# I/O Subsystems

Secure operating systems.

Lecturer: Ms. Suranjini Silva.

Sri Lanka Institute of Information Technology.

Date of Submission:  2023/04/09.

| IT numbers | Name | Individual Contribution |
|---|---|---|
| IT21826368 | Nanayakkara Y.D.T. D | I/O virtualization.<br>I/O Buffers and caches.<br>Error Handling.<br>Vulnerabilities and Security. |
| IT21822612 | Mendis H.R.M | I/O Scheduling.<br>Direct Memory Access (DMA).<br>Future Deployment. |
| IT21831904 | Weerasinghe K.M | Introduction.<br>Device Drivers.<br>Interrupt Handling. |
| IT21828348 | Dissanayake K.D.A.R. A | Hardware Components.<br>Input and Output Operation. |

## Abstract.

I/O Subsystems are the fundamental components of a system, which help in communicating between one device to its external environment. I/O devices, controllers and drivers are some of the main components of I/O subsystems. They will work together to swap data between the system and its peripherals. I/O devices are responsible for receiving input and providing output, and the controllers manage the flow of data transfer between the I/O devices and the main memory. I/O drivers serve as the intermediary between the Operating System and the hardware to ensure the data processing correctly. Devices like Storage devices, Input and Output devices, there are several types of I/O devices exists. Each device has its own nature and purpose to do. Because of this, System I/O subsystems are very important.

To all the hardware components and drivers which will used to control them I/O subsystems have several techniques and protocols to enhance the performance and algorithms for the effective performance of the systems, like interrupt driven I/O, Direct Memory access, Virtualization, and interrupt handling. Using them the I/O operation and management on I/O operations will ensure to work properly.

Despite its critical importance, the I/O subsystems also have their challenges. Common issues include vulnerabilities, security risks, hardware failures, bottlenecks, compatibility issues, and scalability problems. To overcome these challenges, developers have proposed various solutions, such as improving hardware design and developing new protocols, so the new improved devices come out always.

# 1 Introduction.

I/O subsystem is the I/O devices and the parts of the kernel that manages the I/O [1].The I/O subsystems are responsible for A computer system's ability to interface with external devices, including as input devices like keyboards and mice, output devices like monitors and printers, and storage devices like hard drives and flash drives. It is possible for the system to send and receive data to and from external parties thanks to the I/O subsystem, which handles the flow of data among the computer's main memory with the external devices. The I/O subsystem consists of several components, a memory-management component that includes buffering, caching, and spooling, a general device-driver interface, drivers for specific hardware devices [1].

The I/O subsystem consists of a variety of hardware and software elements that cooperate to support input/output tasks. Device driver is an operating system component that provides uniform access to various devices and manages I/O to those devices [1]. The device driver performs tasks like controlling the device's power consumption, creating input/output queues, and managing interrupts that announce the arrival of fresh data from the device. It also interprets instructions from the operating system into commands that the hardware is capable of understanding.

The *interrupt handler*, which is responsible for controlling interrupts generated from external devices, is another essential component of the I/O subsystem. The CPU receives signals known as interrupts when an external device, such as an user pressing a key on a keyboard, requires attention. The interrupt handler reacts to these signals and starts the necessary input/output processes, allowing swift and efficient communication between the internal system and the external device. interrupt handlers and device drivers are used in the construction of efficient I/O subsystems [1].

Another essential part of the I/O subsystem is the *device controller*. device controller is the I/O managing processor within a device [1]. These are specialized hardware components that oversee controlling data transfer between the computer and external devices. Device controllers are independent components or built into the motherboard and connected to the computer using a bus or another interface. These controllers regulate operations including handling low-level input/output process elements and buffering data to and from the device. These also control error events.

The I/O subsystem also consists of several *buses and protocols* that allow the computer and external devices to communicate. Buses is a communication system within a computer, a bus connects various components, such as the CPU and I/O devices, allowing them to transfer data and commands [1]. Buses serve as a means of communication between different computer parts, including the CPU, memory, and peripheral devices.

In conclusion, the I/O subsystem of a computer system plays a critical part that enables it to connect with the world. For users to input and output data, store data, and connect with other devices, the I/O subsystem, which oversees the data flow among the computer and external devices, is a crucial part of modern computing.

# 2 Hardware components.

## 2.1 Description of hardware components in the I/O subsystem.

The communication between the computer system and the outer world, including devices for keyboards, mouse, displays, printers, and more, is handled by the input/output (I/O) subsystem. The I/O subsystem's hardware components might vary depending on the system, but normally include.

## 2.2 Peripheral Devices

A peripheral device is any external device that connects to and interacts with the system to either input or output information. These components are also known as external peripherals, integrated peripherals, auxiliary components, or I/O devices.

## 2.3 What Defines a Peripheral Device?

Generally, the word peripheral is used to refer to a device external to the computer, such as a scanner, but devices that are physically located inside the computer are technically peripherals.

Peripherals add functionality to the computer but are not part of the "main" set of components such as the CPU, motherboard, and power supply. However, even though they are often not related to the main function of the computer, which does not mean that they are not considered necessary components.

For example, a desktop-style computer monitor does not technically support computing and does not require the computer to run and execute programs, but it is required to actually use the computer.

Another way to think about peripheral devices is that they do not act as standalone devices. The only way they work is when they are connected to and controlled by the computer.

## 2.4 Type of Peripheral Device

- Internal Peripheral Devices

Common internal peripheral devices you'll find in a computer include an optical disk drive, a video card, and a hard drive.

In those examples, the disk drive is one instance of a device that is an input and output device. The computer can use it not only to read information stored on the disc (e.g., software, music, movies) but also to export data from the computer to the disc (such as when burning a DVD)

- External Peripheral Devices

Anything that you can connect to the outside of a computer, that typically doesn't operate on its own, could be referred to as an external peripheral device. (Example-mouse, keyboard, pen tablet, external hard drive etc.)

## 2.5 Channels

The I/O Channel notion is an outgrowth of the DMA concept. It can execute I/O instructions on an I/O channel using a special-purpose CPU and has total control over I/O operation. The CPU does not carry out I/O commands. By commanding the I/O channel to run a program in memory, the CPU commences I/O transfer.

### *2.5.1 DMA channels*

DMA channel is an enables a device to transfer data without exposing the CPU to a work overload. Without DMA channels, the CPU copies each piece of data from the I/O device using a peripheral bus.

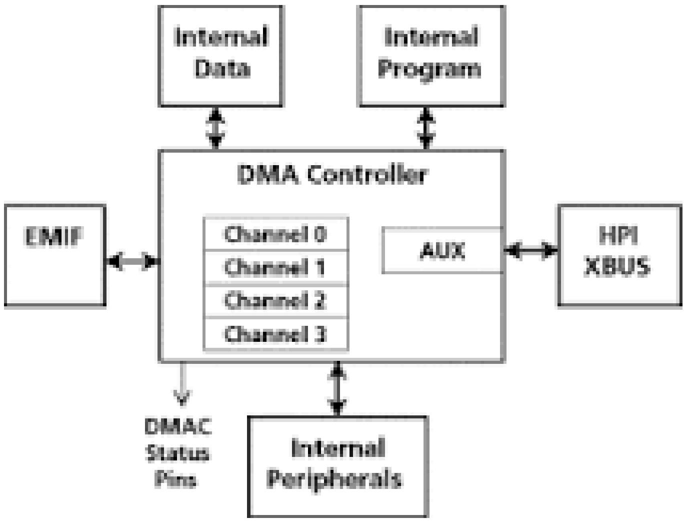

*Figure 1 DMA channel*

DMA channels are often used to transfer data to input/output device. A separate DMA controller is required to handle the transfer. The controller notifies the DSP processor that it is ready for transfer.

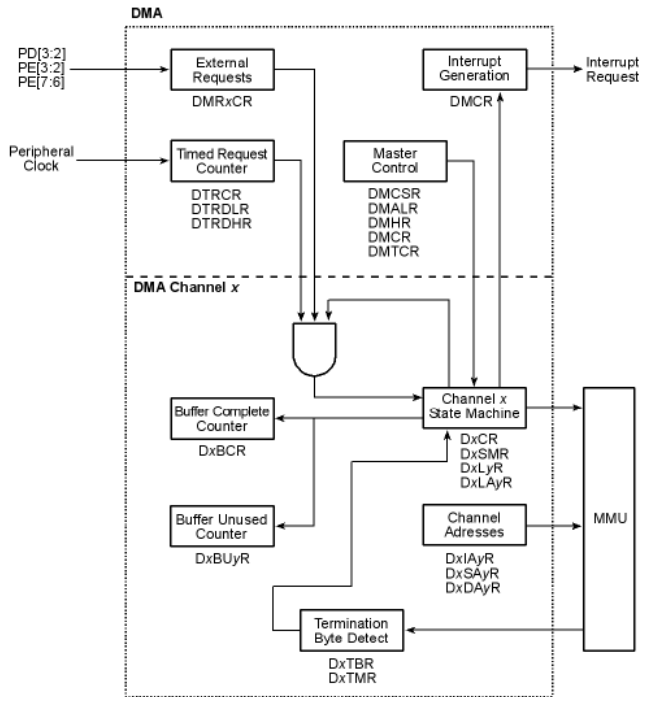

*Figure 2*

All eight channels are identical, and are capable of transferring data to or from memory, external I/O, or internal I/O. The priority between the channels can be either fixed or rotating, and the DMA use of the bus can be limited to guarantee interrupt latency or CPU throughput.

### 2.5.2 *Interrupt-driven channels*

An interrupt is a signal emitted by a device attached to a computer or by a program within the computer. It needs to stop the operating system (OS) and figure out what to do next. An interrupt temporarily stops or terminates a service or current process.

Almost all computer systems run on interrupts. This means that they follow the list of computer instructions in a program and execute the instructions until the end or until they sense an interrupt signal. If the latter event occurs, the computer will start executing the current program or another program. It should halt operations while deciding the next course of action. To do this and work on other programs, the OS uses interrupts in operations.

When the device processor handles interrupts, it notifies the sending device that the interrupt request (IRQ) signal has been recognized. Then the device stops sending IRQ signals.

An OS typically includes code called an interrupt handler to prioritize interrupts and save them in a queue if more than one is waiting to be handled. It also has a scheduler program that determines the next program that gets control.

When an interruption occurs, the associated service may not start immediately. The time interval between the time the interrupt occurs and the time the ISR execution begins is called the interrupt delay.

Interrupts allow the CPU to be notified when there is data to be transferred or when an operation has completed, allowing the CPU to perform other duties when I/O transfers do not require immediate attention.

### *2.5.3 Control channels*

Control channels are hardware or software components that manage the transfer of data between a computer's central processing unit (CPU) and peripheral devices, such as disks, printers, and network adapters.

Control channels are responsible for initiating and coordinating I/O operations, monitoring the progress of data transfers, and handling errors and interrupts. They also help to manage system resources, such as buffer space and memory allocation, to ensure efficient I/O performance.

In modern computer systems, control channels may be implemented in hardware as part of a peripheral interface controller (PIC) or in software as part of an operating system's device driver. Regardless of their implementation, control channels play a critical role in enabling efficient and reliable communication between a computer and its peripheral devices.

### *2.5.4 Data channels*

Data channels are the pathways that transfer data between a computer's central processing unit (CPU) and peripheral devices, such as disks, printers, and network adapters. These channels are responsible for the actual transfer of data and work closely with the I/O control channels to manage the entire I/O process.

Data channels can be implemented in many ways, depending on the type of peripheral device and the computer's architecture. For example, they may use parallel or serial communication protocols, and they may operate in half-duplex or full-duplex mode.

In a typical I/O operation, data channels transfer data between a peripheral device and the computer's main memory. The CPU initiates the transfer by issuing an I/O request to the control channel, which then sends the request to the appropriate data channel. The data channel transfers the data between the peripheral device and the memory, and then signals the control channel when the transfer is complete.

To improve I/O performance, data channels may incorporate various techniques, such as DMA (direct memory access) and caching. DMA allows data to be transferred directly between the peripheral device and memory, bypassing the CPU, while caching can reduce the number of actual data transfers by storing frequently accessed data in a cache memory.

Overall, data channels play a critical role in enabling efficient and reliable communication between a computer and its peripheral devices.

### 2.5.5 *Buses*

Buses are communication pathways or channels that connect the various hardware components of a computer system. They are used to transfer data and control signals between different devices, such as the processor, memory, and input/output devices.

Some commonly used buses in I/O include the system bus, which connects the processor to the main memory and other components, the Peripheral Component Interconnect (PCI) bus, which connects peripheral devices such as sound cards and network cards to the motherboard, and the Universal Serial Bus (USB) interface, which allows devices such as keyboards, mice, printers, and external hard drives to be connected to a computer. Other buses in I/O include the Small Computer System Interface (SCSI) and Serial Advanced Technology Attachment (SATA) interfaces, which are used to connect storage devices to a computer.

**System bus**

The system bus is a network of communication paths that connects the central processor unit (CPU) to other computer system components. It is in charge of enabling data, instruction, and control signal transfers between the CPU and other components like as memory, input/output devices, and other peripherals.

The data bus, address bus, and control bus are the three types of buses that make up the system bus I/O. The data bus transports information between the CPU and other components. The quantity of data that may be transported at once is determined by the size of the data bus. A 32-bit data bus, for example, may transport 32 bits of data at once.

The address bus is used to define the memory location or device address to or from which data is being transported. The greatest amount of memory that the CPU may access is determined by the size of the address bus. A 32-bit address bus, for example, may address up to 4 GB of memory.

Control bus signals govern the timing and sequencing of data transfers and other processes. Signals like as read/write signals, interrupt signals, clock signals, and reset signals are included.

**Expansion bus**

An expansion bus I/O is a type of input/output architecture that allows a computer system to connect and communicate with expansion cards or peripheral devices, such as sound cards, graphics cards, network adapters, and other add-on components.

Expansion bus I/O typically consists of a series of slots or connectors on the motherboard that can accept several types of expansion cards. These slots are connected to the CPU and other components via the system bus I/O, which provides the necessary data transfer and control signals.

There are several types of expansion buses available, including the Peripheral Component Interconnect (PCI), the Accelerated Graphics Port (AGP), the Peripheral Component Interconnect Express (PCIe), and the Universal Serial Bus (USB).

The PCI bus is a standard interface used for connecting peripheral devices to the computer. It can handle high-speed data transfer rates and supports both 32-bit and 64-bit data transfers.

The AGP bus is a specialized interface used for connecting graphics cards to the motherboard. It provides a high-bandwidth connection between the graphics card and the CPU, enabling high-quality graphics rendering and video playback.

The PCIe bus is a newer and faster interface used for connecting high-performance devices to the motherboard, such as graphics cards, network adapters, and solid-state drives. It provides faster data transfer rates than the PCI and AGP buses and supports hot swapping of devices.

The USB bus is a widely used interface for connecting peripheral devices to the computer, such as printers, scanners, keyboards, and mice. It supports plug-and-play functionality and provides a convenient and standardized way of connecting a wide variety of devices to the computer.

**I/O bus**

I/O bus is a type of computer bus that provides a pathway for data to be transferred between input/output devices and the central processing unit (CPU) of a computer system. I/O bus allows for the communication between the CPU and peripheral devices such as keyboards, mice, printers, scanners, and other external devices.

In a computer system, the I/O bus typically consists of a set of wires and connectors that allow for the transfer of control signals, data, and addresses between the CPU and input/output devices. The I/O bus can be implemented using different technologies such as parallel buses, serial buses, or a combination of both.

Parallel I/O buses transfer multiple bits of data simultaneously in parallel over a set of wires, while serial I/O buses transfer data one bit at a time over a single wire or pair of wires. Examples of parallel I/O buses include the ISA (Industry Standard Architecture), EISA (Extended Industry Standard Architecture), and VESA (Video Electronics Standards Association) buses. Serial I/O buses include USB (Universal Serial Bus), FireWire, SATA (Serial Advanced Technology Attachment), and Thunderbolt.

**Backplane busses**

A backplane bus is a type of data communication system that allows multiple electronic components or devices to communicate with each other by sharing a common communication path or bus. Backplane buses are commonly used in computer systems, telecommunications equipment, and industrial control systems.

The backplane bus serves as a backbone for the system, providing a high-speed communication channel that can transfer data between the various components or devices. The backplane bus typically consists of a set of parallel data lines, address lines, and control lines that connect the various components or devices to each other.

There are diverse types of backplane buses, including the Industry Standard Architecture (ISA), Peripheral Component Interconnect (PCI), and Universal Serial Bus (USB). Each of these buses has its own characteristics in terms of speed, data transfer rate, and the number of devices that can be connected to the bus.

# 3 Device Drivers.

## 3.1 Definition of device drivers and their role

Device driver is an operating system component that provides uniform access to various devices and manages I/O to those devices [1]. They are software programs that serve as a bridge between an operating system and a machine's hardware parts. The purpose of it is to provide a standardized interface through which the operating system may communicate with the hardware device and control and manage its functions. Device drivers are sometimes also called as a software driver.

Device drivers are crucial for managing input and output operations between a computer and its peripheral devices. They act as translators between the operating system's general I/O commands and the specific ones required by the hardware device, allowing for smooth communication between the two.  I/O devices including keyboards, mouse, CD/DVD drives, controllers, printers, graphics cards, and ports, there are many distinct types of device drivers. It's possible to refer to a driver as a kernel-mode device driver when it's built into an operating system. User-mode device drivers are those that need the end user to manually download and install them.

Device drivers carry out a variety of functions inside an I/O subsystem, such as initializing and configuring the hardware device, controlling data transfers to and from the device, and handling errors and interruptions that happen during I/O operations. They are necessary for the I/O subsystem to operate effectively and efficiently as well as for the operating system to fully utilize the hardware devices.

## 3.2 Types of device drivers: kernel-level and user-level drivers

Device drivers in I/O subsystems can be roughly divided into two categories: kernel-level drivers and user-level drivers.

**Kernel-Level Drivers:**

privileged mode within the kernel of the operating system is where kernel-level drivers, often referred to as system-level drivers, run. They can interact with the hardware without going via user-space because they have direct access to it. Kernel-level drivers are in charge of low-level device management and communication tasks such memory allocation, interrupt handling, input/output operations, and device setup. Storage drivers (for hard drives, flash drives, and CD/DVD drives), network drivers (for Ethernet, Wi-Fi, and Bluetooth), and graphics drivers are a few examples of kernel-level drivers (for display adapters and GPUs). Kernel device drivers are more efficient and powerful than the user level drivers. If kernel drivers are poorly maintained, it could harm the system.

**User-Level Drivers:**

Application-level drivers are another name for user-level drivers, which run outside of the kernel in user-mode. System calls are used to communicate with the kernel-level drivers, which results in more overhead and worse performance as compared to the kernel-level drivers. User-level drivers are typically used for USB devices, printers, scanners, and other low-performance hardware. Printer, scanner, and USB drivers are a few examples of user-level drivers. User level drivers are also referred to as application-level drivers or non-privileged drivers.

Operating systems heavily rely on both kernel-level and user-level drivers to enable communication between software applications and a variety of hardware components.

## 3.3 Driver development and testing

Device drivers run in kernel mode and have full access to the system's resources, which has significant security implications for driver development and testing in I/O subsystems. A device driver security flaw could give hackers complete access to the system, allowing them to steal sensitive information, set up malware, or perform arbitrary code. Which is why, when creating and testing device drivers for I/O subsystems, security should be given the highest priority.

The development of device drivers requires testing. It is essential to test the driver under many circumstances, such as different component configurations, operating system iterations, and workload scenarios. To check the accuracy and efficiency of the driver, methods including unit testing, integration testing, and stress testing can be utilized.

security prevention controls that must be taken into consideration in general while developing drivers and testing. Code review is necessary to find possible security vulnerabilities in the device driver's source code, a complete code review is necessary. Race conditions, integer overflows, memory leaks, and buffer overflows should all be investigated during code review. Here are some security considerations,

Input validation is used to protect against buffer overflows and other input-related vulnerabilities, device drivers should authenticate all inputs from user-mode components. Before being used by the driver, all user-supplied data should be verified and cleaned.

Secure coding practices will reduce the possibility of adding security vulnerabilities to the device driver, developers should follow secure coding practices. These practices include avoiding the usage of dangerous functions, implementing proper error handling, and applying safe memory allocation and deallocation practices.

Security testing finds security flaws in the device driver, developers should conduct security testing. Fuzz testing, penetration testing, and vulnerability scanning should all be part of this testing.

Secure design**s** developers should build device drivers using security in mind. This involves implementing secure data storage techniques, implementing proper authentication and permission procedures, and establishing secure channels of communication between both the driver and the user-mode component.

In conclusion developing and testing device drivers properly is a challenging task, however it is an important part that makes sure hardware will be used in the most efficient and reliable way in io subsystems and in overall operating system.

## 3.4 I/O request processing by device drivers

An I/O request is the interaction between the device drivers and the operating system to carry out input/output actions. A process sends an I/O request packet whenever it wants to read or write data to an I/O device or storage device. I/O request packet is a data structure in windows to request a file I/O that is sent from the I/O manager to the appropriate device driver [1]The programs and device drivers within the I/O devices control these I/O request packets.

I/O requests can be synchronous and asynchronous. in synchronous request the requesting process blocks until the operation is complete while in an asynchronous request it does not block the requesting process but allows it to continue an execution while the device driver is operating [1].Device drivers must handle continuous I/O requests from many processes and make sure the device is used in a way that is mutually exclusive. In order to do this, it must be able to handle requests from many processes in a way that prevents conflicts and guarantees that only one process may access the device at once.

## 3.5 Security aspects in device drivers

Device drivers play a critical role in an operating system's I/O subsystems. However, they are exposed to some security risks. The following list includes some of the known flaws in device drivers for an I/O subsystem.

- Memory corruption

  Memory corruption vulnerabilities happen when a computer's memory or software is changed without the users' knowledge or consent. Device drivers could be exploited via the above explained memory corruption vulnerabilities. Attackers can use arbitrary codes to exploit these vulnerabilities to crash a system.

- Denial of service attacks

  Denial of service attack, in which communications or processing resources are consumed so that they are unavailable to legitimate users [9]. Attackers can send malicious data to the device drivers in order to shut down a system or use too many resources.

- Injection attacks

  In an injection attack, an attacker provides a program with malicious data. Then that malicious code will be processed by a complier as a part of a command or query [9]. This alteration of code will change the way the software works.

- Privilege escalation

Privilege escalation occurs if an attacker can execute code with the privileges and access rights of the compromised program or a service. If these privileges are greater than those available already to the attacker [9]. attackers can use these vulnerabilities and can exploited to perform unauthorized actions.

To sum up, To mitigate the risks and make sure the system is safe, it's essential to implement secure coding techniques, update drivers often, and conform to best practices.

# 4 I/O Operation Management

## 4.1 Overview of I/O operation management

An integral part of a computer system is the I/O (Input/Output) subsystem. Between a computer's internal processing units and its external peripherals, it controls data transport. The I/O subsystem is made up of a number of parts that cooperate to guarantee reliable and effective data transport.

An essential component of the I/O subsystem is device drivers. A device driver is a kind of software that enables communication between an operating system and a particular piece of hardware. The I/O subsystem loads and unloads these device drivers, ensuring that they are available when needed and unloaded when not in use.

Another crucial component of the I/O subsystem is buffering. When information is sent between internal processing units and external devices, it is often delayed in memory. The I/O subsystem controls data buffering to deliver error-free and efficient data transfer. The numerous locations where this buffering may occur include the operating system kernel, the device driver, or application-level buffers, to name a few.

The scheduling of I/O requests is likewise handled by the I/O subsystem. According to criteria including urgency, data size, and the device being utilized, it prioritizes I/O requests. The I/O scheduler makes sure that each I/O operation is carried out as effectively and promptly as feasible without holding up other tasks.

The I/O subsystem's management of interruptions is another critical component. The I/O subsystem responds to interrupts from devices by suspending the running process and handling the interrupt. This guarantees that the data transfer is carried out properly and effectively.

The I/O subsystem also controls how errors are handled during I/O operations. Errors that might happen during data transfer, such as transmission faults, device malfunctions, and other problems, can be found and recovered from.

In summary, the I/O subsystem is a complicated system that must be managed carefully in order to provide dependable and effective data flow between a computer's internal working units and external peripherals. Together, its parts make sure that data is buffered, planned, and transported effectively and consistently. The effectiveness and performance of a computer system as a whole depend on the efficient administration of the I/O subsystem.

## 4.2 I/O Scheduling

### *4.2.1 Definition of I/O scheduling and its role in the I/O subsystem*

Process schedulers and I/O schedulers differ from one another largely in how they choose which CPU process should be executed next and which I/O operation should be performed on a storage device. The I/O Scheduler may manage and prioritize requests in many contemporary operating systems. The I/O scheduler and I/O traffic controller work together to determine which path is being used to process the current I/O request.

There are several types of I/O scheduling algorithms.

- *Deadline based I/O scheduling.*
- *Priority based I/O scheduling.*
- *Fair-share based I/O scheduling.*

**Deadline based I/O scheduling algorithm.**

Deadline-based I/O scheduling is a way of scheduling I/O tasks in operating systems based on their deadlines. This strategy ensures that I/O requests are responded to in a timely manner, improving overall system performance.

In this technique, each I/O request is assigned a deadline, which is the maximum length of time that the request can take to complete. The method maintains a queue of pending I/O requests and selects the next one to service based on the deadline.

When a new I/O request is received, the algorithm verifies its deadline and places it in the proper queue position. If the deadline is approaching, the request is given more priority and handled before those with later deadlines.

The primary benefit of deadline based I/O scheduling is that it assures that I/O requests are fulfilled as soon as possible. This is especially critical for real-time applications that must adhere to severe time constraints. Nevertheless, because requests with longer deadlines may be repeatedly preempted by requests with shorter deadlines, this method may lead to famine.

Many deadlines based I/O scheduling algorithms have been created, notably the Linux kernel's Deadline I/O Scheduler and the Earliest Deadline First (EDF) method. The EDF algorithm is a well-known and widely utilized algorithm in real-time systems. It chooses the request with the earliest deadline and handles it first.

- **Earliest Deadline First (EDF)**

  Earliest-deadline-firs (EDF) scheduling assigns priorities dynamically according to deadline. The earlier the deadline, the higher the priority; the later the deadline, the lower the priority. Under the EDF policy, when a process becomes runnable, it must announce its deadline requirements to the system. Priorities may have to be adjusted to reflect the deadline of the newly runnable process. Note how this differs from rate-monotonic scheduling, where priorities are fixed. [1]

The system contrasts newly added jobs' deadline dates with those of newly added tasks. If the deadline for the new work is earlier than the deadline for any previous assignment that has already been finished, it is given top priority and scheduled for execution. If the new job has a later deadline than a task that already exists, it is given priority and placed in the task queue.

**Priority based I/O scheduling.**

Priority-based I/O scheduling is a scheduling algorithm I/O processes in operating systems based on their priority levels. This technique prioritizes each I/O request and serves them in the order in which they were received. In priority based I/O scheduling, the operating system assigns a priority level to each I/O request based on the process that issued the request or the kind of I/O activity. The I/O scheduler maintains an I/O request queue with higher priority requests to become first. Priority-based I/O scheduling allows the operating system to expand usage of resources to best performance. Even though the low-priority request arrived earlier, a high-priority I/O request can be served before a low-priority one.

The I/O scheduler chooses the next I/O request to be fulfilled depending on its priority level during runtime. Higher priority I/O requests are handled before lower priority I/O requests, ensuring that high-priority I/O operations are performed fast.

Priority-based I/O scheduling allows the operating system to optimize the utilization of system resources. A high-priority I/O request, for example, can be served before a low-priority request, even though the low-priority request arrived first.

- **Completely Fair Queuing (CFQ)**

  The Completely Fair Queuing (CFQ) method, which is utilized in the Linux kernel, is one of numerous priority based I/O scheduling algorithms. I/O requests are prioritized by CFQ based on their process priority and the amount of time each process has been waiting for I/O. The method guarantees that each process receives a fair part of the available I/O bandwidth while still prioritizing I/O requests according to their priority levels.

  CFQ maintains three queues (with insertion sort to keep them sorted in LBA order): real time, best effort (the default), and idle. Each has exclusive priority over the others, in that order, with starvation possible. It uses historical data, anticipating if a process will likely issue more I/O requests soon. If it is so determined, it idles waiting for the new I/O, ignoring other queued requests. This is to minimize seek time, assuming locality of reference of storage I/O requests, per process. [1]

**Fair-share based I/O scheduling.**

Fair share scheduling is a form of scheduling method used in the I/O subsystem to allot I/O bandwidth fairly across various processes or users. It assures that regardless of the number of processes or users sharing I/O resources, each process or user receives a fair portion of the available I/O bandwidth. Each process or user according to fair share scheduling is given a portion of the available I/O bandwidth, determining the number of I/O operations they are permitted to do. The operating system then responds to I/O requests such that each process or user gets a fair share of the available I/O bandwidth. This is crucial for multi-user systems as it's important to guarantee that each user has an equal chance to access the I/O resources.

In conclusion, effective and reliable operation of modern computer systems depends on the execution of I/O scheduling appropriately. The I/O subsystem can help decrease wait times and improve system performance by prioritizing I/O requests based on a variety of factors, including the kind of I/O device, the urgency of the request, and the size and frequency of data transfers. Therefore, when choosing a scheduling mechanism, it is important to consider the features of the system and the types of I/O requests. With a well-designed I/O scheduling algorithm, consumption of resources, response times, and user experience can all be enhanced.

### *4.2.2 I/O request queues and their management*

In the simplest sense, a block request queue is exactly that: a queue of block I/O requests. If you look under the hood, a request queue turns out to be a surprisingly complex data structure. Fortunately, drivers need not worry about most of that complexity. Request queues keep track of outstanding block I/O requests. But they also play a crucial role in the creation of those requests. The request queue stores parameters that describe what kinds of requests the device is able to service: their maximum size, how many separate segments may go into a request, the hardware sector size, alignment requirements, etc. If your request queue is properly configured, it should never present you with a request that your device cannot handle [2]. I/O schedulers are classified into two categories. The request queue and the completion queue. Request queues assist to hold all outstanding requests, while completion queues help to hold all finished I/O requests. I/O schedulers are in charge of maintaining request queues and prioritizing requests depending on certain criteria.

I/O request queue management is critical to the performance of an I/O subsystem. Poorly managed I/O request queues can cause bottlenecks, higher latency, and reduced overall throughput. One of the most essential functions of I/O request queue management is scheduling I/O requests for execution. The I/O subsystem must set the order in which I/O requests are performed to optimize system performance. Several scheduling algorithms are used, including **First-Come-First-Served (FCFS), Shortest-Seek-Time-First (SSTF), SCAN and C-SCAN.**

In the First-Come-First-Served scheduling method, I/O requests are performed in the order in which they are received. 98, 183, 37, 122, 14, 124, 65, 67, in that order. If the disk head is initially at cylinder 53, it will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65, and finally to 67, for a total head movement of 640 cylinders [1].

Shortest-Seek-Time-First scheduling selects the I/O request with the shortest seek time, which is the time it takes the disk's read/write head to go to the requested data.

SCAN algorithm approach moves the disk head across the disk as I/O requests come. Although it is quicker than FCFS, it may result in longer wait times for particular requests.

simply moves across the disk servicing requests in order across the tracks. Let us call a single pass across the disk a sweep. Thus, if a request comes for a block on a track that has already been serviced on this sweep of the disk, it is not handled immediately, but rather queued until the next sweep. SCAN has a number of variants, all of which do about the same thing. For example, Coffman et al. introduced F-SCAN, which freezes the queue to be serviced when it is doing a sweep [CKR72]; this action places requests that come in during the sweep into a queue to be serviced later. Doing so avoids starvation of far-away requests, by delaying the servicing of late-arriving (but nearer by) requests. [10]

Circular SCAN (C-SCAN) algorithm is similar to SCAN in that the disk head advances in only one direction, servicing requests until it reaches the end of the disk, at which point it returns to the beginning and begins again. This can reduce wait times for certain queries while raising them for others.

Let’s return to our example to illustrate. Before applying C-SCAN to schedule the requests on cylinders 98, 183, 37, 122, 14, 124, 65, and 67, we need to know the direction of head movement in which the requests are scheduled. Assuming that the requests are scheduled when the disk arm is moving from 0 to 199 and that the initial head position is again 53. The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one. [1]

### *4.2.3 Real-time I/O scheduling and its challenges*

Real-time I/O scheduling is the process of prioritizing and scheduling input/output activities in real-time systems. I/O operations in real-time systems must be done within predefined deadlines to ensure that the system works efficiently. Real-time I/O scheduling is crucial because it ensures that time critical I/O activities are executed in a predictable and timely manner.

Real time I/O scheduling poses various difficulties. Among these difficulties are:

Overhead: To schedule and prioritize I/O operations in real-time, more processing overhead is required. This burden has the potential to degrade overall system performance and lengthen the response time of non-real-time processes.

Complexity: Real-time I/O scheduling is a difficult undertaking that necessitates the use of specialized algorithms to prioritize I/O activities. The algorithms must consider a variety of criteria, including the type of I/O operation, the task's priority, and the system load.

Real-time I/O scheduling can cause resource contention, especially in systems with a large number of I/O operations. This can result in time critical I/O activities being delayed or missing deadlines.

Interference: Real-time I/O scheduling can interfere with other system components and operations if the scheduling method is not correctly built. Unexpected behavior and system failures may occur as a result.

System variability can affect real-time I/O scheduling due to hardware and software changes, system load, and I/O device performance. Several factors can affect the predictability and dependability of I/O operations.

### *4.2.4 I/O scheduling optimization strategies, such as prefetching and caching*

I/O scheduling optimization strategies such as prefetching and caching strategies are commonly used in computer systems to improve the performance. These measures can assist in shortening the time it takes to access data from a secondary storage device.

One of the most frequent I/O efficiency techniques is caching. The fundamental principle behind caching us to keep frequently. Requested data close to where it will be used rather than having to retrieve it from a slower storage source. At multiple levels, such as the hardware level, the operating system level, or the application level, a cache can be implemented.

- Hardware level
  Modern CPUs (Central Processor Units) include built-in caches that store frequently requested data from main memory at the hardware level. The quickest and smallest cache is often located closest to the CPU, whereas bigger but slower caches are located farther away. The cache hierarchy is automatically managed by the CPU, which also transfers data across caches, as necessary.

- Operating system level
  A file system cache that caches frequently accessed data from disk or other storage devices can be implemented by the kernel at the operating system level. When requesting data from the storage device in response to an application request, the kernel first determines if the requested data is already in the cache. This can speed up data access considerably, especially for programs that read or write huge files.

- Application level
  Developers can design their own caching algorithms to save frequently used data in memory or on disk at the application level. A web browser, for instance, might cache frequently used web pages, photos, or other resources to speed up the loading process for incoming requests.

Now imagine the file open example with caching. The first open may generate a lot of I/O traffic to read in directory inode and data, but subsequent file opens of that same file (or files in the same directory) will mostly hit in the cache and thus no I/O is needed. [10]

Prefetching is another I/O optimization technique that includes predicting data needs and fetching it into the cache before the actual request is made. Prefetching can be used in a variety of ways, such as at the hardware or operating system levels.

For most pages, the OS simply uses demand paging, which means the OS brings the page into memory when it is accessed, "on demand" as it were. Of course, the OS could guess that a page is about to be used, and thus bring it in ahead of time; this behavior is known as prefetching and should only be done when there is reasonable chance of success. For example, some systems will assume that if a code page P is brought into memory, that code page P +1 will likely soon be accessed and thus should be brought into memory too. [10]

Modern CPUs are equipped with advanced prefetching circuitry that allows them to anticipate which data will be needed next and proactively prefetch it into the cache. These methods rely on sophisticated algorithms that examine memory access patterns and attempt to predict upcoming accesses.

The kernel at the operating system level can implement a prefetching technique that anticipates which data will be needed next and obtains it from the storage device before it is requested. This can be especially useful for applications that read large files sequentially, such as video players or scientific simulations.

In summary, I/O scheduling optimization techniques like prefetching and caching are essential for enhancing computer systems' efficiency. By keeping frequently requested data in a location that is quicker and easier to access, these techniques seek to decrease the amount of time it takes to retrieve data from storage devices like hard disks or solid-state drives. Depending on the particular requirements of the system or application, caching and prefetching can be done at several levels, such as the hardware level, the operating system level, or the application level.

#### *4.2.5 Vulnerabilities and countermeasures in I/O scheduling*

The I/O scheduling algorithm is in charge of prioritizing and handling the incoming I/O requests from various devices. Nonetheless, there are various vulnerabilities linked with I/O scheduling that can be exploited by attackers. In this section, we will look at some of the weaknesses and countermeasures in I/O scheduling based on the I/O subsystem.

- Denial-of-Service Attacks
  DoS attacks are attempts to disrupt a system's normal operation by flooding it with a large number of requests. A DoS attack may cause the I/O subsystem to

slow down or crash by flooding it with a large number of I/O requests in the case of I/O scheduling. The I/O scheduling system should limit the number of concurrent I/O requests to mitigate DoS attacks.

- Resource Exhaustion
  When a malicious user submits a large number of I/O requests that consume all available resources, like CPU, memory, and disk space, a vulnerability known as resource exhaustion occurs. To prevent resource depletion, the I/O scheduling algorithm should limit both the number of I/O requests that a user can give and the number of resources that each request can consume.

- Privilege Escalation
  A vulnerability occurs when a hostile user gains elevated access to the system, allowing them to modify the I/O scheduling algorithm or evade security protections. To avoid privilege escalation, the I/O scheduling algorithm should be designed with security in mind, with only trusted users allowed to change its settings.

- Malicious Code Execution
  Malicious code execution is a security flaw that happens when an attacker executes code that exploits a flaw in the I/O scheduling algorithm or the operating system. The I/O scheduling algorithm should be developed with security in mind and should only enable trustworthy programs to be performed to prevent malicious code execution.

In conclusion, The I/O scheduling algorithm is in charge of prioritizing and managing receiving I/O requests from various devices. Nonetheless, there are several vulnerabilities connected with I/O scheduling that attackers can exploit. To avoid these vulnerabilities, the I/O scheduling algorithm should be developed with security in mind, and countermeasures such as restricting the number of I/O requests, setting resource limitations, and ensuring that only trusted users and code may access the system should be implemented.

## 4.3 Direct memory access (DMA)

### *4.3.1 Definition of DMA and its role in the I/O subsystem*

Computer designers long ago invented a mechanism for offloading the processor and having the device controller transfer data directly to or from the memory without involving the processor. This mechanism is called direct memory access (DMA). DMA is implemented with a specialized controller that transfers data between the network interface and memory independent of the processor, and in this case the DMA engine is inside the NIC [3].

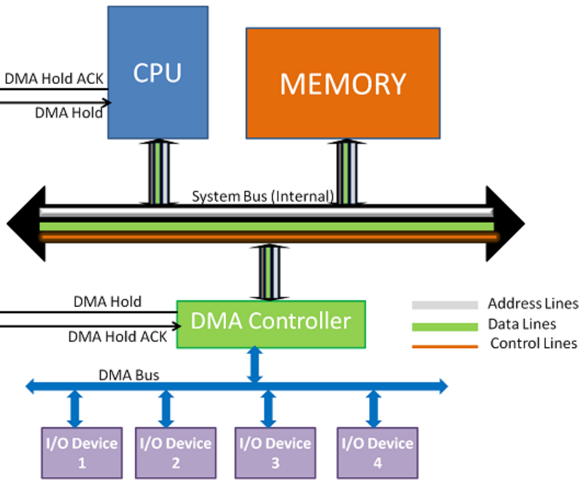

*Figure 3 Direct Memory Access controller and I/O Processor | Computer Architecture (witscad.com)*

Data transmission mode

There are three data transmission modes in direct memory access controller.

- Brust mode:

A complete block of data is shared in burst mode in a single, continuous sequence. Since the CPU has granted the DMA controller access to the system buses, it sends all of the data in the data block earlier and hands back control of the system buses to the CPU. Although the CPU is rendered dormant for associatively extended periods of time, this mode is useful for putting programs or data records into memory.

- Cycle stealing mode:

The BR(Bus Request) and BG(Bus Grant) signals are used by the DMA controller in cycle stealing mode to access the system buses similarly to burst mode. After sharing one byte of data, its desserts BR, and hands back control of the system buses to the CPU.

Just before sharing its entire block of data, it already sends requests via BR, sharing one byte of data every request.

- Transparent mode

Transparent mode requires the greatest time to exchange a block of data, but it is also critical for overall system performance. The DMA controller only distributes data in transparent mode when the CPU is performing activities that do not require the system buses.

### 4.3.2 types of DMA

There are several variations of Direct Memory Access (DMA) used to transfer data between peripheral devices and main memory.

- *Third party DMA*
- *Bus Mastering DMA*
- *Peer to peer DMA*

Third party Direct memory access

Data is sent between the main memory and the peripheral device using a third-party DMA controller. This kind of DMA is mostly used in embedded systems because a processor doesn't have built in DMA. The CPU is free to perform other tasks, but the third-party DMA controller manages data transmission and controls the bus.

Bus Mastering Direct memory access

In using bus mastering DMA, the peripheral device takes control of the bus and sends or receives data directly to or from memory. The peripheral device, acting as the bus master, can initiate and complete data transmission independently of the CPU. Bus mastering DMA is widely used by high performance systems like graphics cards and storage units.

Peer to peer Direct memory access

Peer-to-peer DMA is a direct communication method between two devices that excludes the use of the CPU or the memory subsystem. Data is transmitted between network adapters using this type of DMA, bypassing the CPU, in networking equipment. Peer-to-peer DMA can reduce CPU overhead and latency associated with data transfers, which can increase system performance overall.

### 4.3.3 *DMA engines and their architecture*

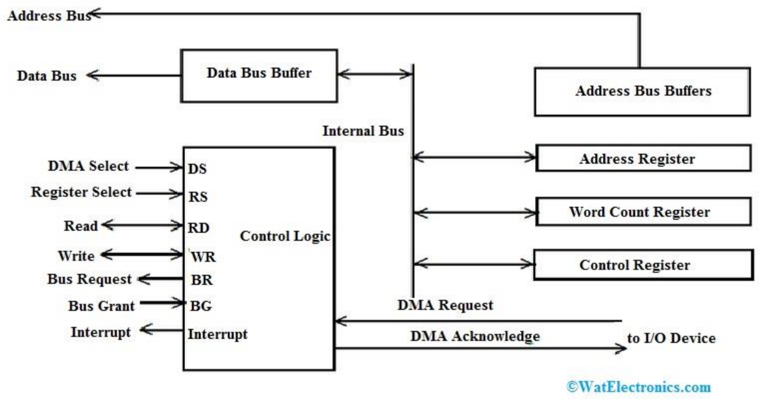

*Figure 4 DMA Controller: Architecture, Types, Working & Its Applications (watelectronics.com)*

The DMA controller block consists of 4 channels that can use up to four Peripheral Devices. Each channel has a 16-bit address and 14bit counter. Each DMA controller can transfer data up to 64kb.

The direct memory access controller reads, writes, and verifies the transferred data because it is just used for data transfers and doesn't perform any alterations. There are two modes of operation for the DMA controller: master mode and slave mode. When the CPU acknowledges a DMA controller, it operates in expert mode. Conversely, if the CPU doesn't acknowledge a DMA controller, it operates in slave mode up to that time. The lines connecting input/output devices and the control logic are DMA request and DMA acknowledgment. When the input-output device is prepared to transfer data, the CPU will initiate the DMA. During the initiation phase, the CPU will supply the address from which data must be written to or read.

### 4.3.4 *DMA channels and their management*

Physical channels called DMA channels are used to transmit and receive data between main memory and peripheral devices. It offers a system for controlling transfers efficiently. They are typically implemented with a system bus connected DMA controller. To start and finish transfers, each DMA channel is given a unique DMA request line and a DMA acknowledgment line. Several channels, each with its own set of registers and control logic, can be supported by the DMA controller.

Direct memory access channels are managed by the operating system. If peripheral devices need to transmit or receive data, it sends a DMA request to the operating system. Then the operating system allocates the DMA channel to transfer data between the main memory and peripheral devices. After setting up the transfer parameters, the device can start the transfer by activating the DMA request line connected to the channel that was assigned by the operating system. The DMA controller then begins the transfer and keeps track of its transfer progress. The DMA controller signals the device that the transfer is finished by raising the DMA acknowledgment line connected to the channel.

DMA channels are mostly allocated according to first come-first serve (FCFS) to avoid device clashes. If all devices are in use, the device might have to wait for channel become available. In some of the systems are using priority-based allocation for DMA channels. It’s given higher prioritized devices to access before lower prioritized devices.

A programmable I/O (PIO) or a DMA controller can be used to handle DMA channels. With PIO, the transfer is controlled by the CPU, and each byte of data is transmitted individually. While using DMA, the transfer is controlled by the DMA controller, which has the ability to send data in larger blocks, often in blocks of 16 or 32 bytes.

In conclusion, DMA channels provide a technique for effectively controlling data transfers between peripherals devices and main memory. They are normally controlled by the operating system and implemented using a DMA controller. DMA channels may be distributed based on priority or on a first-come, first-served basis. The system

### *4.3.5 DMA transfer modes: block mode and scatter-gather*

There are two primary transfer modes in DMA.

- Block mode
- Scatter-gather Mode

Block mode:

To initiate a DMA transfer, the host writes a DMA command block into memory. This block contains a pointer to the source of a transfer, a pointer to the destination of the transfer, and a count of the number of bytes to be transferred. A command block can be more complex, including a list of sources and destinations addresses that are not contiguous. [1]

Scatter-gather mode:

This scatter–gather method allows multiple transfers to be executed via a single DMA command. The CPU writes the address of this command block to the DMA controller, then goes on with other work. [1]

Scatter-gather mode compared to block mode, scatter-gather mode is more flexible and adaptable. but scatter-gather mode required more hardware and software support than block mode. Choosing transfer mode depends on the requirements of the application. Both block mode and scatter-gather mode offer advantages and disadvantages. While scatter-gather mode is more adaptable and ideal for non-contiguous data transport, block mode is easier to use and more effective for continuous data transfer.

### *4.3.6 DMA programming and configuration*

There are various processes required in programming and setting DMA. It necessitates a thorough grasp of hardware and system architecture.

1. Identifying the DMA controller:
   At this phase, the DMA controller can be identified by examining the system documentation or by querying the operating system.
2. Configure the DMA Controller:
   After the DMA controller has been found, the following step is to configure it by setting the relevant control registers. Setting the DMA transfer mode (block or scatter-gather), DMA channel number, source and destination addresses, and transfer count are all part of this. According to the kind of DMA controller utilized, the configuration procedure may be different.
3. Allocate Memory for the DMA Transfer:
   DMA transfers need the use of memory which is accessible by both the device and the DMA controller. Memory can be allocated by providing a technique like kmalloc () or vmalloc (). To avoid data corruption, memory should have been allocated with correct alignment and cache management.
4. Create DMA Descriptors:
   DMA descriptors are used in scatter-gather mode to indicate the position and length of each data block to be sent. These descriptors are generally kept in memory in a buffer. To minimize data corruption, the DMA descriptor buffer should be allocated with suitable alignment and cache management.
5. Start the DMA Transfer:
   Once setting the DMA controller, allocating memory, and defining DMA descriptors, the DMA transfer may be initiated by adding to the DMA controller's suitable control register. The DMA controller will then initiate the transfers.
6. Wait for the Transfer to Complete
   After launching the DMA transfer, the system must wait for it to complete before processing the data. This can be done by polling the appropriate status register or by using an interrupt. The CPU can perform other activities while waiting for the DMA transfer to complete.
7. Release Resources:
   To minimize memory leaks, when the DMA transfer is finished, the allocated memory and any other resources utilized for the DMA transfer should be removed.

In summary, DMA programming and setup is a critical method that may considerably increase the performance of computer systems. Knowing the system architecture and hardware is critical for appropriately programming and setting DMA transfers. The techniques described above will help in programming and configuring DMA transfers in the system.

### *4.3.7 DMA performance optimization techniques*

DMA is effective in Input/Output (I/O) subsystems, where devices like network cards, sound cards, and storage devices must transmit data fast and effectively between the system memory and the device. DMA technology has substantially increased I/O subsystem performance and enhancing DMA performance can further enhance system performance.

Improving DMA performance includes several strategies that assist in increasing data transmission speed and efficiency.

- Increasing block size
  By increasing the block size in DMA transfers can increase speed by reducing the number of DMA transfer requests needed. A bigger block size ensures that more data is sent in each transfer, lowering the total number of transfers necessary.

- Minimize CPU involvement.
  Reducing CPU participation in the DMA transfer process can assist in enhance performance by minimizing the time that the CPU takes processing data. Utilizing DMA controllers with effective interrupt handling can reduce the CPU's participation in the data transmission process, letting the CPU to focus on other activities.

- Using DMA channels efficiently
  By minimizing conflict between several devices attempting to contact the DMA controller at the same time, optimal use of DMA channels can assist to enhance performance. Each device may transport data separately without vying for the same resources by assigning distinct DMA channels to various devices.

- Using memory caching
  Memory caching can increase speed by minimizing the amount of time the CPU spends waiting for transferring data to finish. Caching frequently requested data in memory can minimize the amount of DMA transfers necessary, boosting overall system efficiency.

- Using scatter-gather DMA
  Scatter-gather DMA enables numerous DMA transfers to take place in a single operation, lowering the number of interruptions necessary and enhancing system performance.

- Using burst mode
  Burst mode DMA enables the DMA-enabled device to transmit data in blocks without the need for the CPU's assistance. By decreasing the CPU's participation in the data transmission process, this mode can increase performance.

### *4.3.8 Vulnerabilities and counter measures in DMA*

DMA is an effective technology that is used in modern computer systems to increase efficiency by enabling some Peripheral devices to access system main memory directly without involving the CPU. According to that DMA can be used by attackers to get unauthorized access to sensitive data. however, means that it can potentially present serious security issues.

Attackers mostly use malicious peripheral devices to exploit DMA, which is one of their main techniques. These devices may be connected to the system through USB or other interfaces and may be built to take advantage of DMA security flaws to access information. for example, a malicious USB device, can be used to launch a DMA attack by injecting malicious code into system main memory, which can subsequently be executed to gain control of the system.

To reduce the risk of DMA threats, appropriate security measures must be implemented. Restricting DMA access to just authorized devices is one of the most effective methods. This is accomplished by configuring the system to permit only DMA-capable devices that have been properly authorized and permitted to access system memory. Administrators, for example, can utilize hardware-based security mechanisms such as the Input/Output Memory Management Unit (IOMMU) to restrict DMA access only for authorized devices.

Another critical security implement is to use secure boot protocols to prevent malicious programs from being executed during the system boot process. Secure boot guarantees that only trusted software and firmware are loaded during the boot process, preventing unauthorized code from being executed and exploiting DMA vulnerabilities.

In conclusion, DMA technology is a powerful mechanism that is used in I/O subsystems, but it also poses significant security issues. To reduce the risks of DMA attacks, proper security measures must be implemented, such as restricting DMA access to unauthorized devices, utilizing hardware-based security measures such as IOMMU, and creating secure boot methods to prevent the execution of malicious code. Organizations may dramatically decrease the risk of DMA attacks and safeguard sensitive data from unauthorized access by implementing these counter measures.

## 4.4 Interrupt handling

### *4.4.1 What is an interrupt handler?*

Interrupt handlers are also known as interrupt service routines. Interrupt service routine (ISR) is an operating system routine that is called when an interrupt signal is received [1]. This is which the interrupts operate. Every instruction is executed by the CPU after the interrupt request line, which is a wire, that we call. CPU saves the running processes and launches the interrupt handler routines in a fixed location in memory when it detects a signal declared by controllers in the interrupt request line. The interrupt handlers then resolve the caused interrupt and continue all the required processing. Interrupt handler then performs a state restore and execute a return from interrupt instruction to return to the execution state prior to the interrupt [1]. To summarize what has happened up to this point,

- Device controllers first declare a signal on the Interrupt request line to raise an interrupt.
- Interrupts are received by the CPU and sent to the interrupt handler.
- Interrupt handlers resolve the interrupt and continue the processes.

In conclusion, interrupt handlers are essential for computer systems to operate. Interrupts are used to enable the most urgent work to be done first efficient interrupt handling is required for good system performance [1].

### *4.4.2 Types of interrupts: hardware interrupts, software interrupts.*

Hardware parts or software events have the ability to produce a variety of interrupt kinds.

Hardware Interrupts

Hardware interrupt is an electronic signal which is sent from an external hardware device that signals it requires the OS to be paid attention hardware interruptions typically originate from a variety of sources, including disk drivers, I/O ports, and other sources. For example, moving the mouse or pushing a keyboard key can be taken as some examples for hardware interruptions. In these instances of interruptions, the CPU must stop the current process in order to read the current mouse location or input.

There are three types of hardware interruptions.

1. Maskable interrupts

Maskable interrupts can be turned off by the CPU before the execution of critical instruction sequences that must not be interrupted. The maskable interrupt is used by device controllers to request service [1].

Interrupts that can be disabled or ignored by the instructions of CPU are called as maskable interrupts. Usually, when an interrupt occurs, CPU saves the running processes and launches the interrupt handler routines in a fixed location in memory when it detects a signal declared by controllers in the interrupt request line. The interrupt handlers then resolve the caused interruption and continue all the required processing. However, in some cases CPU may be unable to respond to an interruption until it has finished a critical task. In these kinds of situations CPU can mask the interrupt or disable the interrupt, making it delayed or ignored until the CPU re-enables it.

For example, if a computer is performing a heavy data transfer between CPU and a hardware or sensitive operation that must not be interrupted, the CPU can use maskable interruptions till the completion of the critical tasks.

These maskable interrupts can be either,

- level triggered interrupts

  in level triggered interrupts by holding the interrupt signal at a certain active logic level, an interrupt is produced by this interrupt module. The interrupt remains active as long as the signal is held at a certain level. Once the CPU instructs it after the device has been serviced the signal is ignored. Level triggered interrupts are frequently utilized in systems where an uninterrupted signal needs to be monitored.

- edge triggered interrupts.

  In edge triggered interrupts, interrupts are only generated when it detects and asserting edge of the interrupt source. The edge may be detected when the interrupt source level actually changes, or it may be detected by periodic sampling and detecting an asserted level when the previous sample was deserted. While level triggered interrupts were used where an uninterrupted signal to be monitored in edge triggered interrupts, interrupts are often used in digital systems where a signal change needs to be captured and acted upon quickly.

2. Non-maskable interrupt

   An interrupt that cannot be delayed or blocked is called as a non-maskable interrupt [1]. Non maskable Interrupts cannot be disabled or ignored by the instruction of CPU. When response time is crucial or when an interrupt should never be switched off during regular system operation, a non-maskable interrupt is often utilized. They are reserved for events such as unrecoverable memory events [1].

3. Spurious interrupts

   Another sort of hardware interrupt is a spurious interrupt, which happens when the hardware alerts the CPU to an interrupt even when there is not actually a hardware activity that requires attention.

   Spurious interrupts can be caused by,

   - Electromagnetic interference
   - Faulty hardware
   - Bugs in system software

- Power issues
- Incompatible hardware

Identifying the source of the issue is crucial for determining and fixing spurious interrupt problems. Individual hardware components may need to be tested, system software and drivers may need to be updated, and system settings may need to be changed.

Software interrupts

Software interrupts are software-generated interrupt which are also called a trap. The interrupt can be caused either by an error or by a specific request from a user program that an operating-system service be performed [1], Interrupt that is generated by software or a system rather than hardware is referred as a software interrupt. Software interrupts are also often referred as exceptions. They act as a signal for an operating system or system service to do a certain task or react to an error event. A method to communicate with the kernel or execute system calls is thought to be via a software interrupt. There are two categories of software interrupts,

1. Normal interrupts

Normal interruptions are produced by internal events like a timer or a software-triggered interrupt as well as external devices like a keyboard or mouse. these Interrupts are normally handled by the operating system, which switches control to a specific interrupt handler and momentarily halts the execution of the currently running application. The operating system continues the interrupted program's execution once the interrupt has been addressed.

Examples of normal interrupts include,

- Timer interrupt
  This normal interrupt is generated by a hardware timer, which can cause interruption at regular intervals. Operating systems frequently use this to carry out operations like scheduling procedures, changing the system clock, or processing user input.

- Input/output interrupt

  When an external device like a keyboard or mouse needs to communicate with the processor these interrupts are generated.

- Software interrupt

  These interrupts occur when a software program needs the operating system to provide a service, such as allocating memory or granting access to a file.

2. Exception

Exception A software-generated interrupt caused either by an error or by a specific request from a user program than an operating-system service be performed [1]. Exceptions are also identified as automatically occurring traps. In general, exceptions don't come with any clear guidelines. As a result, an exception happens because of an "exceptional" situation that develops while the program is being executed. a trap or an exception, which is a software-generated interrupt caused either by an error or by a specific request from a user program that an operating-system service be performed by executing a special operation called a system call [1].

Examples of exceptions include,

- division by zero exception

  This interrupt occurs when a software tries to divide a number by zero. When this operation fails, the processor will identify the fault and hand over control to an exception handler so that the problem may be resolved.

- an invalid opcode exception

  An instruction that the processor does not recognize is referred to as an invalid opcode. When the processor tries to execute an invalid opcode, this exception occurs.

- page fault exception

  when a program tries to access a page of a memory that is not currently available in physical memory, usually because it has been changed out to disk. By changing that required page back into physical memory, the operating system will handle this error.

When an exception error occurs, a special exception handler is given control of the processor. The special exception handler is responsible for dealing with the situation and taking necessary actions such as, suspending the program or display an error message.

### *4.4.3 Interrupt latency and its impact on I/O performance*

Interrupt latency refers to the period of time from the arrival of an interrupt at the CPU to the start of the routine that services the interrupt [1]. I/O operations are handled via interrupts, and the latency of these interrupts may have a significant impact on I/O performance.

The processor must prioritize interrupt requests and respond to them immediately if multiple devices send them at the same time. Lost data and decreased I/O performance may occur if the CPU is unable to respond to all requests promptly due to excessive interrupt latency.

There are several factors that can have a significant impact on interrupt latency which include hardware and software used in a system. I/O systems use methods like interrupt coalesce and interrupt moderation to increase overall performance. This would more effectively postpone many interrupts, which would lower latency and performance.

### *4.4.4 Interrupt vectoring and prioritization*

Interrupt vector is an operating-system data structure indexed by interrupt address and pointing to the interrupt handlers. A kernel memory data structure that holds the addresses of the interrupt service routines for the various devices [1].

The purpose of vectored interrupt mechanism is to reduce the need for a single interrupt handler to search all possible sources of interrupts to determine which one needs service [1].but, Computers, however, have more devices than there are interrupt vector address units. With no interrupt vectoring, it would take a long time for the CPU to examine each I/O device to determine if it has created an interrupt.

On the other hand, selecting the order in which the CPU should handle interruptions is known as interrupt prioritization. The interrupt mechanism implements a system of interrupt priority levels. These levels enable the CPU to defer the handling of low-priority interrupts without masking all interrupts and makes it possible for a high-priority interrupt to preempt the execution of a low-priority interrupt [1]. By ensuring that the most essential interruptions are handled first by the CPU, interrupt prioritization lowers the chance of losing critical data and boosts system performance.

In summary, interrupt vectoring and prioritization serve as essential to managing data flow in I/O subsystems and providing smooth interaction among the CPU as well as I/O devices.

### *4.4.5 Security aspect*

Managing interruptions is crucial from a security standpoint. The operating system must transfer control to the appropriate interrupt handler when an interrupt occurs. If not properly secured, this interrupt handling procedure is vulnerable to attacks. The following are some security considerations taken for interrupt handling.

- Privilege escalation

  Since interrupt handling codes run with the kernel privileges, it may have access to private or sensitive information and maybe used to escalate user privileges.

- Denial of service attacks

  Dos attacks overload the system by producing constant interruptions, which makes it incapable of finishing its usual operations. To avoid this, systems could include safeguards.

- Timing attacks

  A timing attack allows the attacker to gain access to data that the program has accidentally disclosed. This disclosed information can be used to determine the information systems status and carry out cryptographic attacks.

To sum up, interrupt handlers play a significant role in I/O subsystems, hence it's critical to prevent any vulnerability from being exploited.

## 4.5 I/O virtualization.

### *4.5.1 Definition of I/O virtualization and its roles in the I/O subsystem.*

I/O virtualization (IOV) is the process of the *encapsulation of physical input and outputs.* It decouples virtual I/O from physical I/O. This introduced a level of indirection between abstract and concrete.

I/O virtualization is a major part in I/O subsystem by enabling efficient sharing of I/O resources among Multiple Virtual machines. Without I/O virtualization, each Virtual Machine requires dedicated access to the physical I/O resources which will lead to poor resource utilization and increased costs. In one hand virtualization helps to improve the system performance by enabling the hypervisor to optimize I/O Operations and its latency. Using this concept a one guest machine can run multiple operating systems concurrently, each one on its virtual environment.

Role of the virtualization systems depends on the decoupling of a VM's logical I/O devices from its physical implementation. Examples are wide form ability to multiply many virtual machines on the same hardware to advanced virtualization features such as Live mitigation and enhanced security. Decoupling enables time- and space multiplexing of I/O devices, allows them to multiply logical devices to be implemented by a smaller number of physical devices. Running varies operating system environments on the same machine relies on this feature. The ability to multiplex logical I/O devices into many ones allows both administration and automated systems to drive I/O devices at higher utilization and achieve better hardware efficiency. Evolution of virtualization through past decades can be attributed to the significant cost savings resulting from such basic partitioning and sever consolidation.

Compatibility and portability enable the same VM to run on any computer with different I/O devices and configurations, with the I/O virtualization layer provides the basic necessaries. Another role of virtualization is ability to suspend and resume a virtual device and save it to the Storage. Moving a running virtual machine between physical machine which referred as live mitigation. For both events, the active logical devices must be decoupled from physical devices and recoupled when the VM resumes after saved or moved, this cannot be done without the I/O virtualization.

### *4.5.2 Types of I/O virtualization.*

In handling I/O virtualization there are several types of I/O virtualization techniques that depend on what type of virtualization and specific needs of the systems. Here are some common techniques:

**Emulated I/O.**

Emulation is the concept of creating, simulating the hardware of one system on in another system. It mimics the qualities and logic of one processor to run in another efficiently, and simply allows the virtual machine to interact with it as a physical device. This allows the software that designed to use in the emulated system to run on the basic system.
This approach is easy and plain, but it can be slow and inefficient because of the constant translation of every I/O request into the physical device format. Normally the guest emulation believes it has exclusive control on I/O devices.

**Full virtualization**:

Similar to the emulation, Full virtualization is a concept of Simulating the whole hardware environment rather than simulating only the hardware like emulated I/O virtualization. This process includes CPU, memory, storage devices, and other network interfaces. The VM operates it as if it were running on a physical machine, but it is running on a hypervisor that manages access to the physical hardware.

**Para-Virtualization I/O:**

Paravirtualization is a concept in virtualization used for more I/O performance.
It involves modifying the guest operating system (The guest term is commonly used to distinguish the layer of software running within the virtual machine) to communicate directly with hypervisor or virtuzlization software layer, bypassing the traditional hardware abstraction layer. [4]
As a result this reduce the overhead which caused to improve the input and output performance.

**Pass-though I/O:**

This virtualization is a technique used to give VMs direct access to physical hardware resources, including network adaptors and storage devices. This concept bypasses the virtualization software layer and have the control directly. It is also referred as Direct I/O and PCI (peripheral component interconnect) passthrough. [2]

**SR-IOV (Single Root I/O virtualization):**

In this virtualization technique the multiple virtual machines allow to share a physical network adaptor, while maintaining near-native network performance. It means a physical network adaptor allows to be divided into multiple virtual functions, each of which can be assigned directly to a virtual machine. Because of this each virtual machine will be able to have its own virtual network adaptor, with direct access to physical adaptor.

### *4.5.3 Virtual Machine Managers.*

What is a Hypervisor?

Hypervisor is a Virtual Machine Monitor (V.M.M) a software layer that allows multiple operating systems to share a single physical machine using creating and managing virtual machines. A guest operating system manages application and virtual hardware, while the hypervisor manages the virtual machines and physical host hardware [5].

Hypervisor is creating a virtual machine environment that emulates the hardware of a different computer architecture (except in the case of paravirtualization, discussed in above 4.5.2 section, allowing the software designed to be used in that architecture to run with the host system at the same time. This is known as "hardware emulation" or "software emulation", and it's a technique that has been used for many years. [1]

The implimentation of virual machine managers are widely depends on the need and type. Some options are following:

**Type 0 : hypervisors:** Are Hardware-Based Solutions provide support VM creation and management using firmware. These VMMs which commly found in mainframe and large to midsized servers [1].

**Type 1: hypervisors:** Software-Based Special Purpose Operating systems have software build to provide virtualization. Including VMware ESX, Citrix Xen. [1]

General purpose OS that provide standerd funtions as well as VMM funtions including MS Windows Server with HyperV, And RedHats KVM feature (KVM is linux-basedKernal Based Virtual Machine Manager just like the HyperV the windows one) The similar They are to type 1 they also refferd as type 1 hypervisors.

**Type 2: hypervisors:** Applications that run on standard operating system which provides the Virtual Machine Manager features to guest operating systems are refferd as type 2. Some Examples are, Parallels Desktop, and Oracle VirtualBox, Workstation and Fusion, and VMware. [1]

From the following figure you will understand how the VMM and VMs are work on OS.

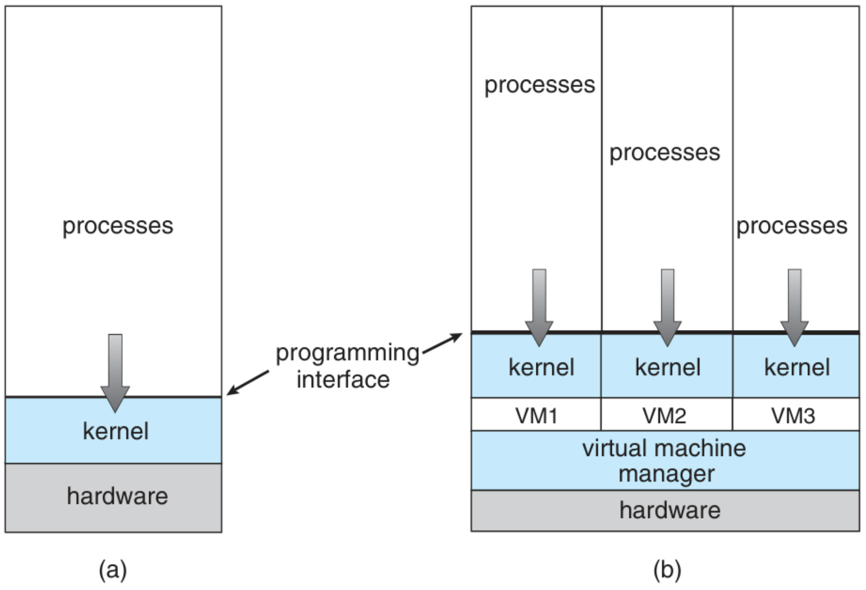

*Figure 5 System Virtualization (a) non-virtual machine (b) Virtual machine.*

### *4.5.4 Virtual I/O devices and their management*

Virtual Input and Output devices are software constructs that emulates the functionality of physical I/O devices in a Virtualized Environment. In Virtualization, Each Virtual Machine Owns its own set of I/O devices, which are mapped to the physical devices.

**Virtual Network Adaptors**: Emulates the functions of physical network adaptor, Helps in Communicating and Networking.

**Virtual Storage Controllers**: Emulates the physical storage. Separated storage for the VM.

**Virtual Graphic Adaptors**: Emulates the functionality of graphics adaptors, allowing VM to display graphics output.

**Virtual Sound Cards**: Emulates the sound adaptors, which helps in output audio.

**Virtual USB Controllers**: Emulates the functionality of Physical USB controllers.

Virtual I/O devices provides the basic mechanism for virtualization, allowing Virtual Machines to access I/O resources without requiring direct access to the physical hardware. These Input and Output devices are managed by the hypervisor, which will discuss in following section.

### *4.5.5 Hypervisor I/O management and configuration.*

Managing input and Output operations in a virtualized environment process is referred to as the hypervisor I/O management. When a multiple virtual machine shares a single physical resource

There are some techniques used for hypervisor I/O virtualization management,

**I/O Resource Allocation:** Allocating portion of physical resources to a virtual machine, such as a specific amount of network bandwidth or storage capacity. This ensures that each virtual machine has sufficient resources and prevents Virtual Machines from monopolizing the resource (Monopolizing is an act of controlling other resources or get own others space).

**I/O Resource Sharing:** The hypervisor can allow multiple Virtual Machines to share a physical resource, like a network adaptor. Using techniques like virtual switches or Network Address Translation (NAT), Which provide Virtual Machines with own Virtual Network interfaces that are mapped to physical adaptor.

**I/O Device Passthrough:** The hypervisor supports device passthrough, which means allows a virtual machine to directly access the physical device. Beneficial for high performance I/O device that require to direct access the hardware.

**Quality of Service (QoS):** As a Virtual Machine Manager the hypervisor can prioritize I/O operations based on their importance or criticality. Which Beneficial for critical applications that needs necessary resources when needed and preventing less important applications from slowing the performance of the entire system.

**I/O Scheduling:** Similar as the QoS the hypervisor can schedule I/O operations to prevent conflicts and ensure that the physical resource is used efficiently. As an example, hypervisor can buffer and reorder I/O operations to reduce the number of disks seeks or prevent network collisions.

**I/O Performance Monitoring:** Mostly the hypervisor is the software layer that monitors performance in each and every Virtual Machine which runs on the guest Operating system. It Provides with performance metrics such as I/O throughput, latency, and queuing delays to help administrators optimize the I/O performance.

Effective Hypervisor I/O management is critical for ensuring the performance and stability of virtualized environments. Hypervisor vendors typically provide tools and utilities for configuring and monitoring I/O operations like above mentioned. As well as integrating with third-party tools for advance monitoring and management like mentioned in 4.5.3 section. For further information you can see our references materials in below reference section. [4]

### *4.5.6 Virtual I/O bus architectures.*

Virtual I/O bus architecture is the way in which virtual I/O devices are connected to virtual machine, and to the physical devices in a virtual environment. The Bus Architecture provides the framework for managing the connectivity and communication between virtual I/O devices and the rest of the system.

There are several ways of virtual I/O bus architectures used in virtualization, including:

**Virtual PCI (Peripheral Component Interconnect)**: Virtual PCI is a method that emulates the PCI bus, which is a common bus used in physical systems for connecting I/O devices. PCI is a high performance I/O bus used to interconnect with peripheral devices in computing as well as communicating platforms [5]. Virtual PCI acts as physical device and allows virtual machines to use the same drivers and software that are used in the physical PCI devices.

**Virtual SCSI (Small Computer System Interface):** Small computer system interface is once popular type of connection for storage and other devices in a Personal Computer. It is cables and port connection method for certain type of hard drives, optical drivers, scanners, and other peripheral devices. The SCSI standard is no longer common in consumer hardware drivers, but it is still use in business and enterprise server environments. More recent versions include USB attached SCSI(UAS) and serial attached SCSI(SAS) [6]. The virtual SCSI is the same as the physical device connection interface. It emulates the physical device and allows VMs to connect with the SCSI devices.

**Virtual IDE (Integrated Drive Electronics):** Same as the SCSI connection interface IDE is legacy connection method nowadays hard to find. It was mostly used in Storage Devices and some peripheral devices for I/O operations. The virtual IDE is the same as the physical one which mimics the features of it and allows to use as it is. Nowadays we can see this change with SATA (Serial Advance Technology Attachment) which is also a faster and upgraded connection interface used.

**Virtual Network Adaptor Bus:** This is a bus architecture that allows VMs to connect with Virtual Network Adaptors. Same as the Physical Network Adaptor and provide the features of it. Because of this every VM can access the network and communicate.

These bus architectures are also Manages by the Hypervisor and by its techniques mentioned in 4.5.5.

### *4.5.7 I/O device sharing and virtual machine migration techniques ‘*

When we need an optimized resource utilization, improved flexibility, and enable workload balancing I/O device sharing and Virtual Machine migration techniques are doing a key role. Some of the techniques are:

**Virtual I/O device Sharing**: Sharing the same physical I/O device through the virtual I/O buses and network adaptors. This is the same as the resource sharing, we have discussed earlier in the above section 4.5.5.

**Live migration:** This technique enables virtual machines to be moved from one physical host to another while still in the running state. Without interrupting their services or disrupting I/O operations. Live Migration requires careful management to ensure the machine activity will be as it is, can continue to communicate with I/O devices, and maintain the performance during the process. Which will done by a Virtual Machine Manager or Hypervisor.

**Offline Migration:** Same as the Live migration offline migration will do when the Virtual machine is shutting down. This method is used when the live migration is no longer can done and needs to migrate to a different version or type of hypervisor.

### *4.5.8 I/O virtualization Advantages and Disadvantages*

Decoupling a logical device from its physical implementation offers many compelling advantages, Like:

**Scalability:** Virtualization allows for dynamic allocation and sharing of resources among VMs, which beneficial in efficient use of resources and enables easy scaling of applications.

**Resource sharing:** I/O virtualization enables Multiple VMs to share a physical I/O device, reducing hardware costs and improved resource utilization.

**Flexibility:** Ability of VMs easy change, move or configure them without being changing the physical hardware.

**Isolation:** Isolation helps the prevention of faults with other VMs and improved security among them.

Some of the Disadvantages are:

**Performance Overhead**: When virtualization the additional layers of processing between the VMs and the physical hardware can result in some performance overhead.

**Complexity:** Can be complex sometimes, requiring a specialized hardware and software with human knowledge. Which can increase the cost and complexity of virtualization environment.

**Compatibility Issues:** Compatibility can be limiting the options in virtualization environments and Issues in VMs when dealing with incompatible physical devices.

**Security Concerns**: Although virtualization has isolation and some access control privileges some security concerns like Network configurations, Hacker attacks, VM sprawls.

*VM sprawls:* when a VMs created specific workloads uncontrollable spread and abandoned after the serving their purpose is called as a VM sprawl. Because of this VMs sensitive information can be compromised due to not being actively managed and updated. [7]

### 4.5.9 *Performance considerations for virtualized I/O.*

**Latency:** When a Virtualized machine do I/O operations additional latency will happen due to the layers of abstraction and indirection between the virtual machines and physical devices. This can result in slower response times and reduced application performance. To overcome this problem Hypervisors and virtualization platforms provide various optimization techniques, such as virtualization techniques like device passthrough, and DMA (direct memory access) acceleration. About DMA look the 4.4 section above.

**Bandwidth:** Virtualized I/O can also affect the availability of bandwidth, reason of the multiple virtual machines sharing a single physical device. As a result, Hypervisor and other Virtual Platforms provide various bandwidth management tools, such as the QoS policies (mentioned in 4.5.5 section).

**Overhead:** When the single physical CPU shared it results additional CPU overhead due to extra processing required to manage and emulated the VMs. This will occur reduce of system performance. To minimize this problem hypervisors and Virtual Platforms use various optimization techniques, such as para-virtualization which we already discussed above which will allows VMs to communicate directly with Hypervisor.

**Compatibility:** Due to the evolution of technology and in poor engineering not all I/O devices and drivers are fully compatible with virtualization platforms, which can be a result in reduce compatibility and performance issues. To overcome this problem *Driver Certification and Compatibility Testing* features are used by Hypervisor and Virtual Platforms.

*Driver Certification* is processing which hardware vendors or third-party developers can certifies the drivers for the use of specific virtualization platform. Certifies ensures the driver has been tested and verified to work properly in the Virtual Environments. And meets specific requirements and standards set by the virtualization platform vendor.

*Compatibility Testing* is the testing of I/O devices and drivers in virtualized environments to ensure that they will work effectively and do not cause any performance and compatibility issues.

**Monitoring Operations:** Effective Monitoring and Management I/O operations is critical in virtualized environments. This task will do by Hypervisor and Virtualization platforms with various techniques, such as enable administration to monitor, optimize I/O performance, and performance counters and Input and Output profiling [1].

### *4.5.10 I/O virtualization security considerations and best practices.*

I/O virtualization additional can have additional security considerations and risks which will needs to be manage carefully. Some of the key security considerations and best practices are:

**Isolation:** Isolation of VMs critically ensure the strong security. It reduces the probability of having vulnerable Virtual Machine. Hypervisor and virtualized platforms have features like Virtual Switches and network segmentation, to ensure the isolation.

**Access Control:** Control accessing is similar to isolation but is does a great role in virtualization I/O security. Best practices are including implementation of strict authentication and authorization policies, using rule-bae access control mechanisms, and Using encryption and password policies to protect sensitive data.

**Encryption:** Proving I/O traffics with encryption will improve the security of virtualized I/O. There some encryption methods used by Hypervisors and virtualized platforms like Virtual private networks (VPNs) and secure sockets layering (SSL) encryption.

**Updating:** Regular updates will ensure the security of Virtualized I/O which will result in every aspect in virtualized I/O.

## 4.6 I/O buffers and caches

### *4.6.1 Definition of I/O buffers and caches and roles.*

Buffer:

A buffer is a memory area that stores data being transferred between two devices or between a device and an application [1].

Caches:

A cache is a type of buffer that scores frequently accessed data in faster and more accessible location, such as the CPU cache or the Hard drive cache [1]. Region of fast memory that holds copies of data. Which in action be more efficient to access the cached data than original stored data. Helps to reduce the amount of time when taking accessing the data.

### ***4.6.2 Types of I/O buffers and caches.***

Buffer types,

Read/Write Buffer: This type of buffers is used to temporarily store data that is being read from or written to a storage device. Read/Write buffers are helps to smooth out any fluctuations in the transfer rate between the storage device and Computer main memory.

Network Buffer: Used in store the data temporarily that is being transmitted over a network. Help to reduce the impact of network latency and bandwidth limitations by temporarily holding data packets until they can be transmitted to their destination. To do the task in one time.

Translation Lookaside Buffer (TLB):
It is a buffer used to store recently accessed memory address translations. The TLB helps in reducing the time when taking the translation of virtual memory address to physical memory addresses.

Caches,

Processor Cache: A Cache which built into the CPU which will be used to store frequently accessed data and instructions.

Disk Cache: This is a type of cache which will use to store temporarily accessed data from storage devices, Like Hardware and other using storage devices.

Page Cache: This Cache is used to store the recently accessed pages of memory in the computer’s main memory.

### ***4.6.3 I/O buffering and caching,***

In network devices.

Devices such as routers and switches, also have buffers and caches to improve their performance and throughput. In the network devices they are used for smooth out traffic spikes and prevent packet loss due to congestion. Size of the buffer is matters because that affects the performance of the network device, using slightly larger one can help to absorb the burst of traffic and improve overall performance.

Using a cache to store frequently accessed data or routing information. This cache can include the MAC address table, ARP cache, and routing tables. Using this the network device can reduce the time needed to perform packet forwarding and routing and improve the performance.

### *4.6.4 Buffer and cache management policies and algorithms.*

When managing the buffer and cache memories there are some policies and algorithms that help to improve and increase the efficiency of the operating system. They are different on depending on the data accessing patterns and performance requirements of the system. Some policies and algorithms are:

Write-through: This is a policy which helps the buffer and cache and underlying storage devices simultaneously whenever data is written. It ensures that data is always up to date.

Write-Back: This is another policy which updates only the buffer and cache when data is written. The changes are later done in underlying storage devices when the data is evicted from the buffer or cache. It is more efficient than write-through as it reduces the number of rights to the storage devices.

Random Replacement: This is an algorithm randomly evicts data from the buffer or cache, which does not make any assumptions of data access probabilities.
Evict: force out.

FIFO (First in First Out) This is an algorithm evicts the data that was first add to the buffer or cache. It assumes the data that has been in the buffer and cache for a long time is less likely to be accessed in the near future.

LFU (Least Frequently Used) This algorithm evicts the data that accessed the lease number of times.

MRU (Most Recently Used) This algorithm evicts the data most recently used from the buffer and cache. It assumed the data evicts has been accessed recently is more likely to be accessed again in the future.

LRU (Least Recently Used) This algorithm evicts the lease recently used data from both buffer and cache.

### *4.6.5 Buffer and Cache security considerations and impact on performance.*

Even the cache and buffers have security considerations to prevent the sensitive information being leaked or compromised, so mainly there are two major security considerations.

Buffer and cache data protection: To protect the sensitive data like passwords, encryption keys, and other confidential data form the attackers who can gain access to buffer or cache the use of encryption and access control mechanisms can be applied. Using the appropriate amount of encryption on sensitive data before store in cache and buffer and access can be restricted to the authorized users, we can reduce the problems.

Cache and buffer data leakage: Leaking or compromising can be happened due to the bugs in the software, poor coding practices, or other vulnerabilities in the system. To prevent the problem proper management like clearing the buffer and cache by overwriting the memory with random data or securely deleting the memory content can be done.

## 4.7 Error handling

### *4.7.1 What is Error handling?*

Process of detecting and responding to errors that may occur during I/O operations in a computer program is referred as Error handling. [1] The OS that uses protected memory can guard against many kinds of hardware and application error, So the complete system will not in a system failure situation usually in each minor mechanical malfunction.

### *4.7.2 Error detection and reporting.*

As a general rule, I/O system call will be return a bit of information about the status of the call, signifying either success or failure as function. [1] (Return codes) Exceptions is a technique that used to detect, and, in this approach, an exception is thrown when the error occurs. Providing clear and informative error messages to the user or developer can also use as a technique in error detection which helps to minimize or stop occurring the error early in the development stage.
When reporting the errors logging and monitoring are the methods that are used in I/O systems. In logging when the error occurs, the program can write the information about the error to a log file including the time, details of the error code

and any relevant data about the I/O operation, in addition to logging monitoring the errors in real-time can be done to detect and report the errors as soon as they occur.

### *4.7.3 Recovery from errors.*

After detecting and reporting the error that occurred during the I/O operation recovering is the after stage that important to user or the developer to ensure the programs to continue to function properly. Some of the techniques used to recovery are, Retrying is a common technique from recovering the errors in I/O, because the I/O operation fail can be happened due to a temporary issue, such as a network or a file issue which will run properly in a delay. Rollback is the operation, which is another technique, as an example if a database fails during the transaction stage the program can roll back the transaction to ensure that the database is in a constant state. Using alternative ways can be useful sometimes in error occurrence, like in network failure. Having a recovering mechanism will help the I/O operations to run smoothly effectively and will give a good user experience.

### *4.7.4 Error handling Security considerations.*

For the reliability, usability, and maintainability of the system and for the protection of the system the error handling must be done perfectly, else the system will have security vulnerabilities and system malfunctions. Some security precautions are like, Validating user inputs: preventing the inputs from untrusting sources will be especially useful to prevent any attacks from happening. Avoiding exposing sensitive information also comparable situation to that. Proper use of the system helps to reduce those threats. Using proper error codes, using secure error loggings and, preventing error-based attacks are also major considerations to prevent any misuse of the errors that occurs during I/O operations.

# 5 Input and Output Operations

## 5.1 Overview of input and output operations.

Input-output (I/O) systems exchange information between computer main memory and the outside world. An I/O system consists of I/O devices (peripherals), I/O control units, and software to perform I/O transaction(s) through a sequence of I/O operations. I/O devices can be classified as serial, meaning bit streams can be transferred one bit at a time, or in parallel. Parallel devices have a wide data bus and can therefore transfer data in words of one or more bytes. Like any other activity in a computer system, I/O is a collective function of both hardware and software. The software that performs an I/O transaction for a specific I/O device is called a device driver. An example of such an I/O transaction is reading a database from disk to memory. To do this, software is simply a sequence of I/O operations (instructions) to transfer data between peripheral devices and main memory and enables the central processing unit (CPU) to control the peripheral devices connected to it. Thus, I/O operations consist of two classes:

## 5.2 Synchronous vs Asynchronous I/O

Synchronous and asynchronous I/O are two separate ways of performing input/output operations in computer systems.

Synchronous I/O:

Synchronous I/O is a blocking type of I/O operation where the program waits for the I/O operation to complete before proceeding with the next task. In other words, synchronous I/O operations will pause the execution of the program until the data is available, which can cause the program to be less responsive or even unresponsive if there is a long wait time.

Asynchronous I/O:

Asynchronous I/O is a non-blocking type of I/O operation where the program continues to execute while waiting for the I/O operation to complete. This means that the program will not be paused and will continue with other tasks until the data is available. Asynchronous I/O operations can be achieved by using callbacks or events.

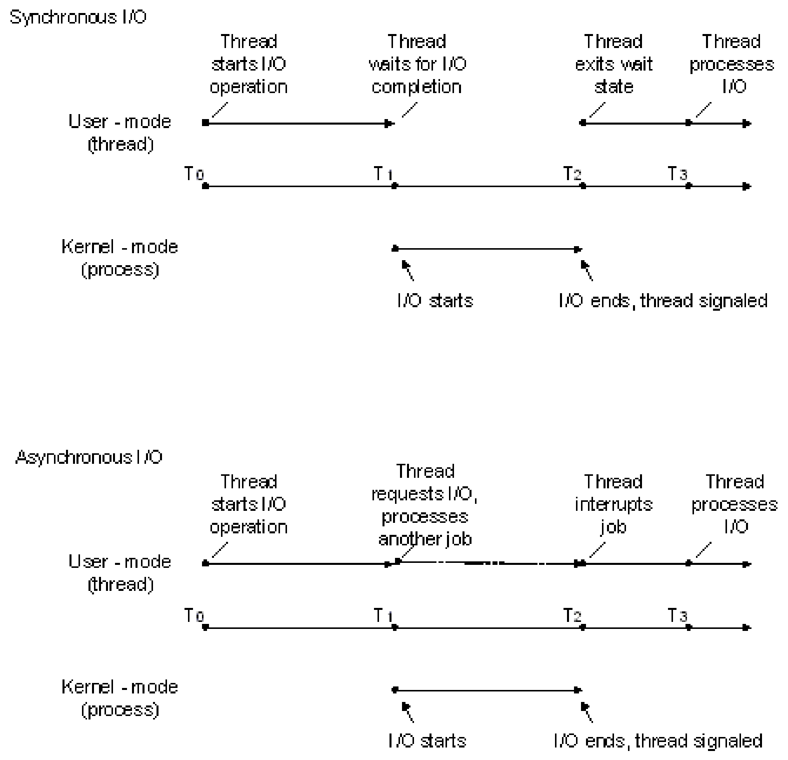

*Figure 6*

Synchronous and asynchronous I/O are two separate ways of performing input/output operations in computer systems. Here are some considerations.

Synchronous I/O

1. Blocking: Synchronous I/O operations are blocking, which means that the program waits for the operation to complete before moving on to the next line of code. This can lead to performance issues, especially in systems where I/O operations take a long time to complete.
2. Simplicity: Synchronous I/O is simpler to understand and implement compared to asynchronous I/O. It is often used in small applications or when the number of concurrent I/O operations is small.
3. Predictable: Synchronous I/O operations are predictable because they occur in a specific order. This makes debugging and error handling easier.

Asynchronous I/O

1. Non-blocking: Asynchronous I/O operations are non-blocking, which means that the program can continue to execute while the I/O operation is being performed. This improves performance because the program can do other things while waiting for I/O operations to complete.
2. Scalability: Asynchronous I/O is highly scalable because it can handle a large number of concurrent I/O operations without negatively impacting performance. It is often used in large-scale systems and high-performance applications.
3. Complexity: Asynchronous I/O is more complex to understand and implement than synchronous I/O. It requires careful management of callbacks and events and can be more difficult to debug and handle errors.

## 5.3 Blocking vs Non-Blocking I/O.

In I/O blocking, when a process requests an I/O operation, it waits until the operation is complete before continuing execution. This means that the process is blocked until the I/O operation completes. I/O blocking is a simple and straightforward approach to handling I/O operations, but if the I/O operations take a long time to complete or there are many I/O requests It can cause performance problems.

In contrast, non-blocking I/O allows a process to run while an I/O operation is being performed. Instead of waiting for the operation to complete, the process is notified when the operation completes. This notification can be done through polls or recalls. Nonblocking I/O can be more efficient than blocking I/O because it allows a process to perform other tasks while waiting for I/O operations to complete.

One disadvantage of nonblocking I/O is that it can be more complex to implement than blocking I/O because the process needs to keep track of multiple I/O requests and their statuses. Additionally, managing I/O operations to avoid blocking I/O may require more system resources such as CPU time and memory, to manage the I/O operations.

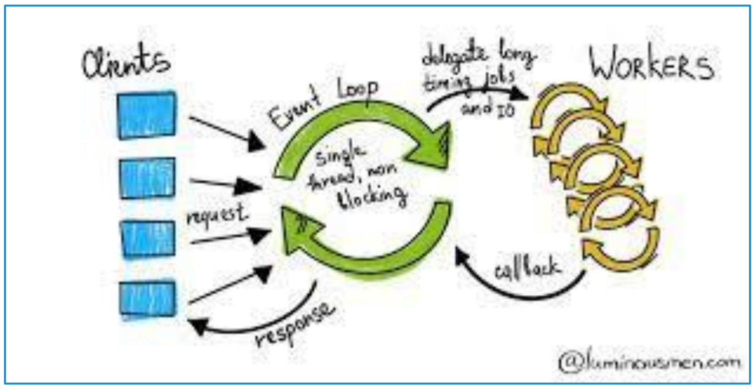

Figure 7

## 5.4 File I/O

File I/O refers to the process of reading and writing files in a computer system. File I/O is an important part of many applications, including operating systems, databases, and file-based applications such as text editors and image editors.

File I/O usually involves opening a file, reading or writing data to the file, and then closing the file. Data read from or written to the file can be in many different formats, including text, binary data, and structured data such as JSON or XML.

Programming languages have various approaches to implementing file I/O. In general, file I/O involves using functions or methods provided by the programming language or operating system to perform the required operations. For example, in the C programming language, file I/O can be performed using functions like 'fopen', 'fread' and 'fwrite'. In Python, file I/O can be performed using the built-in 'open' function and methods such as 'read' and 'write'.

When implementing file, I/O, it is important to consider factors such as performance, error handling, and security. For example, reading or writing large files can be time-consuming, so it may be necessary to use techniques such as buffering or memory mapping to improve performance. Error handling is also important to ensure the program can recover from errors like file not found or permission errors. Security is important to prevent unauthorized access to sensitive data stored in files, which may require using encryption or access controls.

## 5.5 Network I/O

Network I/O stands for Network Input/Output. It refers to the communication between a computer or a device and a network, which involves sending and receiving data through a network interface.

Network I/O can be divided into two categories:

- Network Input: It involves receiving data from the network. For example, when you browse a website, the data is sent from the web server to your computer over the network. This data is received by your computer's network interface, which is responsible for processing it and sending it to the appropriate application.

- Network Output: It involves sending data over the network. For example, when you send an email, the data is sent from your computer to the email server over the network. This data is sent by your computer's network interface, which is responsible for processing it and sending it to the appropriate network address.

Network I/O is an essential aspect of network communication, and it is critical for the proper functioning of various network applications and services, such as email, file sharing, and online gaming. The speed and efficiency of network I/O can have a significant impact on the performance of these applications and optimizing network I/O is a key consideration for network administrators and developers.

## 5.6 Memory-mapped I/O

Memory-mapped I/O (MMIO) is a mechanism for performing input/output (I/O) operations in computer systems. It enables devices to exchange data directly with the main memory of the computer without the need for a separate I/O bus, controller or buffer. In MMIO, the device registers are mapped to a region of memory, allowing the CPU to read and write to them just like any other memory location. When a device performs MMIO, it writes data directly to the memory location assigned to it, and the CPU can read this data from the same location. Similarly, the CPU can write data to the same memory location, which the device can read as input. This eliminates the need for copying data between separate memory locations, which can improve performance and reduce latency.

MMIO is used in many computer systems, including microcontrollers, embedded systems, and personal computers. It is particularly useful in systems that require fast I/O operations, such as graphics cards, network adapters, and storage controllers. By using MMIO, these devices can perform I/O operations more efficiently, without placing an unnecessary burden on the CPU.

One important consideration when using MMIO is that devices must be carefully programmed to avoid conflicts with other devices accessing the same memory regions. In addition, some operating systems and hardware platforms may have restrictions on the use of MMIO for security reasons. Nonetheless, MMIO remains an important mechanism for efficient and high-performance I/O operations in many computing environments.

## 5.7 I/O Multiplexing

I/O multiplexing is a computer system technique for efficiently handling numerous input/output (I/O) tasks at the same time. It entails monitoring and handling I/O operations across various input/output channels or sources, such as network connections, files, and devices, using a single thread of processing. Previously, a thread would be blocked while I/O operations completed, resulting in inefficient use of system resources. I/O multiplexing allows a thread to wait for many I/O operations to complete at the same time, which reduces overall blocking time and increases system performance. There are various techniques to achieving I/O multiplexing in Linux systems, such as using the select (), poll (), or epoll () system calls. These system calls allow a program to wait for numerous I/O operations to complete, and then handle them when one or more I/O operations are ready.

Assume a software wants to read data from numerous network connections at the same time. A single thread may monitor all of the connections and wait for incoming data with I/O multiplexing. As data comes on one of the connections, the thread may handle it without stopping before moving on to the next. I/O multiplexing is commonly used in network programming, when it is necessary to manage numerous network connections at the same time. It is also used in operating systems to manage numerous file and device I/O activities, which improves overall system speed and efficiency.

## 5.8 How data is translated and formatted for outputs

The process of modifying the format, structure, or values of data is known as data transformation. Data can be modified at two phases of the data pipeline in data analytics initiatives. On-premises data warehouses are typically used in an ETL (extract, transform, load) process, with data transformation serving as the middle stage. Most businesses now employ cloud-based data warehouses, which can increase computation and storage resources in seconds or minutes. The cloud platform's scalability allows enterprises to bypass preload transformations and load raw data into the data warehouse, then convert it at query time – a methodology known as ELT (extract, load, transform). Data transformation may be used in processes such as data integration, data transfer, data warehousing, and data wrangling.

Constructive data transformation, destructive data transformation, aesthetic data transformation, or structural data transformation. An organization can select from a number of ETL technologies that automate the data transformation process. Data analysts, data engineers, and data scientists also use scripting languages like Python or domain-specific languages like SQL to change data.

Data transformation has various advantages:

- Data is altered to improve its organization. Converted data may be easier to utilize for both people and computers.
- Data that has been properly prepared and verified enhances data quality and protects programs from possible pitfalls such as null values, unexpected duplication, inaccurate indexing, and incompatible formats.

Yet, there are certain obstacles to properly converting data:

- Data transformation may be costly. The price is determined by the infrastructure, software, and tools used to process data. Costs for licensing, computer resources, and recruiting appropriate employees may be included.
- Data transformation operations may be time-consuming and expensive. Transforming data in an on-premises data warehouse after it has been loaded, or altering data before it is fed into applications, might generate a computational burden that slows down other activities. Because the platform can scale up to meet demand, you can conduct the changes after loading if you utilize a cloud-based data warehouse.

How to transform data

Data transformation may improve the efficiency of analytic and business operations while also allowing for improved data-driven decision-making. Data type conversion and flattening of hierarchical data should be included in the initial phase of data transformations. These actions shape data to make it more compatible with analytics systems. Data analysts and data scientists can apply more transformations as needed as distinct levels of processing. Each processing layer should be designed to accomplish a defined set of operations that satisfy a recognized business or technical need.

Eventually, a collection of transformations can restructure data without altering its content. Casting and converting data types for compatibility, modifying dates and times using offsets and format localization, and renaming schemas, tables, and columns for clarity analytics stack are all part of this.

# 6. Vulnerabilities and Security

## 6.1. Overview of vulnerabilities and security risks in I/O subsystems.

I/O subsystems in a system is so important that it can be a reason for having vulnerabilities and security risks that can lead to data theft, unauthorized access, and unwanted attacks. The entire system will fail due to small vulnerability or a security risk because it will lose its purpose. Identifying the risks will help to mitigate them even before happening. Some common vulnerabilities are like,

- Hardware vulnerability: Hardware vulnerabilities in I/O subsystems help attackers to gai unauthorized access to a system and steal sensitive data or gain access. For example, attackers may exploit vulnerabilities in firmware or device drives to inject malware to the system.

- Malware: Malicious computer software or code that interferes with system functions to steal sensitive data.

- Injection Attacks: There are lots of Injection attack types. Normally an attacker suppling untrusted input to a program and the program interprets the input normally but as a result attacker will gain an advantage like data theft, data control and denial of service.

- Buffer overflows: It is an attack that overflows the buffers due to a software code or vulnerability that can be exploited by hackers. After the attack, the hacker can crash the system, access control loss, and make more security issues.

- Race conditions: It is a condition or a vulnerability that occurs due to multiple processors or threads access the same resource simultaneously. Exploiting this vulnerability, a hacker can do a denial-of-service attack.

- Misconfigured permissions: Giving unauthorized users access to sensitive data and resources is the nature of the vulnerability.

- Denial of service (DOS): This is a popular attack that launches on systems to flood them with request, in I/O subsystems as an example can cause the system to unresponsible or crash.

- Data theft: Stealing sensitive data like passwords, bank details, personal data can cause every system to fail integrity. Using methods like social engineering, system vulnerabilities, weak password cracks and human errors the hackers will be able to leak or gain the data.

## 6.2. Common solutions

Taking security measures for the risks that identified is the next step to prevent them. This includes using the following solutions.

- Secure code practices are practices that set guidelines like bounds checking, input validation, data sanitization least privilege, secure configuration, code reviews, error handling and scanning. Using them we can prevent hacker attacks like buffer overflow attacks, Malwares.

- Implementing Access and Authentication mechanisms is another way of maintaining and preventing the risks from happening. Define access control policies, assigning them and implementing will establish effective access control that will provide more protection.

- Use security focused I/O devices that have already implemented security mechanisms to secure integrity, confidentiality, and availability against attacks.

- Update and patch software and hardware components.

- Monitoring the system is a basic solution for all. The vulnerabilities and risks can handle and prevent happening from the early stage. Conducting regular security assignments after monitoring them will even identify vulnerabilities and risks and can use the above solutions to mitigate.

By implementing these steps and solutions, and I/O subsystem can protect sensitive data and resources and do its purpose.

# 7. Future Development

## 7.1. Emerging trends and technologies in I/O subsystems.

- PCIe Gen 4
  Peripheral Component Interconnect Express (PCIe) is a high-speed serial computer expansion bus standard that is frequently used to link hardware components to a motherboard of a computer, such graphics cards and network adapters. The most recent version of the standard, PCIe Gen 4, doubles the bandwidth of PCIe Gen 3 and enables faster data transfer rates.

- NVMe.
  Accessing solid-state drives (SSDs) through PCIe is made possible by the NVMe (Non-Volatile Memory Express) protocol. High-performance computing settings benefit greatly from the usage of NVMe-based SSDs since they provide quicker read and write speeds than conventional SATA-based SSDs.

- Thunderbolt 4
  A single cable may provide both data transfer and power delivery using the high-speed interface standard known as Thunderbolt. The most recent version of the standard, Thunderbolt 4, provides up to two 4K monitors, up to 40 Gbps of speed, and quick charging for linked devices.

- USB 4
  The most recent version of the USB (universal serial bus) standard, USB4, offers up to 40 Gbps of data transfer rate. Moreover, USB4 is backwards compatible with previous USB standards, making it an option for users to use older devices.

- RDMA
  Using a technology called remote direct memory access, two networked computers may exchange data directly from main memory without using either computer's CPU, cache, or operating system. Like locally based Direct Memory Access (DMA), RDMA boosts speed and throughput by freeing up resources, leading to quicker data transfer rates and lower latency across RDMA-enabled devices. Both networking and storage applications can benefit from RDMA.

- 5G
  The fifth generation of wireless technology, or 5G, offers more network capacity, reduced latency, and quicker data transfer rates than 4G. Wireless I/O over 5G allows for fast data transmission between devices without the requirement of physical connections.

**7.2. Potential improvements and advancements in the future.**

- Faster and more efficient storage
  For many years, storage speed has been a limiting element in I/O subsystems. The need for speedier storage grows increasingly important as applications continue to demand more data and faster access times. Emerging non-volatile memory technologies, such as 3D XPoint and MRAM, promise substantially higher storage rates than existing options. These technologies use distinct techniques to provide quicker access times, reduced latency, and better durability than standard NAND flash memory, which is typically utilized in SSDs.

- Greater Bandwidth and lower latency
  I/O performance may be enhanced through faster interconnects between storage devices and other system parts. Compared to existing solutions, newer technologies like PCIe 5.0 and 112G Ethernet promise to offer more bandwidth and less latency.

- Improve caching algorithms.
  Caching algorithms can have a significant impact on I/O performance, particularly for workloads with a high degree of locality. By keeping frequently requested data closer to the CPU or GPU, caches can assist minimize I/O latency. Caching algorithm improvements, particularly those that use machine learning and other sophisticated approaches, can significantly reduce I/O latency, and enhance throughput. Certain caching algorithms, for example, can dynamically adapt to changing workload patterns to enhance cache efficiency.

- Enhanced virtualization support
  Virtualization can increase the amount of overhead in I/O subsystems. Virtualization support improvements can help decrease this cost, making I/O performance more predictable and efficient. SR-IOV (Single Root I/O Virtualization) and NVMe-oF (NVMe over Fabrics) technologies, for example, can assist increase I/O performance in virtualized settings by allowing more direct access to storage devices.

- Greater intelligence and automation
  More intelligence and automation can help I/O subsystems. Predictive analytics and machine learning, for example, may be used to proactively manage I/O workloads, lowering latency, and increasing throughput. Similarly, automation solutions can assist administrators enhance performance by simplifying I/O subsystem administration. Certain modern storage systems, such as those based on artificial intelligence (AI), can detect and handle performance issues automatically by dynamically altering storage settings.

## 8. References.

[1] ABRAHAM SILBERSCHATZ, PETER BAER GALVIN, GREG GAGNE, Operating System Concepts, Laurie Rosatone.

[2] Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman, Linux Device Drivers, O'Reilly Media, 2005.

[3] David A. Patterson, John L. Hennessy, Computer Organization and Design: The Hardware Software Interface [RISC-V Edition], Morgan Kaufmann, 2017.

[4] J. G. David E. Williams, Virtualization With XEN, Syngress.

[5] R. B. D. A. T. S. Mindshare Inc., PCI Express System Architecture, Addison-Wesley Professional, 2003.

[6] T. Fisher, "Small Computer System Interface (SCSI)," 2022.

[7] solarwindssoftware, "What Is and How to Control Virtual Machine (VM) Sprawl".

[8] C. W. Mendel Rosenblum, "VIRTUALIZATION".

[9] w. s. a. l. brown, computer security principles and practice second edition, 2011.

[10] Remzi H. Arpaci-Dusseau, Andrea C. Arpaci-Dusseau, Operating Systems: Three Easy Pieces, Arpaci-Dusseau, 2015.