Operating System

Bachelors in Computer Engineering

Chapter five

Input/Output management

OUTLINES:

- Introduction, Interrupts Handlers
- Principles of I/O Hardware (I/O Device, Device Controller, Memory Mapped I/O, Direct Memory Access)
- Principles of I/O Software (Goals of I/O Software, Polled I/O verses Interrupt Driven I/O, Character User Interface and Graphical User Interface, Device Driver, Device Independent I/O Software, User -space I/O Software,
- System Resources: Preemptable and Non-preemtpable, Method of handling Deadlocks, Deadlock prevention,
- Deadlock avoidance: Banker's Algorithm, Deadlock detection: Resource allocation graph, Recovery from Deadlock
- Redundant Array of Inexpensive Disks
- RAM Disks

Introduction

- In a computer system there are varieties of input and output device with different speed and function.
- An input/output management module is responsible to manage and control such types of devices.
- I/O refers to the communication between information processing system, such as a computer, and the outside world possibly a human, or another information processing system.
  - Inputs are the signals or data received by the system, and
  - outputs are the signals or data sent from it.
- Input output management is also one of the primary responsibilities of an operating system.

Principle of I/O Hardware

- Different people look input/output hardware in different ways.
- Electrical engineers look it in term of chips, wires, power supplies and all other physical component that make up the hardware.
- Programmers look it as the interface presented to the software. The hardware accepts commands, carry out functions and report the result.
- The role of operating system in computer system is to manage and control I/O operation and I/O devices.

I/O Devices

- These are the devices that communicate with a computer system by sending signals over a cable.
- The devices use a common set of wires, called bus, for communication.
- An input device is a device by which the computer takes input from the outside world.
- An input device:
  - Accept data from outside world
  - Convert data in computer readable form
  - Passes the data to the CPU for further action.
- An output device is a device by which the computer gives the information to the users.
- An output device:
  - Receive data from the CPU in computer readable form.
  - Convert these information into user readable form.
  - Gives the result to the user.

I/O devices can be of two types:

Block Device:

- A device which stores information in fixed size blocks, each one with its own address is termed as block device.
- Common block size would be range from 512 bytes to 32,768 bytes.
- The main feature of this device is that it reads or writes each block independently.

Example: Disks are the most common block devices.

Character Device:

- A character device delivers or accepts a stream of characters, without regard to any block structure.
- It is not addressable and does not have any seek operation.

Example: printer, mouse, etc

Device Controllers

- An electronic device in the form of chip or circuit board that controls functioning of the I/O device is called the device controller or I/O Controller or adopter. The operating system directly deals with the device controller.
- It is the hardware that controls the communication between the system and the peripheral drive unit.
- Any device connected to the computer is connected by a plug and socket, and the socket is connected to a device controller.
- Each device controller has a local buffer and a command register. It communicates with the CPU by interrupts.
- A device's controller plays an important role in the operation of that device; it functions as a bridge between the device and the operating system.
- I-O devices generally contain two parts: mechanical and electrical part. This electrical part is known as a device controller and can take the form of a chip on personal computers and mechanical part is a device.
- It takes care of low level operations such as error checking, moving disk heads, data transfer, and location of data on the device.
- There are many device controllers in a computer system. Example: serial port controller, SCSI bus controller, disk controller

- Function of device controllers:
  - Stops and starts the activity of the peripheral device.
  - Generate error checking code.
  - Checks the error in the data received from the interface.
  - Abort that command which have errors.
  - Retry the command having an error
  - Receives the control signals from the interface unit
  - Convert the format of the data
  - Check the status of the device.

Memory Mapped I/O

- Each controller has a few I/O registers that are used for communicating with the CPU.
  - By writing into these registers, the operating system can command the device to deliver data, accept data, switch itself on or off, or otherwise perform some action. \
  - By reading from these registers, the operating system can learn what the device's state is, whether it is prepared to accept a new command, and so on.

- The system that contains I/O registers as the part of regular memory address space is known as memory mapped I/O. Each control register is assigned a unique memory address to which no memory is assigned. Usually, the assigned addresses are at the top of the address space.
- In short, the scheme that assigns specific memory location to I/O devices is known as memory mapped I/O.
  - Thus communication to and from I/O device can be same as reading and writing to memory address that responds to the I/O devices.

Direct Memory Access (DMA)

- DMA Stands for "Direct Memory Access." DMA is a method of transferring data from the computer's RAM to another part of the computer without processing it using the CPU. Most of the data that is input to or output from the computer is processed by the CPU, but some data does not require processing, or can be processed by another device. In these situations, DMA can save processing time and is a more efficient way to move data from the computer's memory to other devices.
- In other words, DMA is a method that allows an I/O device to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations. DMA is an essential feature of all modern computers, as it allows devices to transfer data without subjecting the CPU to a heavy overhead. With DMA, the CPU gets freed from this overhead and can do useful tasks during data transfer. The process is managed by a chip known as a DMA controller (DMAC). A DMA Controller is a device, which takes over the system bus to directly transfer information from one part of the system to another.
- For example,
    - a sound card may need to access data stored in the computer's RAM, but since itcan process the data itself, it may use DMA to bypass the CPU.
    - Video cards that support DMA can also access the system memory and process graphics without needing the CPU

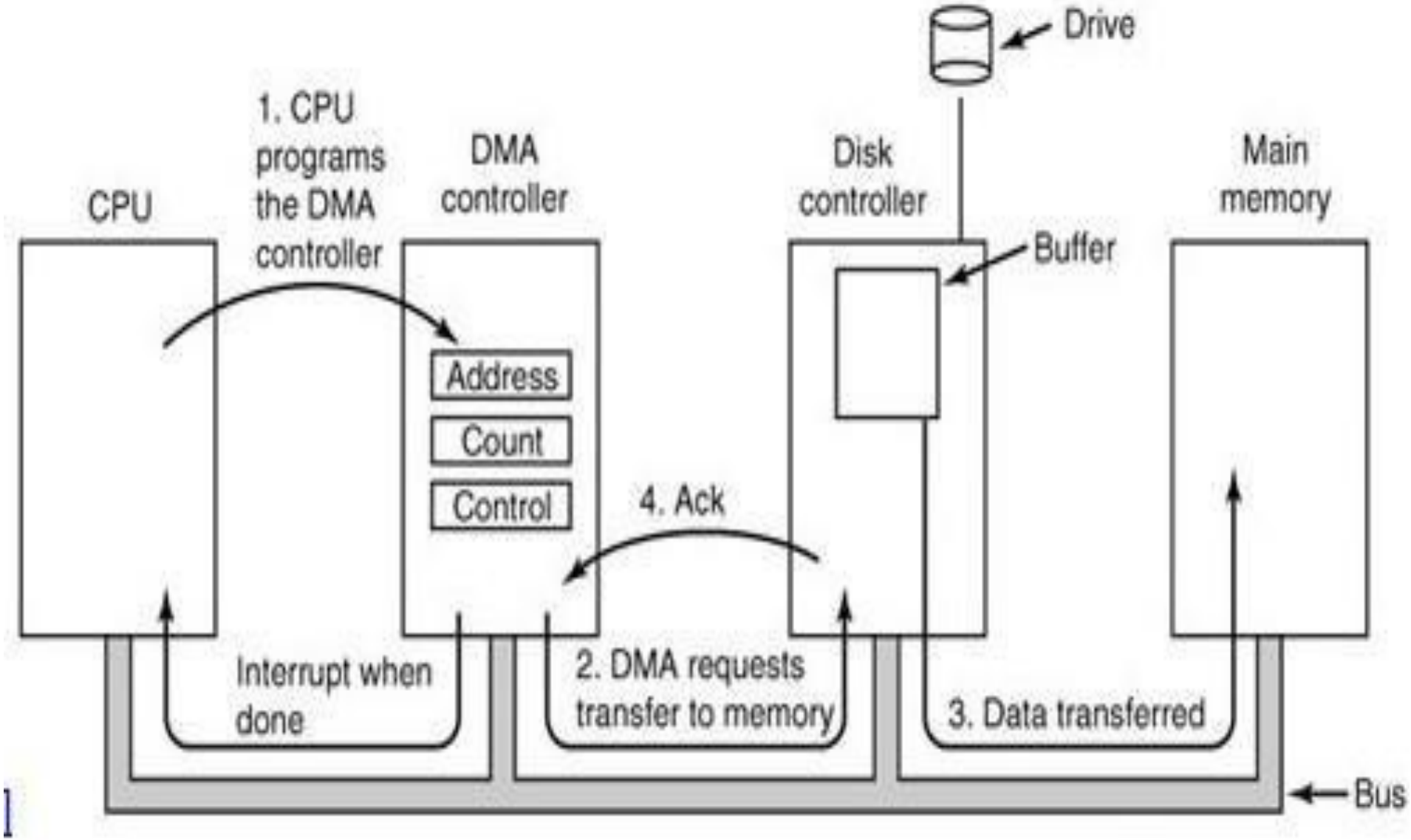


Fig: Operation of DMA transfer

- First the CPU programs the DMA controller by setting its registers so it knows what to transfer and where (step 1 in Fig.).
- The DMA controller initiates the transfer by issuing a read request over the bus to the disk controller (step 2). This read request looks like any other read request, and the disk controller does not know or care whether it came from the CPU or from a DMA controller.
- Typically, the memory address to write to is on the address lines of the bus so when the disk controller fetches the next word from its internal buffer, it knows where to write it. The write to memory is another standard bus cycle (step 3).
- When the write is complete, the disk controller sends an acknowledgement signal to the disk controller over the bus (step 4).
- The DMA controller then increments the memory address to use and decrements the byte count. If the byte count is still greater than 0, steps 2 through 4 are repeated until the count reaches 0. At this point the controller interrupts the CPU to let it know that the transfer is now complete.

- **Working of DMA in short**
  - Transfers a block of data directly to or from memory
  - An interrupt is sent when the task is complete.
  - The processor is only involved at the beginning and end of the transfer

Principle Of I/O Software

Goals of I/O Software

The main goals of I/O software are:

- Device Independence: It means that it should be possible to write programs that can access any I/O device without having to specify the device in advance. For example, a program that reads a file as input should be able to read a file on a floppy disk, on a hard disk, or on a CD-ROM, without having to modify the program for each different device.
- Uniform Naming: The name of a file or a device should simply be a string or an integer and not depend on the device in any way. The naming scheme for any device should be uniform or similar.
- Error Handling: Errors should be handled as close to the hardware as possible. If the controller discovers a read error, it should try to correct the error itself if it can. If it cannot, then the device driver should handle it.
- Buffering: Data that come off a device cannot be stored directly in its final destination. The data must be put into the buffer before storing in its final destination to increase the rate of data transfer.
- Provide a Device Independent Block Size: The sector size and block size vary from device to device. The I/O software must provide a uniform block size by treating many sectors as a single logical block.

Program I/O

- When the processor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate I/O module.
- In programmed I/O, the I/O module will perform the required action and then set the appropriate bits in the I/O status register.
- The I/O module takes no further action to alert the processor.
  - In particular, it doesn't interrupt the processor.
- After the I/O instruction is invoked, the processor must periodically check the status of the I/O module until it finds that the operation is complete.
- With this technique, the processor is responsible for extracting data from main memory for output and storing data in main memory for input.

- I/O software is written in such a way that the processor executes instructions that give it direct control of the I/O operation, including sensing device status, sending a read or write command, and transferring the data.
- Thus, the instruction set includes I/O instructions in the following categories:
  - **Control:** Used to activate an external device and tell it what to do. For example, a magnetic-tape unit may be instructed to rewind or to move forward one record.
  - **Status:** Used to test various status conditions associated with an I/O module and its peripherals.
  - **Transfer:** Used to read and/or write data between processor registers and external devices.

Interrupt Driven I/O

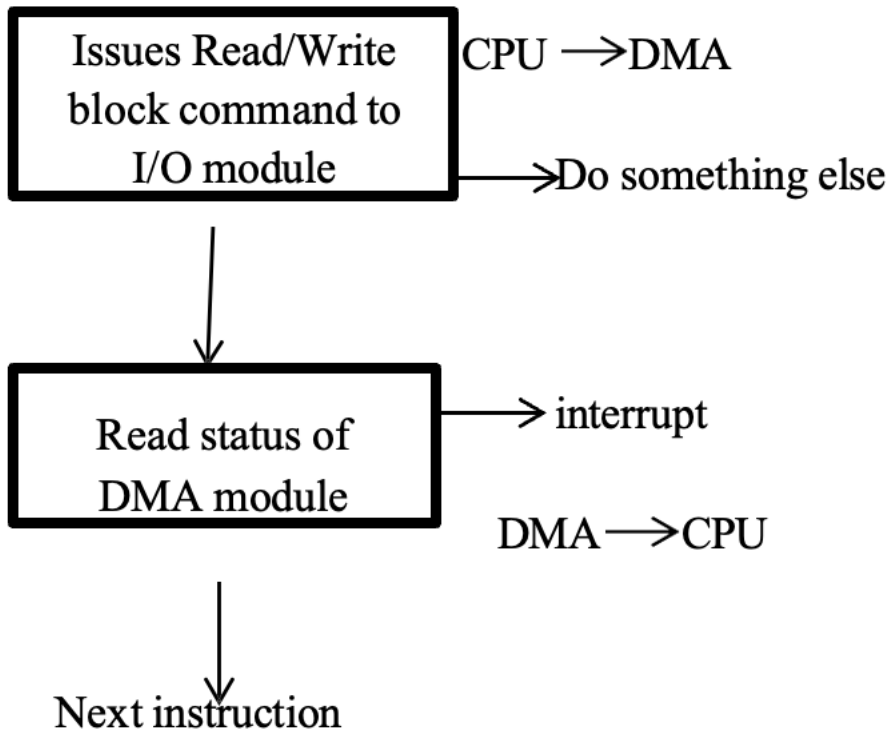
- The problem with the programmed I/O is that the processor has to wait a long time for the I/O module of concern to be ready for either reception or transmission of more data.
  - The processor, while waiting, must repeatedly interrogate the status of the I/O module.
  - As a result the performance level of entire system is degraded.
- An alternative approach for this is interrupt driven I/O.
- In this approach, the processor issue an I/O command to a module and then go on to do some other useful work.
- The I/O module will then interrupt the processor to request service when it is ready to exchange data with the processor.
- The processor then executes the data transfer and then resumes its former processing.
- Interrupt-driven I/O still consumes a lot of time because every data has to pass with processor.

- The interrupt driven I/O works as follow:
  - For input, the I/O module receives a READ command from the processor.
    - The I/O module then proceeds to read data in from an associated peripheral.
    - Once the data are in the module's data register, the module signals an interrupt to the processor over a control line.
    - The module then waits until its data are requested by the processor.
    - When the request is made, the module places its data on the data bus and is then ready for another I/O operation.
  - Sometimes there will be multiple I/O modules in a computer system, so mechanisms are needed to enable the processor to determine which device caused the interrupt and to decide, in the case of multiple interrupts, which one to handle first.

I/O Using DMA

- Interrupt-driven I/O is more efficient than simple programmed I/O but it still requires the active intervention of the processor to transfer data between memory and an I/O module, and any data transfer must traverse a path through the processor. Thus both of these forms of I/O suffer from two inherent drawbacks:
  - The I/O transfer rate is limited by the speed with which the processor can test and service a device.
  - The processor is tied up in managing an I/O transfer and a number of instructions must be executed for each I/O transfer.
- When large volumes of data are to be moved, a more efficient technique is required and this technique is by using DMA. The DMA function can be performed by a separate module on the system bus, or it can be incorporated into an I/O module. In either case, the technique works as follows:

- When the processor wishes to read or write a block of data, it issues a command to the DMA module, by sending to the DMA module the following information.
  - Whether a read or write is requested.
  - The address of the I/O devices involved.
  - The starting location in memory to read data from or write data to.
  - The number of words to be read or written.
- The processor then continues with other work. It has delegated the I/O operation to the DMA module.
  - The DMA module transfers the entire block of data, one word at a time, directly to or from memory, without going through the processor.
  - When the transfer is complete, the DMA module sends an interrupt signal to the processor.
  - Thus the processor is involved only at the beginning and at the end of the transfer.
- The DMA module needs to take control of the bus to transfer data to and from memory.
- Thus, a DMA module controls the exchange of data between main memory and an I/O module. The processor sends a request for the transfer of a block of data to the DMA module and is interrupted only after the entire block has been transferred. DMA is far more efficient than interrupt driven or programmed I/O.


Issues Read/Write block command to I/O module
CPU → DMA
Do something else
Read status of DMA module
interrupt
DMA → CPU
Next instruction

I/O Software Layers

Fig. Layers of the I/O software system and the main functions of each layer.

Interrupt Handler

- It is the bottom layer of I/O software.
- When an event occurs the micro-controller generates a hardware interrupt.
- The interrupt forces the micro-controller's program counter to jump to a specific address in program memory. This special memory address is called the interrupt vector.
- At this memory location we install a special function known as an interrupt service routine (ISR) which is also known as an interrupt handler.
- So when a hardware interrupt is generated, program execution jumps to the interrupt handler and executes the code in that handler. When the handler is done, then program control returns the micro-controller to the original program it was executing.
- So a hardware interrupt allows a micro-controller to interrupt an existing program and react to some external hardware event.

Device Drivers

• A driver is a small piece of software that tells the operating system and other software how to communicate with a piece of hardware.

• A device driver provides a software interface to hardware devices, enabling operating systems and other computer programs to access hardware functions without needing to know precise details of the hardware being used.

• A device driver typically communicates with the device through the computer bus or communications subsystem to which the hardware connects.

• Device drivers are hardware-dependent and operating system specific.

• Each device controller has registers used to give it commands or to read out its status or both.

• The number of registers and the nature of the commands vary radically from device to device.

• This code, called the device driver, is generally written by the device's manufacturer and delivered along with the device on a CD-ROM.

• Since each operating system needs its own drivers, device manufacturers commonly supply drivers for several popular operating systems.

Device Independent I/O Software

- **Device independence** is the process of making a software application to be able to function on a wide variety of devices i.e. the capability of a program, operating system or programming language to work on varieties of computers or peripherals, despite their electronic variation.
- The basic function of the device-independent software is to perform the I/O functions that are common to all devices and to provide a uniform interface to the user-level software.
- It also helps in buffering, error reporting, allocating and releasing dedicated devices and providing a device- independent block size.

User Space I/O Software

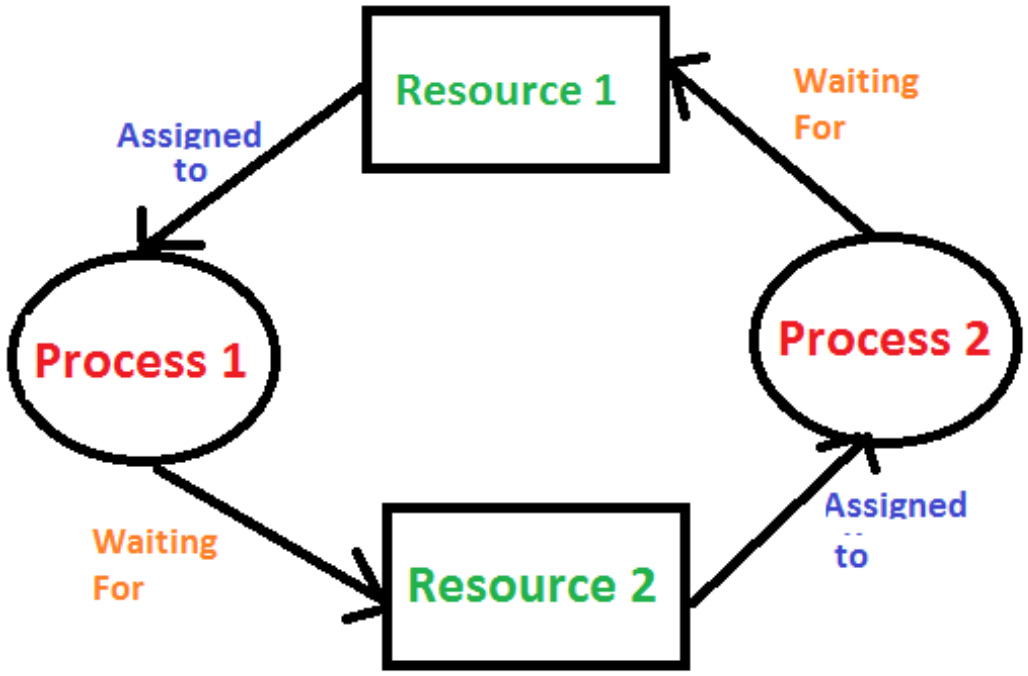
- It is the upper most layer of I/O system.
- This layer mainly represents the outermost user interface that occurs between the operating system and end user.
- A user interface is that portion of an interactive computer system that communicates with the user.
- Proper design of a user interface can make a significant difference in training time, performance speed, error rates, user satisfaction, and the user's retention of knowledge of operations over time.

System Resources: Preemptable and non-preemptable resources

- Resources come in two types
  - **Preemptable**, meaning that the resource can be taken away from its current owner (and given back later).
    - An example is memory.
  - **Non-preemptable**, meaning that the resource cannot be taken away.
    - An example is a printer.

Deadlock

- Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.
- Since all the processes are waiting, none of them will ever cause any of the events that would wake up any of the other members of the set & all the processes continue to wait forever.



Example of deadlock

- Two processes A and B each want to record a scanned document on a CD.
- A requests permission to use Scanner and is granted.
- B is programmed differently and requests the CD recorder first and is also granted.
- Now, A asks for the CD recorder, but the request is denied until B releases it. Unfortunately, instead of releasing the CD recorder B asks for Scanner. At this point both processes are blocked and will remain so forever. This situation is called Deadlock.

Condition of Deadlock arises:

Mutual Exclusion –

- At least one resource must be kept in a non-shareable state; if another process requests it, it must wait for it to be released.

Hold and Wait –

- A process must hold at least one resource while also waiting for at least one resource that another process is currently holding.

No preemption –

- Once a process holds a resource (i.e. after its request is granted), that resource cannot be taken away from that process until the process voluntarily releases it.

Circular Wait –

- Circular wait occurs when a process holds a resource while waiting for another resource that is held by a different process in the system. This creates a circular dependency among the processes and their requested resources, forming a cycle.
- There exists a set {P1 , ..., Pn } of waiting processes. P1 is waiting for a resource that is held by P2. P2 is waiting for a resource that is held by P3... Pn is waiting for a resource that is held by P1.

Deadlock Modeling

- A deadlock occurs when a set of processes is stalled because each process is holding a resource and waiting for another process to acquire another resource.
- for example: Process 1 is holding Resource 1 while Process 2 acquires Resource 2, and Process 2 is waiting for Resource 1.
- Deadlocks can be described more precisely in terms of Resource allocation graph. It's a set of vertices V and a set of edges E. V is partitioned into two types:

  P = {P1, P 2, ..., P n}, the set consisting of all the processes in the system.

  R = {R1, R2 , ..., Rm}, the set consisting of all resource types in the system.

  request edge – directed edge Pi→ Rj

  assignment edge – directed edge Rj → Pi

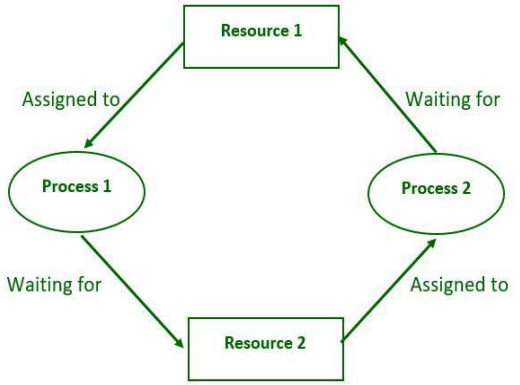


Figure: Deadlock in Operating system

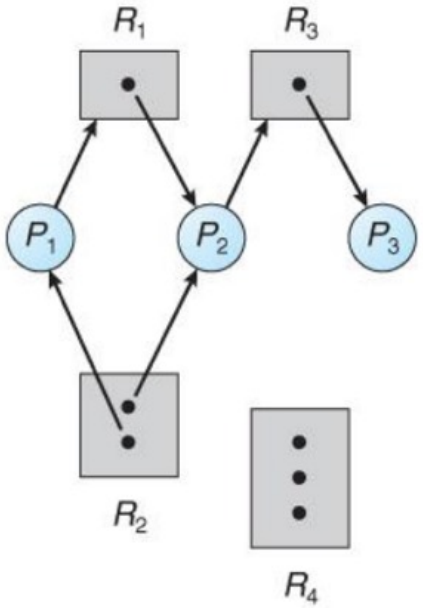


Figure a

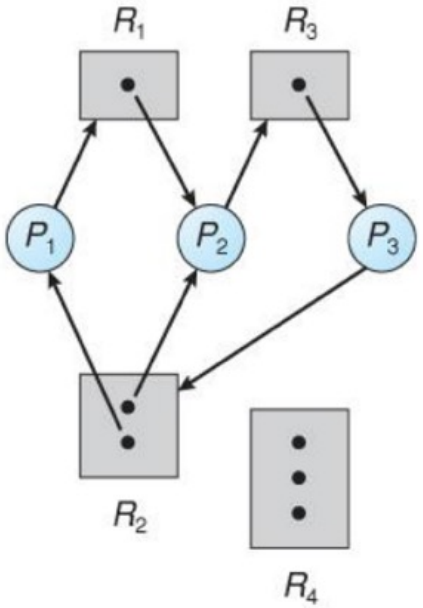


Figure b

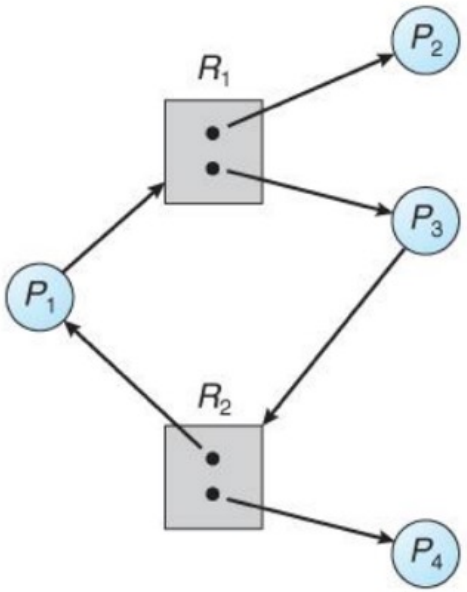


Figure c

- Fig a: Resource Allocation Graph
- Fig b: Resource Allocation graph with deadlock
- Fig c: Resource Allocation graph with a cycle but no deadlock

Basic Facts:

If graph contains no cycles ⇒ no deadlock.

If graph contains a cycle ⇒

- If only one instance per resource type, then deadlock.
- If several instances per resource type, possibility of Deadlock

Methods for Handling Deadlock:

i. ***Detection and recovery:*** *Allow system to enter deadlock and then recover*

- *It requires deadlock detection algorithm*
- *It must ensure some technique for forcibly preempting resources and/or terminating tasks which ensure that system will never enter a deadlock*
- *It need to monitor all lock acquisitions*
- *Selectively deny those that might lead to deadlock*

ii. Deadlock Prevention: *Ignore the problem and pretend that deadlocks never occur in the system*

- *Used by most operating systems, including UNIX*

Deadlock Detection:

- To prevent the system from deadlocks, one of the four discussed conditions that may create a deadlock should be discarded. The methods for those conditions are as follows:

I. Mutual Exclusion:

- In general, we do not have systems with all resources being sharable. Some resources like printers, processing units are non-sharable. So it is not possible to prevent deadlocks by denying mutual exclusion.

II. Hold and Wait:

- One protocol to ensure that hold-and-wait condition never occurs says each process must request and get all of its resources before it begins execution. Another protocol is “Each process can request resources only when it does not occupy any resources.”
- The second protocol is better. However, both protocols cause low resource utilization and starvation.
- Many resources are allocated but most of them are unused for a long period of time. A process that requests several commonly used resources causes many others to wait indefinitely.

III. No Preemption:

- One protocol is “If a process that is holding some resources requests another resource and that resource cannot be allocated to it, then it must release all resources that are currently allocated to it”. Another protocol is “When a process requests some resources, if they are available, allocate them. If a resource it requested is not available, then we check whether it is being used or it is allocated to some other process waiting for other resources. If that resource is not being used, then the OS preempts it from the waiting process and allocate it to the requesting process. If that resource is used, the requesting process must wait.” This protocol can be applied to resources whose states can easily be saved and restored (registers, memory space). It cannot be applied to resources like printers.

IV. Circular Wait:

- One protocol to ensure that the circular wait condition never holds is “Impose a linear ordering of all resource types.” Then, each process can only request resources in an increasing order of priority. For example, set priorities for r1 = 1, r2 = 2, r3 = 3, and r4 = 4. With these priorities, if process P wants to use r1 and r3, it should first request r1, then r3. Another protocol is “Whenever a process requests a resource rj, it must have released all resources rk with priority(rk) ≥ priority(rj).

Deadlock Avoidance:

- Given some additional information on how each process will request resources, it is possible to construct an algorithm that will avoid deadlock states.
- The algorithm will dynamically examine the resource allocation operations to ensure that there won't be a circular wait on resources.
- One of the deadlock avoidance algorithms is Banker's algorithm.

Bankers Algorithms:

- A resource allocation and deadlock avoidance algorithm developed by Edsger Dijkstra.
- Resource allocation state is defined by the number of available and allocated resources and the maximum demand of the processes.
- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.

The Banker algorithm does the simple task

– If granting the request leads to an unsafe state the request is denied.

– If granting the request leads to safe state the request is carried out.

Basic Facts:

- If a system is in safe state ⇒ no deadlocks.
- If a system is in unsafe state ⇒ possibility of deadlock.
- Avoidance ⇒ ensure that a system will never enter an unsafe state.

- A state is said to be **safe** if there is some scheduling order in which every process can run to completion even if all of them suddenly request their maximum number of resources immediately.
- A state is safe if the system can allocate resources to each process in some order avoiding a deadlock.
- If the system is in a safe state, there can be no deadlock.
- If the system is in an **unsafe** state, there is the possibility of deadlock. A deadlock state is an unsafe state.

Bankers Algorithms for a single resource:

Customer = Processes

Units = Resource say tape drive

Bankers = OS

Available units: 10

Customer	Used	Max
A	0	6
B	0	5
C	0	4
D	0	7

- In the above fig, we see four customers each of whom has been granted a certain no. of credit units (eg. 1 unit=1K dollar).
- The Banker reserved only 10 units rather than 22 units to service them. since not all customers need their maximum credit immediately.
- At a certain moment the situation becomes:

Customer	Used	Max
A	1	6
B	1	5
C	2	4
D	4	7

Available units: 2

Safe State:

- With 2 units left, the banker can delay any requests except C's, thus letting C finish and release all four of his resources.
- With four in hand, the banker can let either D or B have the necessary units & so on.

Unsafe State:

Customer	Used	Max
A	1	6
B	2	5
C	2	4
D	4	7

Available units: 1

- Suppose B requests one more unit and is granted. This is an unsafe condition.
- If all of the customers namely A, B, C & D asked for their maximum loans, then Banker couldn't satisfy any of them and we would have deadlock.
- It is important to note that an unsafe state does not imply the existence or even eventual existence of a deadlock.
- What an unsafe state does imply is that some unfortunate sequence of events might lead a deadlock.

Detection and Recovery

- It is a technique for handling deadlock. When this technique is used, the system does not attempt to prevent deadlocks from occurring. Instead, it lets them occur, tries to detect when this happens and then takes some action to recover after the fact.
- Once the deadlock has been detected, it can be recovered through various ways:

a. Recovery through preemption: In this method, a resource can be temporarily taken away from its current owner and given to another process. The ability to take a resource away from a process, have another process use it and then give it back without the process noticing it, is highly dependent on the nature of the resource.

b. Recovery through rollback: When a deadlock is detected, it is easy to see which resources are needed. To do the recovery, a process that owns a needed resource is rolled back to an earlier point when it did not have the resource. The resource can now be assigned to one of the deadlocked processes.

c. Recovery through killing processes: The simplest way to break a deadlock is to kill one or more processes. Every time a resource is requested or released, the resource graph is updated, and a check is made to see if any cycles exist. If a cycle exists, one of the processes in the cycle is killed. If this does not break the deadlock, another process is killed, and so on until the cycle is broken.

The Ostrich Algorithm

- The simplest approach for dealing with deadlocks is the ostrich algorithm.
- It just ignores the problem and thinks if we ignore it, it will ignore us.
- It makes the simple analogy with the ostrich.
- The ostrich algorithm means that the deadlock is simply ignored and it is assumed that it will never occur.
  - Pretend (imagine) that there's no problem.
  - This is the easiest way to deal with problem.
  - This algorithm says that stick your head in the sand and pretend (imagine) that there is no problem at all.
  - This strategy suggests to ignore the deadlock because deadlocks occur rarely, but system crashes due to hardware failures, compiler errors, and operating system bugs frequently, then not to pay a large penalty in performance or convenience to eliminate deadlocks.
- Most operating systems, including UNIX, MINIX 3, and Windows, just ignore the problem on the assumption that most users would prefer an occasional deadlock to a rule restricting all users to one process, one open file, and one of everything. If deadlocks could be eliminated for free, there would not be much discussion. The problem is that the price is high, mostly in terms of putting inconvenient restrictions on processes. Thus we are faced with an unpleasant trade-off between convenience and correctness, and a great deal of discussion about which is more important, and to whom. Under these conditions, general solutions are hard to find.

Disk

Disk Hardware

- A hard disk is part of a computer system that stores and provides relatively quick access to large amounts of data i.e. a disk drive is a randomly addressable and rewritable storage device.
- A hard disk is really a set of stacked "disks," each of which has data recorded electromagnetically in concentric circles or "tracks" on the disk.
- A "head" records (writes) or reads the information on the tracks.
- Two heads, one on each side of a disk, read or write the data as the disk spins.
- Each read or write operation requires that data be located, which is an operation called a "**seek**."

Technical terms related to Disk Drive:

- **Platter**: Hard drives are normally composed of multiple disks called platters. These platters are stacked on top of each other. It is the platter that actually stores the data.
- **Spindle/Motor:** The platters are attached at the center to a rod or pin called a spindle that is directly attached to the shaft of the motor that controls the speed of rotation.
- **Head-Actuator Assembly:** This assembly consists of an actuator, the arms, the sliders and the read/write heads. The actuator is the device that moves the arms containing read/write heads across the platter surface in order to store and retrieve information. The head arms move between the platters to access and store data. At the end of each arm is a head slider, which consists of a block of material that holds the head. The read/write heads convert the electronic 0s and 1s in the magnetic fields on the disks.

Logic Board: Logic boards consisting of chips, memory and other components control the disk speed and direct the actuator in all its movements. It also performs the process of transferring data from the computer to the magnetic fields on the disk.

Tracks: A track is a concentric ring on the disk where data is stored.

Cylinders: On drives that contain multiple platters, all the tracks on all the platters that are at the same distance from the center are referred to as a cylinder. The data from all the tracks in the cylinder can be read by simply switching between the different heads, which is much faster than physically moving the head between the different tracks on a single disk.

Sectors: Tracks are further broken down into sectors, which are the smallest units of storage on a disk. A larger number of sectors are recorded on the outer tracks, and progressively fewer toward the center. Data can also be read faster from the outer tracks and sectors than the inner ones.

Clusters: Sectors are grouped together into clusters. All the sectors in one cluster are taken as a single unit.

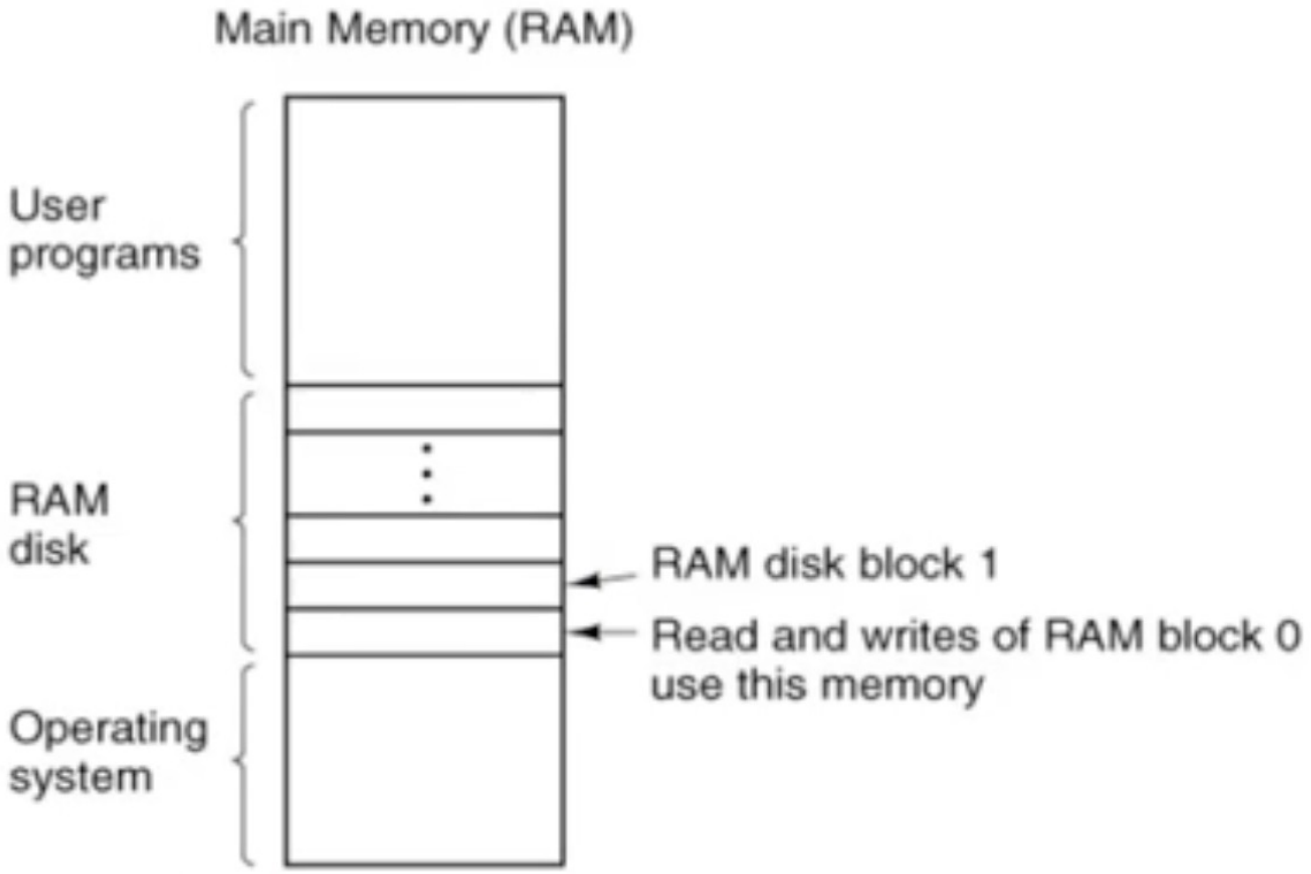
Extents: A set of contiguous clusters storing a single file or part of a file is called an extent. Reducing the number of extents in a file is achieved using a process known as defragmentation, which improves performance.

Bandwidth: The disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

RAM Disk

- A RAM disk or RAM drive is a block of RAM that a computer's software is treating as if the memory were a disk drive(Secondary storage).
- It is sometimes referred as a virtual RAM drive or software RAM drive to distinguish it from a hardware RAM drive that uses separate hardware containing RAM.
- Unlike traditional storage devices such as hard drives or solid-state drives (SSDs), which store data persistently even when the power is turned off, RAM disks are volatile and lose their data when the computer is shut down or restarted.

Idea behind RAM Disk

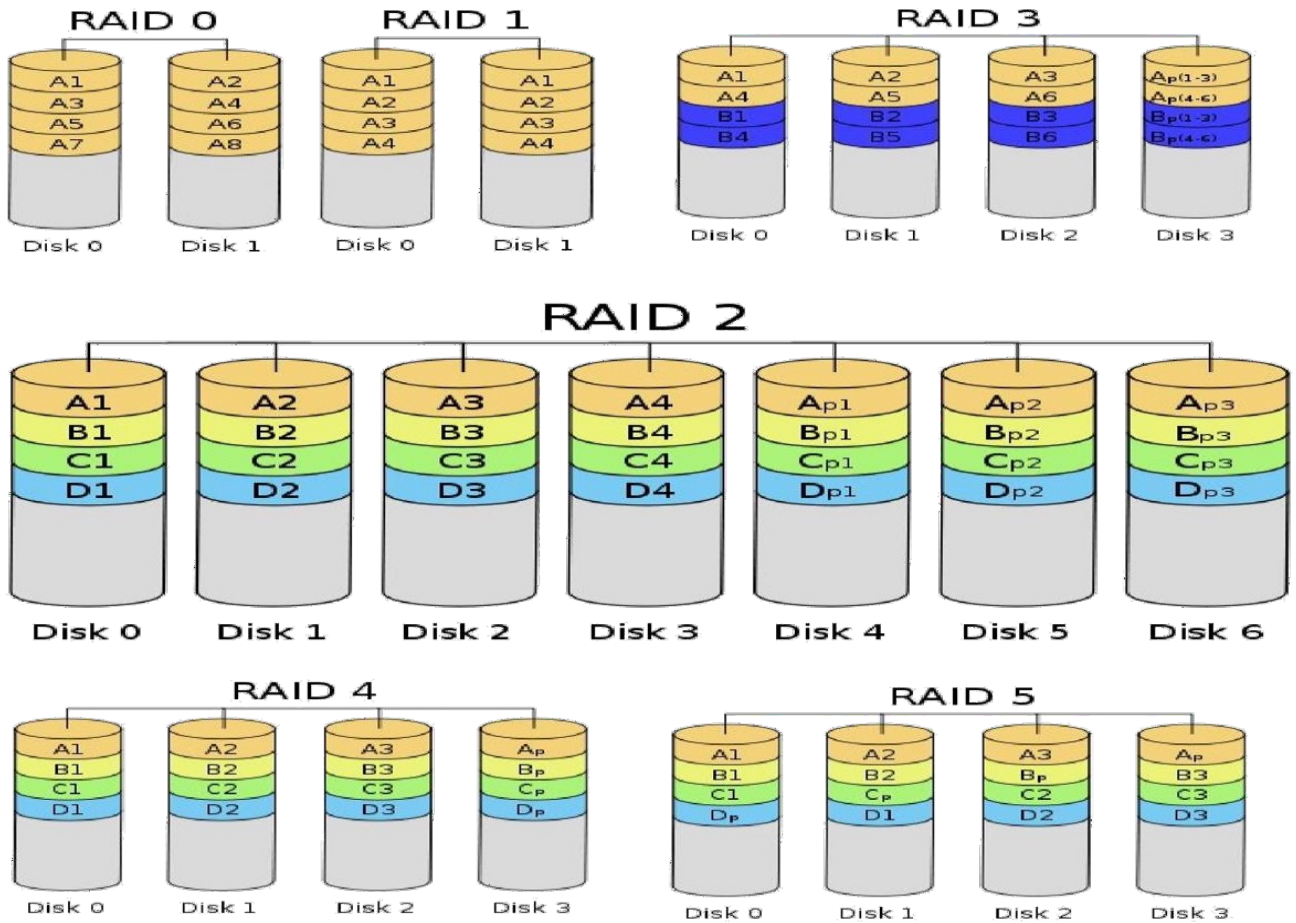


- The RAM Disk is split up into n blocks, depending on how much memory has been allocated for it. Each block is of the same size as the block size used on the real disks.
- When the driver recieves a message to read or write a block, it just computes where in the RAM disk memory the requested block lies and reads from it or writes to it, instead of from or to a floppy or hard disk.
- Users can create a RAM disk by allocating a portion of their system's RAM through OS tools or third-party software.

- The performance of a RAM disk is in general, orders of magnitude faster than other forms of storage media. This performance gain is due to multiple factors, including access time, maximum throughput and file system as well as others.
- Because the storage is in RAM, it is volatile memory.
- In many cases, the data stored on the RAM disk is created, for faster access, from data permanently stored else where, and is re-created on the RAM disk when the system reboots.
- Best suited for scenarios where ultra-fast data access is crucial, but not suitable for long-term storage of critical data.

RAID

- **RAID** (**redundant array of independent disks**, originally **redundant array of inexpensive disks)** is a storage technology that combines multiple disk drive components into a logical unit.
- Data is distributed across the drives in one of several ways called "RAID levels", depending on what level of redundancy and performance (via parallel communication) is required.


RAID 0
A1
A3
A5
A7
A2
A4
A6
A8
Disk 0
Disk 1
RAID 1
A1
A2
A3
A4
A1
A2
A3
A4
Disk 0
Disk 1
RAID 3
A1
A4
B1
B4
A2
A5
B2
B5
A3
A6
B3
B6
Ap(1-3)
Ap(4-6)
Bp(1-3)
Bp(4-6)
Disk 0
Disk 1
Disk 2
Disk 3
RAID 2
A1
B1
C1
D1
A2
B2
C2
D2
A3
B3
C3
D3
A4
B4
C4
D4
Ap1
Bp1
Cp1
Dp1
Ap2
Bp2
Cp2
Dp2
Ap3
Bp3
Cp3
Dp3
Disk 0
Disk 1
Disk 2
Disk 3
Disk 4
Disk 5
Disk 6
RAID 4
A1
B1
C1
D1
A2
B2
C2
D2
A3
B3
C3
D3
Ap
Bp
Cp
Dp
Disk 0
Disk 1
Disk 2
Disk 3
RAID 5
A1
B1
C1
Dp
A2
B2
Cp
D1
A3
Bp
C2
D2
Ap
B3
C3
D3
Disk 0
Disk 1
Disk 2
Disk 3

RAID Levels:

- RAID 0 (block-level striping without parity or mirroring) has no (or zero) redundancy.
- It provides improved performance and additional storage but no fault tolerance. Hence simple stripe sets are normally referred to as RAID 0.
- Any drive failure destroys the array, and the likelihood of failure increases with more drives in the array.
- A single drive failure destroys the entire array because when data is written to a RAID 0 volume, the data is broken into fragments called blocks. The number of blocks is dictated by the stripe size, which is a configuration parameter of the array.
- The blocks are written to their respective drives simultaneously on the same sector. This allows smaller sections of the entire chunk of data to be read off each drive in parallel, increasing bandwidth.
- RAID 0 does not implement error checking, so any error is uncorrectable.
- More drives in the array means higher bandwidth, but greater risk of data loss.

- In **RAID 1** (mirroring without parity or striping), data is written identically to multiple drives, thereby producing a "mirrored set"; at least 2 drives are required to constitute such an array.
- While more constituent drives may be employed, many implementations deal with a maximum of only 2;
- The array continues to operate as long as at least one drive is functioning.
- With appropriate operating system support, there can be increased read performance, and only a minimal write performance reduction; implementing RAID 1 with a separate controller for each drive in order to perform simultaneous reads (and writes) is sometimes called multiplexing (or duplexing when there are only 2 drives).

- In **RAID 2**, all disk spindle rotation is synchronized, and data is striped such that each sequential bit is on a different drive. Hamming-code parity is calculated across corresponding bits and stored on at least one parity drive.
- In **RAID 3** (byte-level striping with dedicated parity), all disk spindle rotation is synchronized, and data is striped so each sequential byte is on a different drive. Parity is calculated across corresponding bytes and stored on a dedicated parity drive.

- **RAID 4** (block-level striping with dedicated parity) is identical to RAID 5 (see below), but confines all parity data to a single drive. In this setup, files may be distributed between multiple drives. Each drive operates independently, allowing I/O requests to be performed in parallel. However, the use of a dedicated parity drive could create a performance bottleneck; because the parity data must be written to a single, dedicated parity drive for each block of non-parity data, the overall write performance may depend a great deal on the performance of this parity drive.

- **RAID 5** (block-level striping with distributed parity) distributes parity along with the data and requires all drives but one to be present to operate; the array is not destroyed by a single drive failure. Upon drive failure, any subsequent reads can be calculated from the distributed parity such that the drive failure is masked from the end user. However, a single drive failure results in reduced performance of the entire array until the failed drive has been replaced and the associated data rebuilt. Additionally, there is the potentially disastrous RAID 5 write hole. RAID 5 requires at least 3 disks.

RAID 6 (block-level striping with double distributed parity) provides fault tolerance of two drive failures; the array continues to operate with up to two failed drives. This makes larger RAID groups more practical, especially for high-availability systems. This becomes increasingly important as large-capacity drives lengthen the time needed to recover from the failure of a single drive. Single-parity RAID levels are as vulnerable to data loss as a RAID 0 array until the failed drive is replaced and its data rebuilt; the larger the drive, the longer the rebuild takes. Double parity gives additional time to rebuild the array without the data being at risk if a single additional drive fails before the rebuild is complete.

Thank You