Input/Output Networks

Input/Output systems are the backbone of computer-device communication. They manage data flow between the CPU, memory, and external devices, enabling user interaction and efficient data transfer. These systems consist of hardware components like controllers and buses, as well as software components like device drivers and I/O schedulers. I/O systems handle various device types, including input devices like keyboards, output devices like monitors, and storage devices like hard drives. They abstract complex operations, allowing developers to focus on high-level functionality while ensuring reliable data transfer and minimizing performance impact on the overall system.

What Are I/O Systems?

- Handle communication between the computer and external devices (keyboards, displays, storage devices)
- Enable data transfer to and from the CPU and memory
- Consist of hardware components (I/O devices, controllers, buses) and software components (device drivers, I/O schedulers)
- Facilitate user interaction with the computer system through input devices and output devices
- Manage the flow of data between the computer and peripheral devices
- Provide an interface for applications to access and manipulate data on storage devices
- Ensure efficient and reliable data transfer while minimizing the impact on system performance
- Abstract the complexities of I/O operations, allowing developers to focus on higher-level functionality

Types of I/O Devices

- Input devices
    - Keyboards and mice for user input
    - Scanners and cameras for capturing images and documents

    - Microphones for audio input
- Output devices
    - Monitors and displays for visual output
    - Printers for producing hard copies of documents
    - Speakers and headphones for audio output
- Storage devices
    - Hard disk drives (HDDs) for high-capacity, non-volatile storage
    - Solid-state drives (SSDs) for faster, more reliable storage
    - Optical drives (CD/DVD) for portable storage and media playback
- Network devices
    - Network interface cards (NICs) for connecting to local area networks (LANs)
    - Modems for establishing internet connectivity
- Specialized devices
    - Graphics processing units (GPUs) for accelerated video rendering and gaming
    - Touchscreens for intuitive user interaction on mobile devices and kiosks

I/O Hardware Basics

- I/O devices connect to the computer system through ports and cables
- Ports provide a physical interface for connecting devices (USB, HDMI, Ethernet)
- Cables transmit data and power between the device and the computer (USB cables, SATA cables)
- I/O controllers manage the communication between the device and the system
    - Implement protocols for data transfer and device control
    - Handle data buffering, error detection, and correction
- Buses facilitate data transfer between components
    - System bus connects the CPU, memory, and I/O controllers
    - Peripheral buses (USB, PCIe) connect I/O devices to the system
- Interrupts signal the CPU when I/O operations require attention
    - Devices raise interrupts to indicate the completion of a task or to request service

- The CPU suspends its current task and invokes the appropriate interrupt handler

I/O Software Layers

- Application layer
  - Provides high-level APIs and libraries for interacting with I/O devices
  - Enables developers to read from and write to files, communicate over networks, and capture user input
- Operating system layer
  - Manages I/O requests from applications and coordinates access to shared resources
  - Implements file systems, device drivers, and I/O schedulers
  - Provides system calls for applications to perform I/O operations
- Device driver layer
  - Communicates directly with I/O devices and controllers
  - Translates generic I/O requests into device-specific commands
  - Handles device initialization, configuration, and error handling
- Firmware layer
  - Low-level software embedded in I/O devices
  - Implements basic functionality and communication protocols
  - Provides an interface for device drivers to interact with the hardware

Device Drivers and Controllers

- Device drivers are software components that enable communication between the operating system and I/O devices
- Provide a standardized interface for the OS to interact with diverse hardware devices
- Encapsulate device-specific details and present a uniform API to higher-level software
- Responsible for initializing and configuring the device
  - Set up device registers, buffers, and interrupt handlers
  - Allocate necessary resources (memory, DMA channels)

- Handle I/O requests from applications and the operating system
    - Translate generic I/O commands into device-specific instructions
    - Manage data transfer between the device and system memory
- Implement error handling and recovery mechanisms
    - Detect and report device errors (timeouts, data corruption)
    - Attempt to recover from errors or notify the operating system for further action
- Optimize device performance through buffering, caching, and scheduling techniques

I/O Scheduling Algorithms

- I/O scheduling algorithms optimize the order in which I/O requests are serviced to improve system performance
- First-Come, First-Served (FCFS)
    - Processes I/O requests in the order they arrive
    - Simple to implement but can lead to long wait times and poor disk utilization
- Shortest Seek Time First (SSTF)
    - Selects the I/O request with the shortest seek time from the current head position
    - Minimizes disk head movement but may starve requests far from the current position
- SCAN (Elevator)
    - Moves the disk head in one direction, servicing requests along the way, then reverses direction when reaching the end
    - Provides a fair balance between response time and disk utilization
- C-SCAN (Circular SCAN)
    - Similar to SCAN, but moves the disk head in only one direction, then returns to the beginning when reaching the end
    - Ensures a more uniform wait time distribution compared to SCAN
- LOOK and C-LOOK

    - Variants of SCAN and C-SCAN that reverse direction based on the presence of pending requests rather than reaching the end of the disk
    - Avoids unnecessary disk head movement when no requests exist in the current direction

Buffering and Caching in I/O

- Buffering and caching techniques are used to improve I/O performance by reducing the frequency and latency of disk accesses
- Buffering
    - Temporary storage area in memory used to hold data being transferred between devices or between a device and an application
    - Allows I/O operations to be performed asynchronously, enabling the CPU to continue executing while data transfer occurs in the background
    - Smooths out the speed differences between the CPU, memory, and I/O devices
- Caching
    - Stores frequently accessed data in a high-speed memory area (cache) to reduce the need for slower disk accesses
    - Operates on the principle of locality, assuming that recently accessed data is likely to be accessed again in the near future
    - Read caching
        - Retrieves data from the disk and stores a copy in the cache
        - Subsequent reads for the same data can be served directly from the cache, avoiding disk access
    - Write caching
        - Buffers write operations in the cache and periodically flushes the changes to the disk
        - Improves write performance by allowing the application to proceed without waiting for the disk write to complete
- Caching policies

    - Determine which data to cache and when to evict data from the cache
    - Least Recently Used (LRU) evicts the cache entry that has been accessed least recently
    - Least Frequently Used (LFU) evicts the cache entry that has been accessed least frequently

I/O Performance and Optimization

- I/O performance is critical for overall system performance, as I/O operations are often the bottleneck in many applications
- Factors affecting I/O performance
    - Device characteristics (seek time, rotational latency, data transfer rate)
    - I/O request patterns (sequential vs. random access)
    - File system overhead (metadata management, fragmentation)
    - I/O software layers (device drivers, I/O schedulers)
- Techniques for optimizing I/O performance
    - Disk partitioning and file system selection
        - Choose appropriate partition sizes and file systems based on workload characteristics
        - Use journaling file systems (ext4, NTFS) for improved reliability and recovery
    - I/O request merging and reordering
        - Combine adjacent I/O requests into larger, more efficient operations
        - Reorder requests to minimize disk head movement and seek times
    - Asynchronous I/O and non-blocking operations
        - Perform I/O operations asynchronously to allow the application to continue executing while I/O is in progress
        - Use non-blocking I/O APIs to avoid blocking the application when I/O resources are unavailable
    - Caching and buffering

        - Implement effective caching strategies to reduce disk accesses
        - Tune cache sizes and policies based on application requirements and available memory
    - I/O load balancing and parallelization
        - Distribute I/O requests across multiple devices or storage nodes to improve throughput
        - Leverage parallel I/O techniques (striping, RAID) to increase I/O bandwidth
- Monitoring and profiling I/O performance
    - Use system monitoring tools (iostat, iotop) to identify I/O bottlenecks and resource contention
    - Profile application I/O behavior to optimize data access patterns and minimize I/O overhead
    - Analyze I/O traces and logs to identify opportunities for performance tuning and optimization

One of the important jobs of an Operating System is to manage various I/O devices including mouse, keyboards, touch pad, disk drives, display adapters, USB devices, Bit-mapped screen, LED, Analog-to-digital converter, On/off switch, network connections, audio I/O, printers etc.

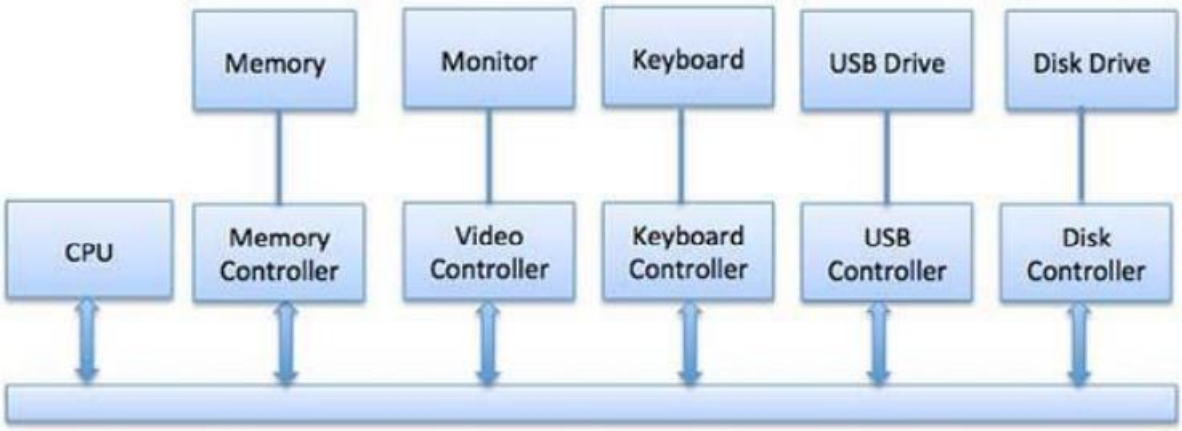
An I/O system is required to take an application I/O request and send it to the physical device, then take whatever response comes back from the device and send it to the application. I/O devices can be divided into two categories −

- Block devices − A block device is one with which the driver communicates by sending entire blocks of data. For example, Hard disks, USB cameras, Disk-On-Key etc.
- Character devices − A character device is one with which the driver communicates by sending and receiving single characters (bytes, octets). For example, serial ports, parallel ports, sounds cards etc

Device Controllers

Device drivers are software modules that can be plugged into an OS to handle a particular device. Operating System takes help from device drivers to handle all I/O devices. The Device Controller works like an interface between a device and a device driver. I/O units (Keyboard, mouse, printer, etc.) typically consist of a mechanical component and an electronic component where electronic component is called the device controller.

There is always a device controller and a device driver for each device to communicate with the Operating Systems. A device controller may be able to handle multiple devices. As an interface its main task is to convert serial bit stream to block of bytes, perform error correction as necessary. Any device connected to the computer is connected by a plug and socket, and the socket is connected to a device controller. Following is a model for connecting the CPU, memory, controllers, and I/O devices where CPU and device controllers all use a common bus for communication.



Synchronous vs asynchronous I/O

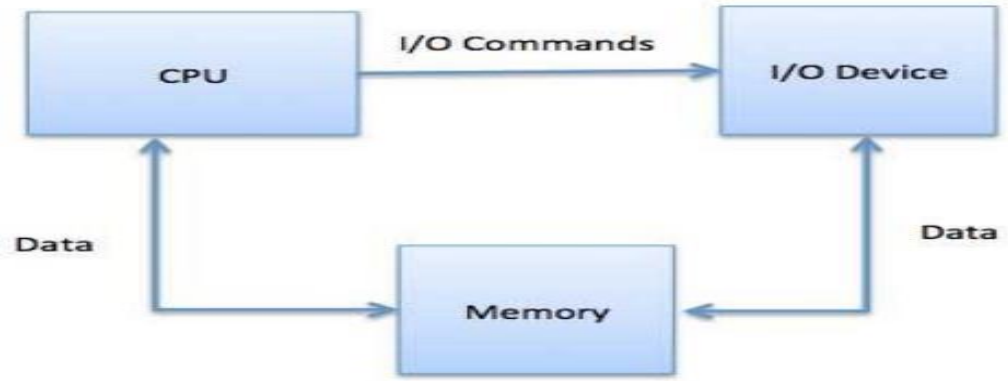
- Synchronous I/O − In this scheme CPU execution waits while I/O proceeds
- Asynchronous I/O − I/O proceeds concurrently with CPU execution Communication to I/O Devices The CPU must have a way to pass information to and from an I/O device. There are three approaches available to communicate with the CPU and Device.
- Special Instruction I/O
- Memory-mapped I/O
- Direct memory access (DMA)

Special Instruction I/O

This uses CPU instructions that are specifically made for controlling I/O devices. These instructions typically allow data to be sent to an I/O device or read from an I/O device.

Memory-mapped I/O

When using memory-mapped I/O, the same address space is shared by memory and I/O devices. The device is connected directly to certain main memory locations so that I/O device can transfer block of data to/from memory without going through CPU.



While using memory mapped IO, OS allocates buffer in memory and informs I/O device to use that buffer to send data to the CPU. I/O device operates asynchronously with CPU, interrupts CPU when finished.

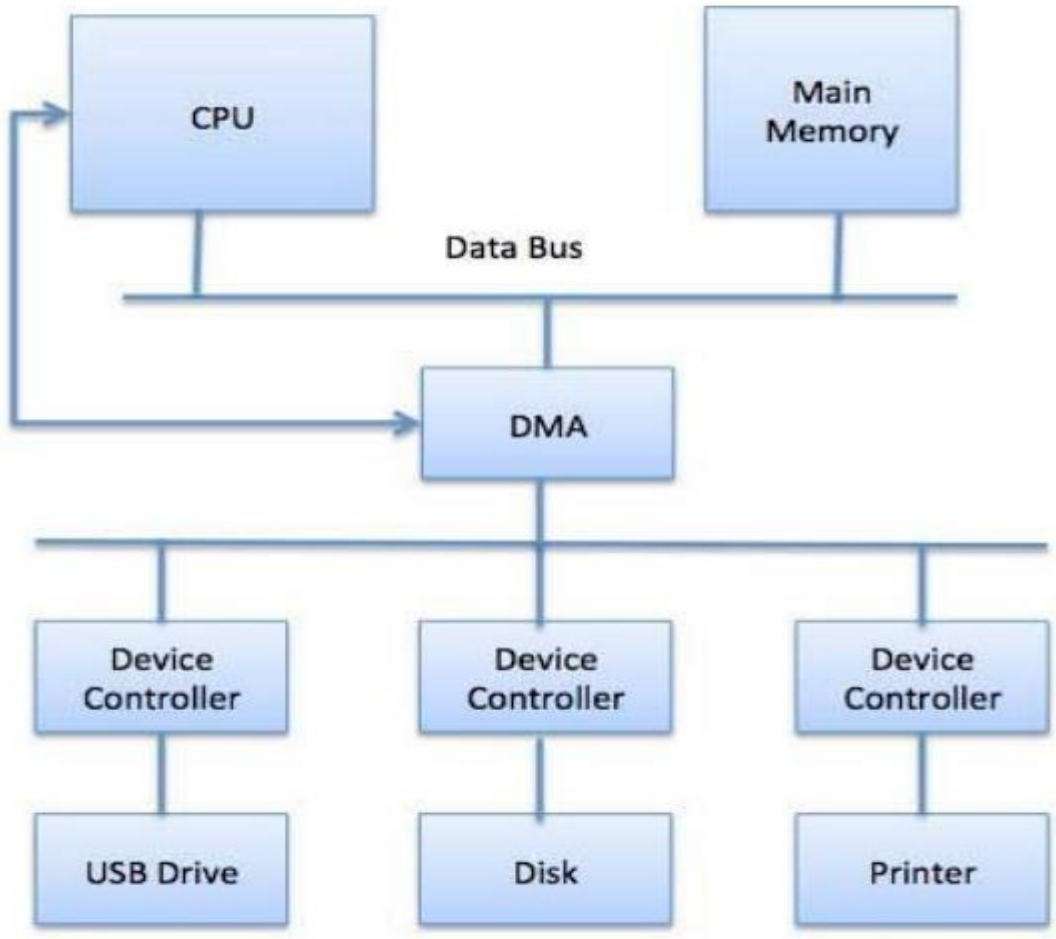
The advantage to this method is that every instruction which can access memory can be used to manipulate an I/O device. Memory mapped IO is used for most high-speed I/O devices like disks, communication interfaces.

Direct Memory Access (DMA)

Slow devices like keyboards will generate an interrupt to the main CPU after each byte is transferred. If a fast device such as a disk generated an interrupt for each byte, the operating system would spend most of its time handling these interrupts. So a typical computer uses direct memory access (DMA) hardware to reduce this overhead.

Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement. DMA module itself controls exchange of data between main memory and the I/O device. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

Direct Memory Access needs a special hardware called DMA controller (DMAC) that manages the data transfers and arbitrates access to the system bus. The controllers are programmed with source and destination pointers (where to read/write the data), counters to track the number of transferred bytes, and settings, which includes I/O and memory types, interrupts and states for the CPU cycles.



The operating system uses the DMA hardware as follows −

Step Description

1 Device driver is instructed to transfer disk data to a buffer address X.

2 Device driver then instruct disk controller to transfer data to buffer.

3 Disk controller starts DMA transfer.

4 Disk controller sends each byte to DMA controller.

5 DMA controller transfers bytes to buffer, increases the memory address, decreases the counter C until C becomes zero.

6 When C becomes zero, DMA interrupts CPU to signal transfer completion.

Polling vs Interrupts I/O

A computer must have a way of detecting the arrival of any type of input. There are two ways that this can happen, known as polling and interrupts. Both of these techniques allow the processor to deal with events that can happen at any time and that are not related to the process it is currently running.

Polling I/O

Polling is the simplest way for an I/O device to communicate with the processor. The process of periodically checking status of the device to see if it is time for the next I/O operation, is called polling. The I/O device simply puts the information in a Status register, and the processor must come and get the information.

Most of the time, devices will not require attention and when one does it will have to wait until it is next interrogated by the polling program. This is an inefficient method and much of the processors time is wasted on unnecessary polls.

Compare this method to a teacher continually asking every student in a class, one after another, if they need help. Obviously the more efficient method would be for a student to inform the teacher whenever they require assistance.

Interrupts I/O

An alternative scheme for dealing with I/O is the interrupt-driven method. An interrupt is a signal to the microprocessor from a device that requires attention.

A device controller puts an interrupt signal on the bus when it needs CPU‟s attention when CPU receives an interrupt, It saves its current state and invokes the appropriate interrupt handler using the interrupt vector (addresses of OS routines to handle various events). When the interrupting device has been dealt with, the CPU continues with its original task as if it had never been interrupted.