# I/O Systems

Managing I/O is a significant challenge in OS design due to the vast variety of devices and their differing speeds. The primary role of the OS in I/O is to manage and control I/O operations and devices. The OS aims to make the programming task easier by providing certain services to programs and users.

## *Controllers*

Electronic component capable of operating a port, a bus, or a device. Some devices (disk drives) include a built in controller on their circuit board

**Host Controller (HBAs)** - Computer at the end of the bus

**Device Controller** - Built into the devices

Controller manages local buffer storage and registers and is responsible for transferring data between the peripheral devices and its buffer.

## *Polling*

- Host (device driver usually) repeatedly checks the status of a device controller by reading a status bit in its register
- Host actively executes instructions to attend to device. Host might repeatedly reads a 'busy' bit until it clears, indicating device is ready
- **Pros** - Efficient if device is fast, as busy waiting time is minimized
- **Cons** - Can be inefficient if the de ice is slow, as CPU wastes cycles repeatedly checking a status that rarely changes
- OS does not have to busy-wait but must attend to device regularly
- **Compared to interrupts** - Efficient in terms of CPU cycles per check (ie three instructions cycles). Inefficient if the checks frequently find the device not ready

**File-Organization Module** - Understands files and their logical blocks, mapping logical block #s to physical blocks and managing free space

**Logical File System** - Manages file system structure, handling metadata, directory structure, and protection

**Caching**

Storing copies of frequently accessed data in faster mem to reduce access time to slower devices (like disks)

**Cache Hit** - Requested data is found in cache → fast access

**Cache Miss** - Data is not in cache → fetch from slower device and possibly update cache

**Policies**

- ***Replacement Policy*** - When cache is full, decides which block to evict
- ***Write Policy***
  - ***Write Through*** - Write to cache and disks simultaneously (slower but safer)
  - ***Write Back*** - Write only to cache at first, write to disk later (faster but risks data loss on failure)

**Differences from buffering**

- ***Buffering*** manages data transfer in transit
- ***Caching*** stores copies to avoid re-fetching data

## *Interrupts*

- Hardware can signal CPU at any time by asserting a signal on interrupt-request line, usually thru system bus. This signals event requiring CPU attention
- When interrupt occurs, CPU stops current activity, saves state, and transfers execution to a fixed mem location containing starting address of appropriate **interrupt service routine (ISR)**
- Interrupt vector is often used, providing a table of pointers to ISRs indexed by a unique number associated with request. Allows for efficient dispatching to correct handler without polling all devices
- Interrupts are used for various purposes. (1) Device service requests (2) Hardware faults (3) Exceptions (page faults, divide by zero) (4) System calls (implemented as software interrupts or traps)
- **Multilevel Interrupts** & interrupt priority levels - Allows OS to prioritize urgent work and handle concurrent interrupts
  - **Maskable** - Interrupt that can be turned off by CPU for critical code sections
  - **Non-maskable** - Interrupt cannot be deferred
- Often split into a **first level interrupt handler (FLIH)** for quick tasks (context switch, state saving, queuing), and second level interrupt for more involved processing, which is scheduled separately.
- **Pros** - Compared to polling, interrupts avoid wasting CPU cycles in busy waiting when devices are slow
- **Cons** - Interrupts handling introduces overhead from state saving/restoring and executing handler
- **Def - Interrupt Latency** - Time from interrupt arrival to start of ISR
  - Minimizing is critical for real time systems

**Def - Bus** - The bus connections the CPU memory subsystem to devices

- If devices share a set of wires, the connection is referred to as a bus. Examples include the PCIe bus connecting to fast devices, and expansion buses for slower devices

**Def - Daisy Chain** - Arrangement of connected devices that usually operates as a bus

**Def - Port** - Devices communicate with the computer system by sending signals over a cable or wirelessly through a connection point called a port

CPU interacts with the device controller by reading and writing patterns in the controllers registers. Two main methods for the OS to know when a device needs servicing : **Polling, interrupts**

***Blocking I/O*** - When an application performs a blocking I/O call (ie `read()`, `write()`), calling thread's execution is suspended.

- Thread is moved from OSs run queue to wait queue associated with device/event it is waiting. Execution of thread only resumes after I/O operation completed→ it is then moved back to ready queue. Considered **synchronous** from process's perspective, execution is synchronized with completion of I/O

***Nonblocking I/O*** - Nonblocking system call does not suspend thread for an extended period. Instead, it returns quickly, often with a return value indication how much data was transferred. Application thread can then continue executing other tasks while I/O proceeds in background. Application can later check for completion status or availability of more data, like using `select()`. Useful for applications needing to remain responsive during I/O

***Synchronous I/O*** - Process initiating I/O request waits until operation is fully completed before it can continue its execution

***Asynchronous I/O*** - Process issues I/O request and immediately continues its execution without waiting for operation to finish

## DMA (Direct Memory Access)

For large data transfers (e.g. disks), using CPU one byte/word at a time is inefficient. DMA offloads this work to a specialized **DMA controller**.

- CPU writes a **DMA command block** to memory → source/destination addresses, size, optional list of non-contiguous addresses (**scatter-gather**).
  - **Scatter:** Reads large block → scatters across multiple memory buffers.
  - **Gather:** Gathers data from multiple buffers → writes as one output.
- CPU passes command block address to DMA controller, which then takes over the bus and performs transfer **without CPU intervention**.
- **Handshaking** between DMA and device controller coordinates transfer.
- Only **one interrupt** generated per block transfer (vs per byte/word), reducing interrupt overhead.
- CPU can execute other programs while DMA runs.
  - DMA seizing bus can cause **cycle stealing** (temporarily prevents CPU mem access), but overall improves system performance.
- Some systems support **Direct Virtual Memory Access (DVMA):** DMA uses virtual addresses, translated during transfer.
  - **Pros**: avoids extra copy through kernel buffers.
  - **Cons**: complicates controller design, requires hardware support.

## Buffering

- When system temporarily stores data in mem while it is being moved between two paces. Needed because sender and receiver of data often operate at different speeds, or expect data in different sizes, or we want to ensure data correctness
- ***Producer*** - part of system generating data
- ***Consumer*** - Part of system using that data
- **Normal Buffering**
  - One buffer is used, and both producer and consumer read/write it alternately
  - Problem - If they access it at same tie, conflicts or delays can happen
- **Double Buffering**
  - Two buffers are used
    - While producer fills buffer A, consumer processes buffer B
    - Then they switch roles, producer fills B, consumer processes A
  - Allows both to work independently - decoupled - which improves performance and avoid waiting

**At lowest level** is hardware itself

**I/O control layer** consists of device drivers and interrupt handlers that manage data transfers between memory and devices

**Device driver** layer hides details of different hardware and provide a common way for OS to use them. Writing new drivers lets new hardware work with existing OSs

## Kernel I/O Subsystem Services

Kernel provides various services building on device drivers

**I/O Scheduling**

- Ordering pending I/O requests for a device to improve overall performance, fairness, or average waiting time
- **Why it matters?**
  - Disk operations are much slower that mem and CPU
  - Access time depends heavily on order of operations - scheduling can reduce seek time, rotational latency, and overall response time
- **Goal of scheduling**
  - Minimize average seek time, maximize throughput, fairness across processes (prevent starvation), prioritize time-sensitive requests
- **Common Disk Scheduling Algorithms**
  1. **FCFS (First Come, First Serve)** - Simple, fair, Poor performance under heavy load (no seek time optimization)
  2. **SSTF (Shortest Seek Time First)** - Selects request closest to current head position, Faster than FCFS, but can lead to starvation of far requests
  3. **SCAN (Elevator Algorithm)**, Disk arm moves in one direction servicing all requests, then reverses, Prevents starvation, balances performance and fairness
  4. **C-SCAN (Circular SCAN)**, Like SCAN, but only services in one direction, jumps back to beginning when done, Provides more uniform wait times
  5. **LOOK/C-LOOK**, Variants of SCAN/C-SCAN that only go as far as furthest request in that direction

## Error Handling

Refers to detection, reporting, and recovery from errors that occur during device operations

**Types of I/O Errors**

- ***Hardware Errors*** - Bad disk sectors, corrupted files
- ***Media Errors*** - Bad disk sectors, corrupted files
- ***Application Errors*** - invalid system calls, bad parameters
- ***Transmission Errors*** - dropped packets, checksum mismatches

**OS Responsibilities**

- ***Detection*** - via hardware signals, interrupts, or status bits
- ***Reporting*** - to user or application
- ***Recovery***
  - Retry operation
  - Fail gracefully and notify user or application
  - Log error for diagnostics
  - Mark bad sectors or isolate faulty components

## Application I/O Interface

OS provides standard functions for apps to access I/O devices, abstracting hardware details. Devices are grouped into types:

- **Block Devices**

  Transfer data in fixed-size blocks (e.g. disks, SSDs).

  Support read(), write(), seek().

  Accessed via file system or as raw/memory-mapped devices.
- **Character Devices**

  Transfer data byte-by-byte (e.g. keyboards, mice, serial ports).

  Accessed serially, one byte at a time.
- **Memory-Mapped Files**

  Map disk blocks to virtual mem → allows file access via normal mem loads/stores.

  Leverages demand paging.
- **Network Sockets**

  Interface for network communication.

  Support connect, listen, send, receive.

  **Packets** = small data units sent/reassembled in network.
- **Clocks & Timers**

  Provide current/elapsed time, allow timer setting.

  Often use programmable interval timers + system calls (ioctl).

## Kernel Data Structures

Track state/activity of I/O components & operations.

**Types:**

- **Device-Status Tables**

  Track device state (idle, busy, error), interrupt status, buffer pointers.
- **Open File Tables**

  Track files open by all processes.

  Store file descriptor, file pointer, access mode.
- **Per-Process File Tables**

  Map process-specific file descriptors → global open file table.
- **I/O Request Queues**

  Hold pending I/O requests (used by schedulers).
- **Network Connection Tables**

  Track socket states, endpoints, buffers for active connections.

**Why It Matters:**

Critical for managing I/O resources, ensuring correct operation, concurrency, and system stability. Efficient structure → better performance & security.

**Device Reservation**

Device reservation refers to mechanisms that allow a process to request exclusive access to a physical deice for a period of time

**Why it matters?**

- Some devices cannot be shared safely or meaningfully
- Without reservation, two processes might interfere with each other's use, leading to corrupted output or undefined behavior

**How it works**

- OS provides system calls that grant access to a device
- Once reserved, other processes are either blocked or denied access until current process releases it

## Error Handling

Detection, reporting, and recovery from I/O errors.

**Types of Errors:**

- **Hardware:** bad disk sectors, device faults.
- **Media:** corrupted files, bad sectors.
- **Application:** invalid sys calls, bad params.
- **Transmission:** dropped packets, checksum errors.

**OS Responsibilities:**

- **Detection:** via hardware signals, status bits, interrupts.
- **Reporting:** notify user/app.
- **Recovery:**
  - Retry operation.
  - Fail gracefully, notify app.
  - Log error.
  - Mark bad sectors / isolate faulty hardware.

## Performance

**Why I/O affects performance**

- CPU load from divers and kernel code handling I/O
- Context Switches - Caused by interrupts and blocking I/O → expensive in terms of CPU cycles
- Data copying - Moving data between user space and kernel buffers adds overhead