices

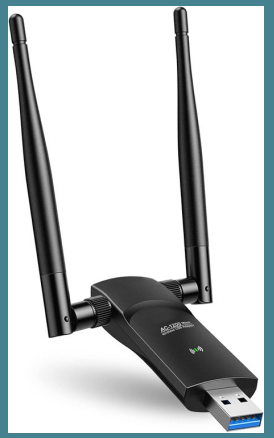
3. Network Devices

- Enable data communication between computers.
- Transfer data packets through a network interface.

Examples: Network Interface Card (NIC), Modem, Wi-Fi Adapter.

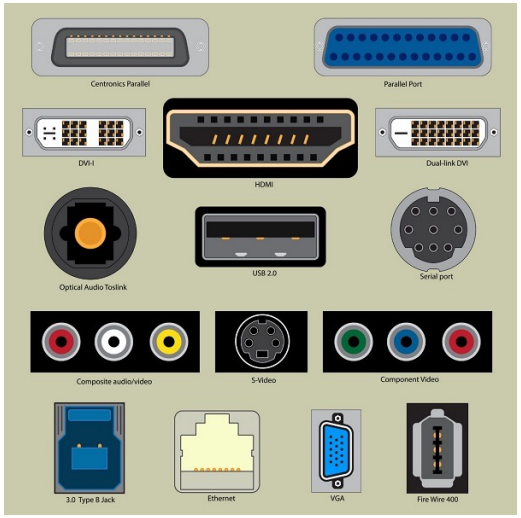
OS Interface: Managed by network protocols (TCP/IP) and handled via socket programming interfaces.





I/O Ports

- **I/O Ports** is a **connection point** through which data is transferred between the CPU and an external device.
- It acts as the interface between the computer and the peripheral hardware.



Functions:

- Send or receive data to/from a device
- Exchange control and status information
- Provide communication pathways for device drivers

Types of Ports:

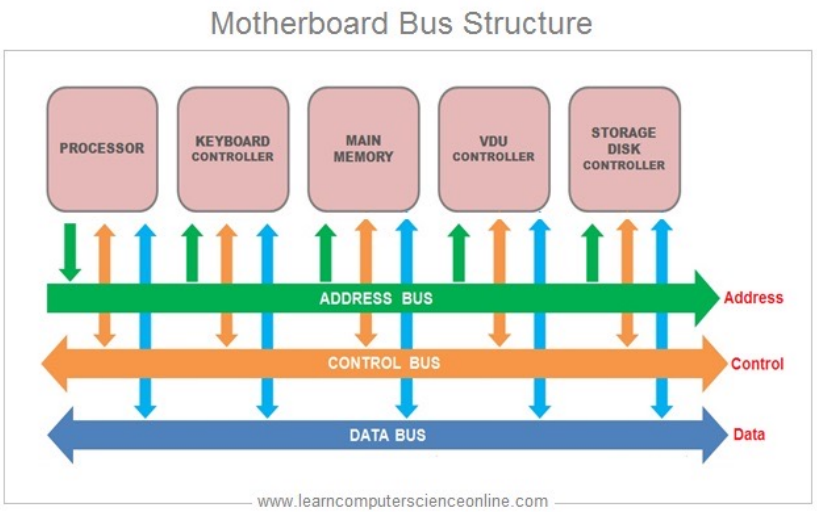
- **Serial Port (COM port):** Transfers data one bit at a time (e.g., old mouse, modem)
- **Parallel Port:** Transfers multiple bits simultaneously (e.g., older printers)
- **USB Port (Universal Serial Bus):** Common modern interface for storage, keyboard, mouse, etc.
- **Network Port:** Connects to network devices (e.g., Ethernet port, Wi-Fi adapter)

System Buses

- A bus is a communication pathway that connects components inside the computer — including the CPU, memory, and I/O devices — for transferring data and control signals.

Main Types of Buses:

- **Data Bus** – Transfers actual data between components
- **Address Bus** – Carries memory or I/O addresses specifying data locations
- **Control Bus** – Carries control signals (e.g., read/write commands, interrupts)



Device Controllers

- A device controller is a hardware component that manages a specific type of device and acts as a translator between the device and the operating system.

Functions:

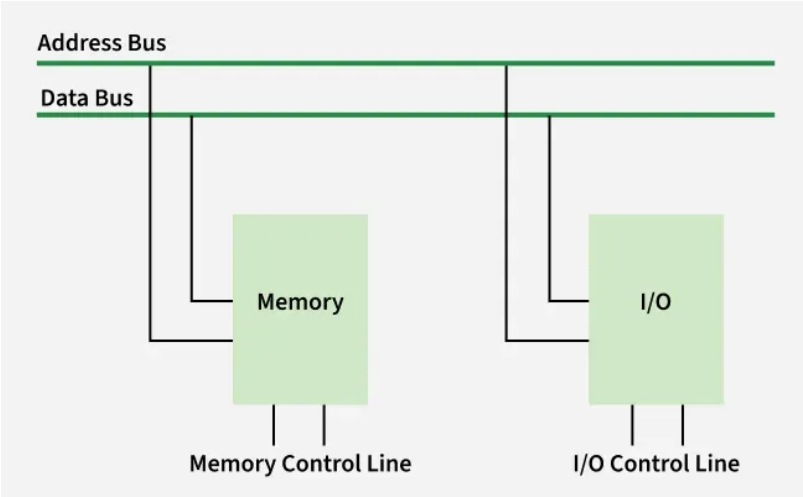
- Converts generic I/O commands from the CPU into device-specific actions
- Controls the physical operation of the device
- Contains data registers and control registers
- Uses interrupts to signal completion or errors

Disk Controller

Memory-Mapped I/O vs. **Isolated I/O**

- Both **Memory-Mapped I/O (MMIO)** and **Isolated I/O (Port-Mapped I/O)** are techniques used by the CPU to communicate with peripheral devices.
- They define how the CPU **accesses device registers** (control, status, data) for input and output operations.

Isolated I/O



Memory-Mapped I/O

Feature / Aspect	Memory-Mapped I/O (MMIO)	Isolated I/O (Port-Mapped I/O or PMIO)
Basic Concept	I/O devices share the **same address space** as main memory.	I/O devices use a **separate address space** from main memory.
Addressing Method	Devices are accessed using **memory addresses**.	Devices are accessed using **I/O port addresses**.
Access Instructions	Uses **standard memory instructions** (LOAD, STORE, MOV).	Uses **special I/O instructions** (IN, OUT).
Programming Simplicity	Easier to program — devices appear as memory variables.	More complex — requires specific assembly instructions.
Hardware Design	Simpler — only one address space to manage.	More complex — needs separate I/O control lines and decoding logic.
CPU Address Space Usage	Consumes part of the CPU's memory address space.	Uses a distinct I/O address space, preserving memory addresses.
Compatibility	Common in **RISC architectures** (e.g., ARM, MIPS).	Common in **x86 architectures** (Intel, AMD).
Cache Interaction	May require cache control to prevent inconsistencies.	No cache conflict, as it's separate from memory.
Performance	Typically **faster** due to unified address access.	Slightly **slower**, since I/O instructions are separate.
Example Access	*(volatile int*)0x80000000 = 1;	OUT 0x60, AL
Example Devices	Used in embedded systems, GPUs, ARM SoCs.	Used in older PCs for keyboards, printers, etc.

Interrupts

- An interrupt is a signal sent to the CPU that temporarily stops its current execution so the CPU can respond to an event (usually from an I/O device or the system itself).
- It allows the CPU to handle asynchronous events efficiently — such as keyboard input, disk read completion, or hardware errors.

Example: When you press a key, the keyboard sends an interrupt to tell the CPU that input data is ready.

Purpose of Interrupts

- Improve **efficiency** — the CPU doesn't have to keep checking (polling) for device status.
- Enable **asynchronous I/O processing** — devices can operate concurrently with the CPU.
- Provide **faster response** to important events.

Interrupt Handling Process

Steps:

1. **Interrupt occurs** — a device or software sends an interrupt signal.
2. **CPU finishes the current instruction.**
3. **CPU saves the current state** (program counter, registers).
4. **Control transfers to the Interrupt Handler (ISR)** — a special routine in the OS.
5. **ISR services the interrupt** (e.g., read data from a device).
6. **CPU restores the saved state** and resumes the interrupted program.

Type	Triggered By	Example
Hardware Interrupt	External hardware devices	Keyboard input, mouse movement, disk I/O completion
Software Interrupt	Generated by programs or OS system calls	System calls, traps, exceptions
Maskable Interrupt (IRQ)	Can be turned off (masked) by the CPU	Normal device requests
Non-Maskable Interrupt (NMI)	Cannot be ignored by the CPU	Hardware failure, emergency stop
Internal (Exception)	Error or condition within CPU	Divide by zero, invalid opcode

The events from 0 to 31, which are nonmaskable
The events from 32 to 255, which are maskable

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

Direct Memory Access (DMA)

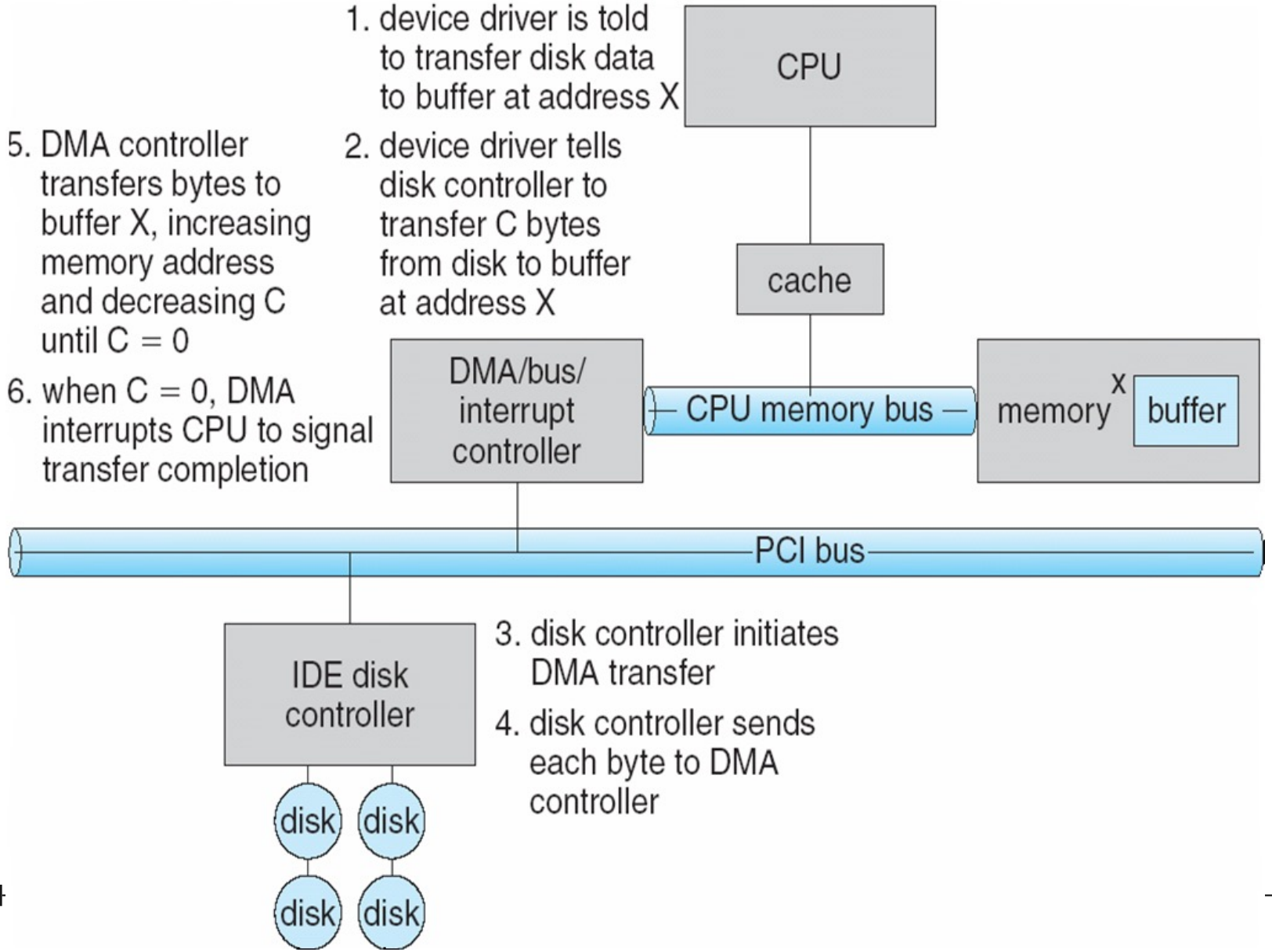
- DMA is a technique that allows I/O devices to transfer data directly to or from main memory without continuous CPU involvement.
- It is managed by a special hardware component called the DMA Controller (DMAC).

Example:

When copying a large file from disk to memory, DMA lets the transfer happen automatically — freeing the CPU to do other work.

Without DMA: Device → CPU → Memory

With DMA:
Device → DMA Controller → Memory
↓
(Interrupts CPU after completion)


1. device driver is told to transfer disk data to buffer at address X
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
6. when C = 0, DMA interrupts CPU to signal transfer completion
CPU
cache
DMA/bus/ interrupt controller
CPU memory bus
memory
X
buffer
PCI bus
IDE disk controller
disk
disk
disk
disk

Application I/O Interface

- Interface between user programs and the I/O subsystem
- It provides standardized methods (system calls) for applications to perform input and output operations without needing to know hardware details.

Operation	System Call Example (C / UNIX)	Description
Open	`fd = open("file.txt", O_RDONLY);`	Opens a file or device and returns a file descriptor.
Read	`read(fd, buffer, size);`	Reads data from a file or device into memory.
Write	`write(fd, buffer, size);`	Writes data from memory to a file or device.
Close	`close(fd);`	Closes a file or device.
Seek	`lseek(fd, offset, SEEK_SET);`	Moves the file pointer to a specific position.
Control	`ioctl(fd, command, arg);`	Sends device-specific control commands.

Application I/O Interface

Blocking vs. Non-Blocking I/O

Mode	Description	Example
Blocking I/O	The process **waits** until the I/O operation completes.	`read()` waits until data is available.
Non-Blocking I/O	The process continues execution **without waiting**; it checks later if the I/O is done.	Used in real-time and event-driven systems.

Synchronous vs. Asynchronous I/O

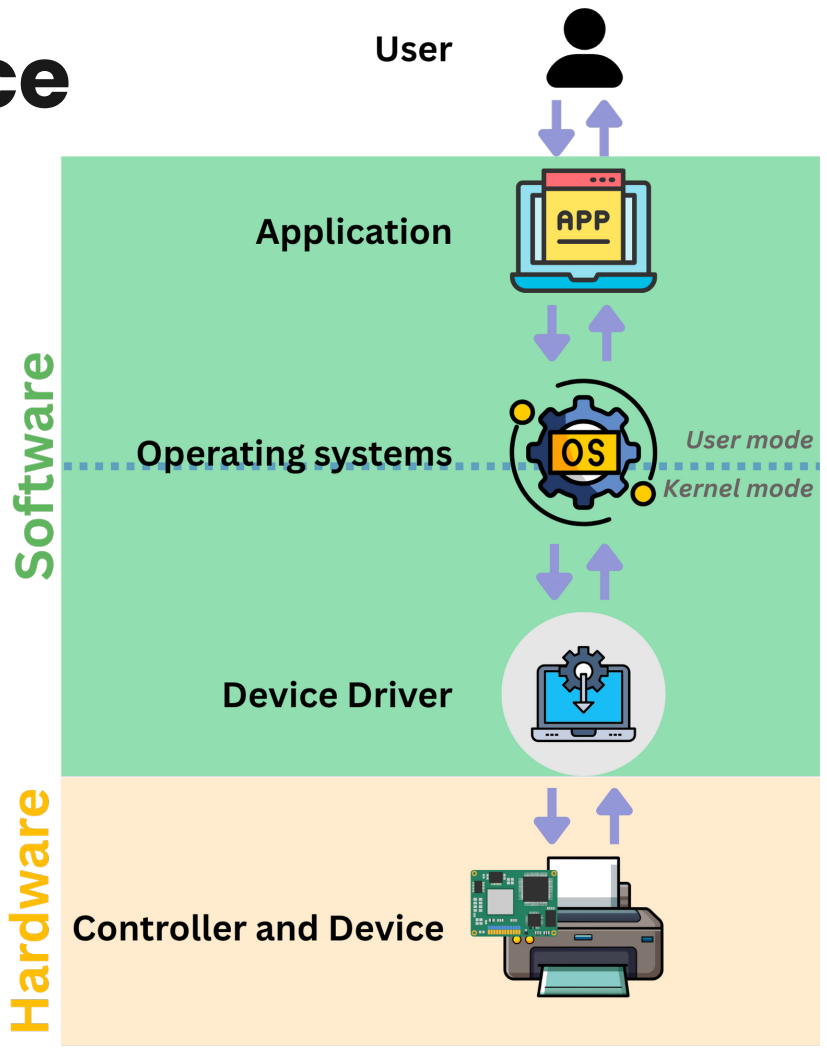
Type	Description	Example
Synchronous I/O	The CPU waits for the I/O operation to complete before continuing.	`fread()` in standard C
Asynchronous I/O	The CPU continues executing other tasks while I/O happens in the background; the process is notified when done.	`aio_read()` or callback-based I/O

Application I/O Interface

Device Driver

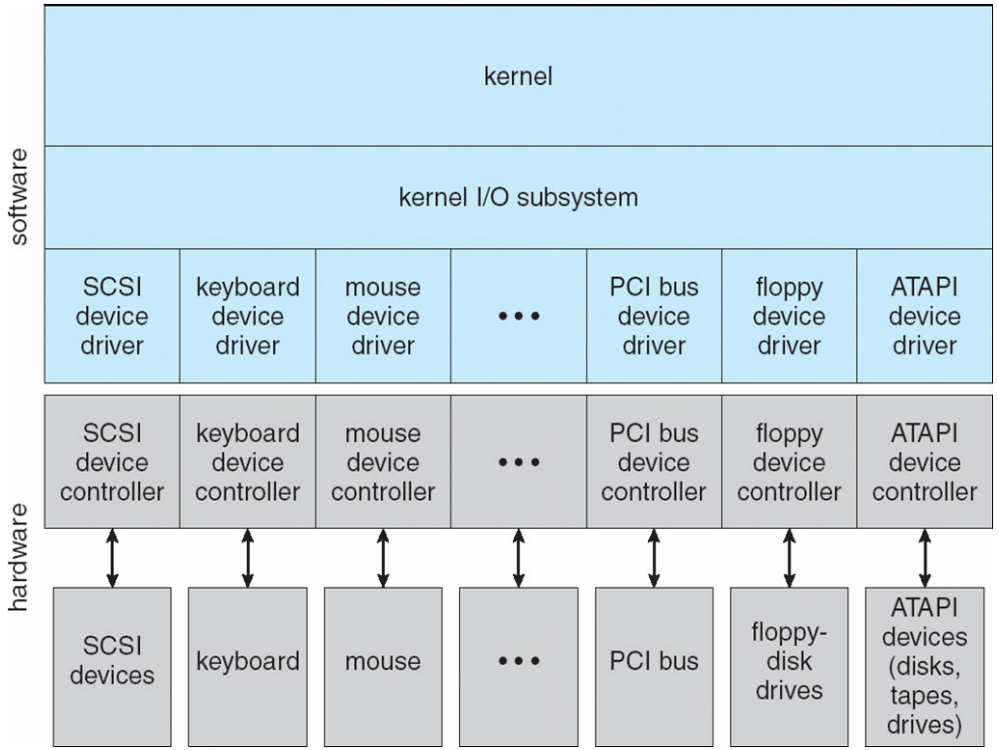
A device driver is a software module that:

- Acts as an interpreter between the OS and hardware.
- Converts generic I/O requests from the OS into specific commands for the device controller.
- Manages device initialization, data transfer, interrupt handling, and error checking.



Kernel I/O Subsystem

- The Kernel I/O Subsystem is the core part of the operating system that manages all I/O operations between user applications and hardware devices.



It handles tasks like:

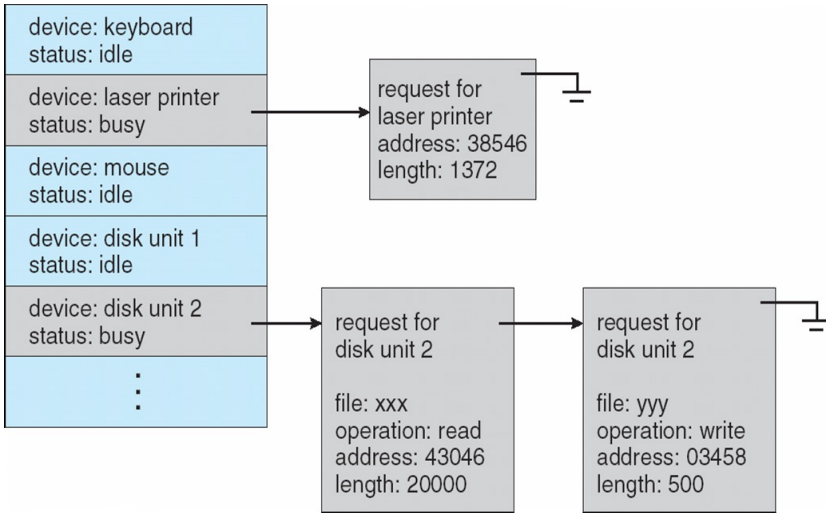
- Scheduling I/O requests
- Buffering and caching data
- Managing spooling and device queues
- Performing error handling and device protection

Major Components of the Kernel I/O Subsystem

Component	Description
I/O Scheduling	Determines the order in which I/O requests are processed to improve performance (e.g., minimize disk seek time).
Buffering	Uses temporary memory areas (buffers) to smooth speed differences between devices and processes.
Caching	Stores recently accessed data in fast memory for quicker future access.
Spooling	Queues data for devices that can handle only one operation at a time (e.g., printers).
Device Reservation	Prevents conflicts by allowing only one process to access a device at a time.
Error Handling	Detects and recovers from I/O failures or hardware errors.
Protection	Controls which users/processes can access which devices.

I/O Scheduling

- Decides the order in which I/O requests are handled.
- Aims to improve throughput and reduce waiting time.
- The kernel manages device-status table (an entry for each I/O device, device's type, address, and state (not functioning, idle, or busy))

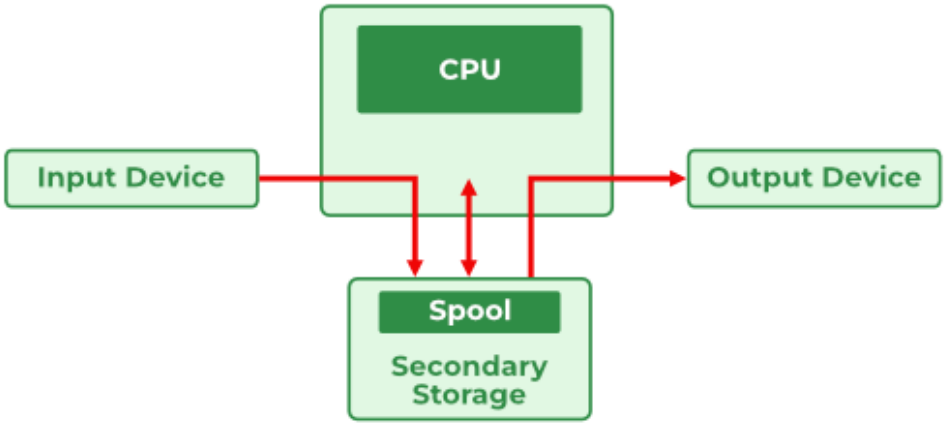


Buffering and Caching

Aspect	Buffering	Caching
Definition	Temporary storage of data while it is being transferred between two devices or processes.	Storing copies of frequently accessed data in faster memory for quick access.
Purpose	To handle **speed mismatch** between producer and consumer, and ensure smooth data transfer.	To **reduce access time** for data that is likely to be used again.
When Used	During **data transfer** between devices or between process and device.	During **data retrieval** operations (e.g., reading from disk or network).
Data Lifetime	Short-term — data is discarded after use.	Longer-term — data stays until cache replacement occurs.
Location	In **main memory (RAM)** or kernel space as a temporary area.	In **main memory or CPU cache**, managed by OS or hardware.
Example	Keyboard input stored temporarily before application reads it.	Frequently accessed disk blocks stored in file system cache.
Main Benefit	Smooths I/O flow and prevents data loss.	Improves performance by avoiding repeated slow accesses.
Managed By	Operating system's **I/O subsystem**.	Operating system or **hardware caching mechanism**.
Data Reuse	Typically **not reused**.	Data is **reused** if accessed again.

Spooling

Overlap the I/O Operation of One Job with the Execution of the Another Job



Device Reservation

Error Handling

- Detects hardware or transmission errors.
- Retries operations or reports errors to applications.
- Logs errors for system administrators.

Example: Disk read error → OS retries → if persistent, reports “Input/output error”.

Device Protection

- Ensures only authorized processes can perform I/O operations.
- Uses file permissions, access control lists (ACLs), and kernel privileges.