Unit 1 (History of the Operating Systems)

What is OS?

1: An operating system is a program that works as an interface between the user of a computer and the computer hardware.
2: The purpose of an operating system is to provide an environment in which a user can execute programs in a convenient and efficient manner.
3: In simple terms, an OS is a program that controls all the computer resources/hardware and provides a platform in which a user can execute/run programs.
4: The OS helps you to communicate with the computer without knowing how to speak the computer's language.
5: It is not possible for the user to use any computer or mobile device without having an operating system.

Features of OS

1:Convenience: An operating system makes a computer more simple or convenient to use by hiding the hardware details of the computer.

2:Memory Management: The operating system manages memory. It has complete knowledge of primary memory.

3:Security and Reliability: It is very reliable because any virus and harmful code can be detected in it with the help of proper security enforcing techniques.

4:Process Management: The OS manages processes, which are programs in execution. It allocates resources, such as CPU time and memory, to different processes, schedules their execution, and handles process synchronization and communication.

Applications of OS

1: Embedded systems (Mostly use in home appliances)

2:Automobile engine controllers system.

3: Industrial robots, Medical Robots and Research.

4: Spacecraft Control System.

5: Industrial Control.

6: Large-scale computing systems such as green computing, mobile computing, soft computing etc.

7:Diagnostic mode of a computer operating system (OS).

8: Real time systems are used in: Airlines reservation system, Air traffic control system, Defense application systems like RADAR, Networked Multimedia

Systems, and Command Control Systems.

CP/M

1:CP/M stands for “Central Program for microcomputers" was almost the first OS developed for microcomputer platform based on Intel 8080 machine in 1974 by Gary Kildall.

2:Initially it was developed only as a file system to support all the file related operations.

3:CP/M was originally designed for 8-bit microprocessors, but later versions were adapted to support 16-bit processors as well, such as the Intel 8086.

4:CP/M was designed to support multiple programming languages, which contributed to its popularity among early microcomputer developers.

Proprietary Operating System

1:Proprietary term describes something owned by a specific individual or company.

2:Usually proprietary software like operating system is ―closed-sourced i.e. It‘s not open source, freely available or freely licensed.

3: It is designed, customized, developed and sell by only one company.

4:For example-MAC OS X operating system for apple machines developed by Apple Company which is strictly allowed to run on apple hardware.

5:Some of the features of proprietary OS are simplified user experience; complex graphics, limited customizability, interoperability, development and resource support from manufacturer etc.

6: These types of OS are usually costly as compared to open source OS.

DOS/MS-DOS

A company called ―Seattle Computer developed an OS called QDOS for Intel 8086 m/c. Later, Microsoft Corporation acquired rights to it and developed MS-DOS (Microsoft Disk Operating

system) OS.

The main features of MS-DOS are:

1: It is a character user interface (CUI) based OS.

2:It is a command line operating system.

3:It supports single-user, single-tasking.

4:It is a 16-bit OS.

5:Ms-DOS became popular because of features like user-friendliness, faster disk operations, compiler support for various high-level languages like BASIC, COBOL and C language and networking capabilities like LAN.

6: Later it adopted GUI (Graphical User Interface) and Windows (MS Windows) was developed.

UNIX, LINUX

1:UNIX was initially developed as a single user operating system.

2:It was developed much before Microsoft developed DOS.

3:The design of DOS is based on the features of the UNIX operating system.

4:It is popular because of features like multiple user support simultaneously.

5:It is a CUI based OS that supports multiprogramming, time-sharing and multitasking.

6: It‘s written in high level language; the code developed in UNIX can be easily modified and compiled on any system even though the hardware is different hence portability and machine independence are the important features of UNIX.

7: Later LINUX, an open sourced OS was developed based on features of UNIX.

8: It doesn‘t need license and it follows open development model which allows any programmers to access the source code and modify operating system.

A command line interface (CLI)

1: A command line interface (CLI)is a text-based user interface (UI) used to view and manage computer files.

2: Before the mouse, users interacted with an operating system or application with a keyboard. Users typed commands in the command line interface to run tasks or programs on a computer.

3:The user responds to a prompt/text in the command line interface by typing a command.

4:The output or response from the system can include a message, table, list, or some other confirmation of a system or application action.

5: The software that handles the command line interface is called command language interpreter.

Client-Server

1: It is an operating system that operates within the desktop. It is used to obtain services from a server.

2:It runs on the client devices like laptops, computers and is a very simple operating system.

3: It is an operating system that is designed to be used on server.

4: It is used to provide services to multiple clients.

5: It can serve multiple clients at a time and is a very advanced operating system.

6: Client/Server communication involves two components, namely a client and a server.

7: They are usually multiple clients in communication with a single server.

8: The clients send requests to the server and the server responds to the client requests.

Batch operating system

1: - Computer systems in the past were limited to performing one task at a time in a serial manner.

2: Development and preparation of programs were slow, involving numerous manual operations.

-3: Batch processing was popular in the 1970s.

4: Jobs were executed in batches on mainframe computers.

5: Multiple users submitted their jobs to the system.

6: Jobs were processed in a first-come-first-served queue.

7: Batch operating systems combined programs and data before processing.

8: Memory management was divided into operating system and user program areas.

9: Scheduling followed the order of submission.

10: After job completion, the output was copied to an output spool for later printing, similar to a buffer queue.

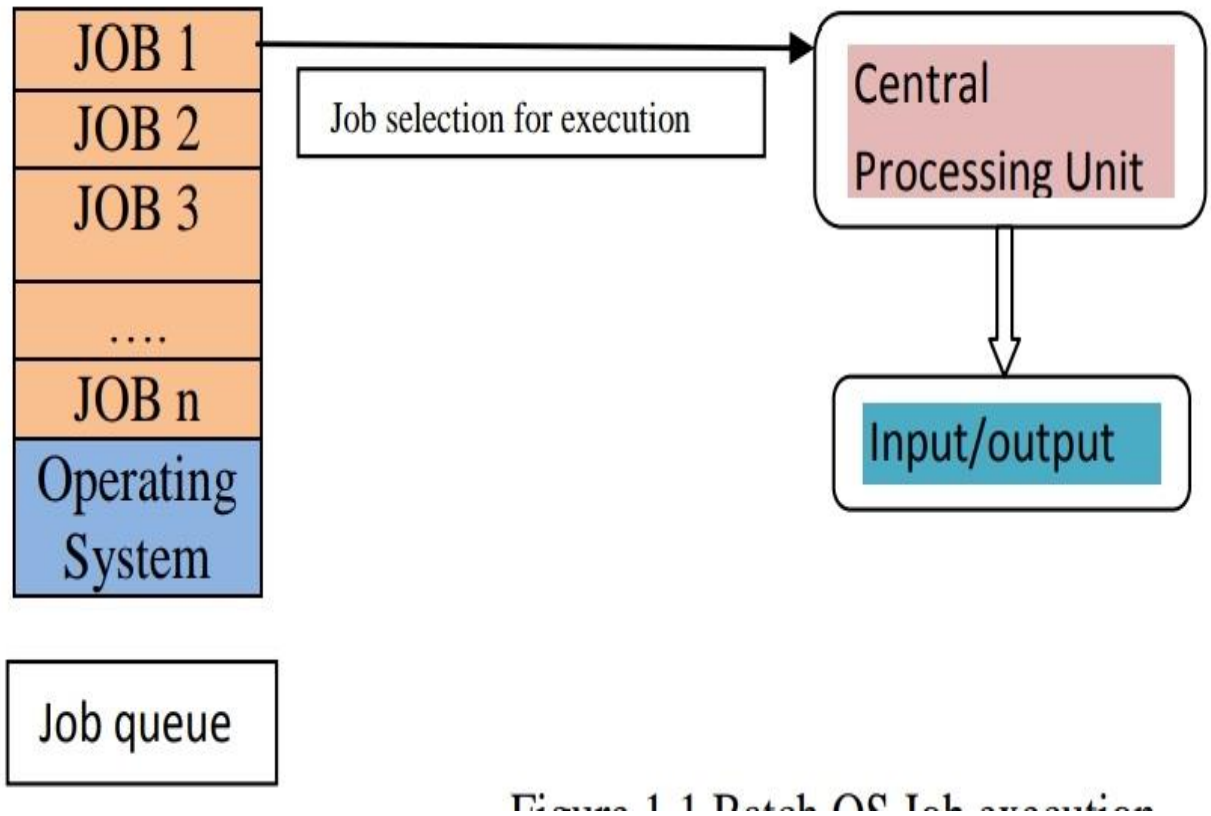


Figure 1.1 Batch OS Job execution

Advantages of batch system

1: Move much of the work of the operator to the computer.

2: Increased performance since it was possible for job to start as soon as the previous job is finished.

Disadvantages of Batch OS

1: Starvation.

2: Not Interactive

3: Difficult to debug program

4: A job could enter an infinite loop

5: A job could corrupt the monitor, thus affecting pending jobs

6: Due to lack of protection scheme, one batch job can affect pending.

Multiprogramming Operating System

1: Multiprogramming extends batch processing to keep the CPU constantly busy.

2: Processes in a multiprogramming environment require CPU time and IO time.

3: While a process performs I/O, the CPU can execute other processes simultaneously.

4: This improves the overall efficiency of the system.

5: In a multiprogramming operating system, multiple programs share a single processor.

6: CPU utilization is optimized by ensuring there is always a job to execute.

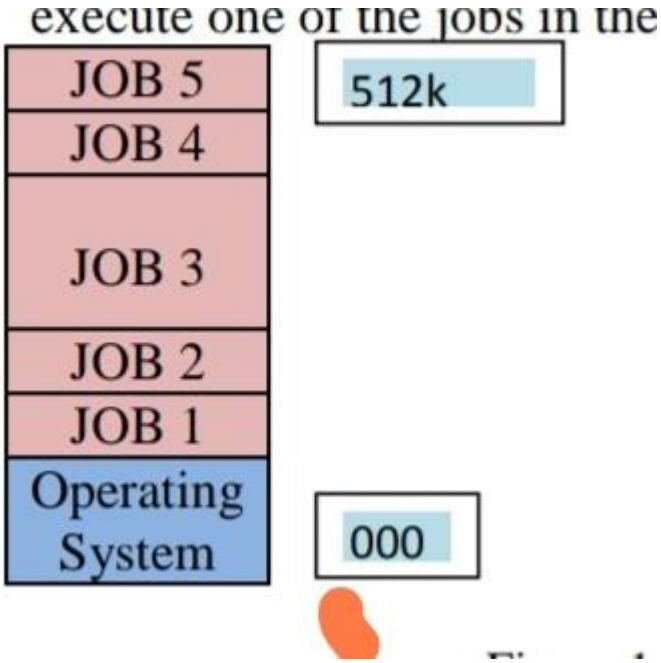
7: However, user interaction with the computer system is not provided.

8: The operating system keeps several jobs in memory from the job pool.

- It selects and begins to execute one of the jobs in memory.

9: Memory management is essential for handling multiple programs in memory.

10: The multiprogramming OS monitors active program states and system resources to prevent CPU idleness, ensuring the CPU is always working, as long as there are jobs available.



Advantages:

1: High CPU utilization.

2: It appears that many programs are allotted to CPU almost simultaneously.

Disadvantages:

1: CPU scheduling is required.

2: To accommodate many jobs in memory, memory management is required.

Time-sharing systems

1: A time-sharing system, also known as multitasking, is an extension of multiprogramming that supports interactive users.

2: It employs CPU scheduling and multiprogramming to provide an economical and interactive environment for two or more users.

3: Each user is given a small portion of the shared computer's CPU at a time, with their separate program in memory.

4: it allows users to use the CPU in turns. Time-sharing systems are more complex than multiprogramming ones, requiring memory management for isolation and protection of co-resident programs.

5: They utilize techniques like virtual memory to optimize response times by swapping jobs in and out of disk from main memory.

Advantages:

1: Each task gets an equal opportunity.

2: Less chances of duplication of software.

3: CPU idle time can be reduced.

Disadvantages:

1: Reliability problem.

2: One must have to take of security and integrity of user programs and data.

3: Data communication problem.

Real Time Operating System

1: RTOS is an operating system intended to serve real time application that process data as it comes in, mostly without buffer delay.

2: Real time systems which were originally used to control autonomous systems such as satellites, robots and hydroelectric dams.

3: A real time operating system is one that must react to inputs and responds to them quickly.

4: A real time system cannot afford to be late with a response to an event or else the output will be useless or failure.

5: Real time systems are divided into two groups: Hard real time system and soft real time system.

A hard real time system: Hard RTOS, the deadline is handled very strictly which means that given task must start executing on specified scheduled time, and must be completed within the assigned time duration. Example: Medical critical care system, aircraft systems, etc.

Soft real time system: is a less restrictive type. Soft Real time RTOS, accepts some delays by the Operating system. In this type of RTOS, there is a deadline assigned for a specific job, but a delay for

a small amount of time is acceptable. So, deadlines are handled softly by this type of RTOS. Example: Online Transaction system and Livestock price quotation System.

Applications: Example

Detection: Radar system, Burglar alarm

Process monitoring and control: Petroleum, Paper mill

Communication: Telephone switching system

Flight simulation and control: Auto pilot shuttle mission simulator

Transportation: Traffic light system, Air traffic control

Military applications: missiles

Weather forecasting.

OS for Personal Computers and Workstations

1: In the late 1970s, computers showcased faster CPUs, creating a significant gap with slower I/O access time.

2: Multi programming faced limits due to expensive, limited main memory.

3: MS Windows and Apple Macintosh OS introduced advancements like virtual memory and multitasking, enabling programs to run without fully residing in memory.

4: Microcomputers, initially for single users, evolved to handle more extensive software and greater speeds.

5: MS-DOS exemplifies a microcomputer OS. Today, powerful microcomputers are used by commercial, educational, and government entities, thanks to low hardware costs.

6: Workstations, superior to personal computers, find applications in businesses and professions requiring ample processing power, RAM, and high-speed graphics, often using UNIX as the OS.

User View of Operating System

1: If the user is using a personal computer, the operating system is largely designed to make the interaction easy.

2: If the user is using a system connected to a mainframe or a minicomputer, the operating system is largely concerned with resource utilization.

3: If the user is sitting on a workstation connected to other workstations through networks, then the operating system needs to focus on both individual usage of resources and sharing though the network.

4: If the user is using a handheld computer such as a mobile, then the operating system handles the usability of the device including a few remote operations.

Unit 2 (Operating System Functions and Structure)

Process management

1: Process management involves handling programs in execution.
2: Every software application runs as one or more processes on the operating system.
3: For instance, using Google Chrome initiates a process for the browser program.
4: Simultaneously, the OS runs multiple processes performing various functions.
5: Processes require specific resources like CPU time, memory, and I/O devices, either given during creation or allocated while running.
6: When a process ends, reusable resources are reclaimed by the OS.
7: Process management ensures these processes are sequentially executed, with at least one instruction executed on behalf of each process.
8: Unlike a passive program, a process refers to an executing set of machine instructions.

Main Memory Management

1: Main Memory is a vast array of bytes with unique addresses, managed through reads and writes of specific memory addresses.

2: To run a program, it must be mapped to absolute addresses and loaded into Memory.

3: The OS's memory management modules handle primary memory, focusing on functions like tracking memory status (free or allocated), determining allocation policies, allocating and updating memory for processes, and deallocating memory when no longer needed.

4: Memory management's efficient allocation impacts overall resource utilization and performance criteria of a computer system, as it deals with finite capacity physical memory allocated to processes.

Secondary Storage Management

1: A storage device is a mechanism by which the computer may store information in such a way that this information may be retrieved at a later time.

2: Secondary storage device is used for storing all the data and programs.

3: These programs and data accessed by computer system must be kept in main memory. The Size of main memory is small to accommodate all data and programs.

4: It also loses the data when power is lost.

5: For this reason, secondary storage device is used.

6: Therefore the proper management of disk storage is of central importance to a computer system

---

File Management

1: Files are named collections of logically related data items on secondary storage.

2: They represent logical collections of information, such as reports, executable programs, or OS commands.

3: Files are composed of bits, bytes, lines, or records with meanings defined by their creators.

4: Physical media like magnetic disks, tapes, and optical disks store these files.

5: Each medium has unique characteristics and physical organization, controlled by specific devices.

6: Secondary storage efficiently holds various files and their data on different types of physical media, facilitating data access and management for computers.

---

operating system responsibility for connection with file management

1. Creating and deleting of files.

2. Mapping files onto secondary storage.

3. Creating and deleting directories

4. Backing up files on stable storage media.

5. Supporting primitives for manipulating files and directories.

6. Transmission of file elements between main and secondary storage.

System Calls

1: A system call serves as the crucial interface between a process and the operating system, enabling a computer program to request services from the OS kernel.

2: Functions like file creation, deletion, reading, and writing rely on system calls.

3: These calls provide user programs access to OS services via an API (Application Programming Interface) and serve as the sole entry points to the kernel system.

4: Modern processors offer instructions designed for system calls, triggering an interrupt that grants the operating system control over the processor when executed.

5: In summary, system calls facilitate seamless communication and interaction between user programs and the underlying operating system.

Services Provided by System Calls

1. Process creation and management

2. Main memory management

3. File Access, Directory and File system management

4. Device handling(I/O)

5. Protection

6. Networking, etc.

Types of System Calls

1. Process control/creation and management:

a. Create process

b. Terminate process: Terminate the process making the system call

c. Wait: wait for another process to exit

2. File Access, Directory and File system management :

create file.

open file to read or write.

close file.

delete a file.

3. I/O Device management:

a. Request device: to ensure exclusive use of device.

b. Release device: Release the device after execution is finished

c. Read/Write: Same as file system call.

4. Information maintenance:

a. Get time and date.

b. Set time and date.

c. Get process, file or device attributes.

5. Inter-process communication:

a. Create message queue: Create a queue to hold message Send a message to a message queue.

b. Send message: send message to message queue

c. Receive message: Receive a message from a message queue.

Kernel

1: The kernel is a crucial component of an operating system, managing system resources and acting as a link between software and hardware.

2: Loaded after the boot loader during start-up, it facilitates secure access to hardware for programs.

3: Employing inter-process communication and system calls, it bridges applications and hardware-level data processing.

4: Responsible for low-level tasks like disk, task, and memory management, the kernel comes in various types, with monolithic and microkernel being the most widely used.

5:Kernel space executes the core OS functions, while user space is reserved for user processes.

6: A process denotes an executing instance of a program.

Monolithic (Simple) Operating System

1: Monolithic Kernel manages system resources between application and hardware, but user services and kernel services are implemented under same address space.

2: It increases the size of the kernel, thus increases size of operating system as well.

3: This kernel provides CPU scheduling, memory management, file management and other operating system functions through system calls.

4: As both services are implemented under same address space, this makes operating system execution faster.

5: If any service fails the entire system crashes, and it is one of the drawbacks of this kernel.

6: The entire operating system needs modification if user adds a new service.

Advantages of monolithic kernel

1: Monolithic kernels tend to be more efficient in terms of performance as they operate in a single address space, allowing direct communication between kernel modules.

2: Monolithic kernels are relatively easier to design, implement, and maintain.

3: Monolithic kernels are well-suited for systems where the hardware and software are tightly integrated.

Disadvantages of Monolithic Kernel:

1: Monolithic kernels lack the modular design, making it difficult to isolate and update individual components without affecting the entire system.

2: A bug or failure in one kernel module can potentially crash the entire system, affecting its stability and reliability.

3: Adding new features or functionalities to the kernel might be complex and can result in code bloat, affecting overall system performance.

Layered Operating System

1: The operating system architecture can be designed in a layered approach for robustness and modularity.

2: Each layer has different privileges, with the most privileged layer handling interrupt and context switching, followed by device drivers, memory management, file systems, user interface, and applications.

3: The hardware layer (0) forms the bottom, while the user interface (layer N) represents the topmost layer.

4: Layers use the interface of the layer below and provide a more intelligent interface above.

5: This modularity simplifies debugging, system verification, and information hiding. However, layered implementations may be less efficient compared to other approaches due to the strict hierarchy and dependency on lower layers.

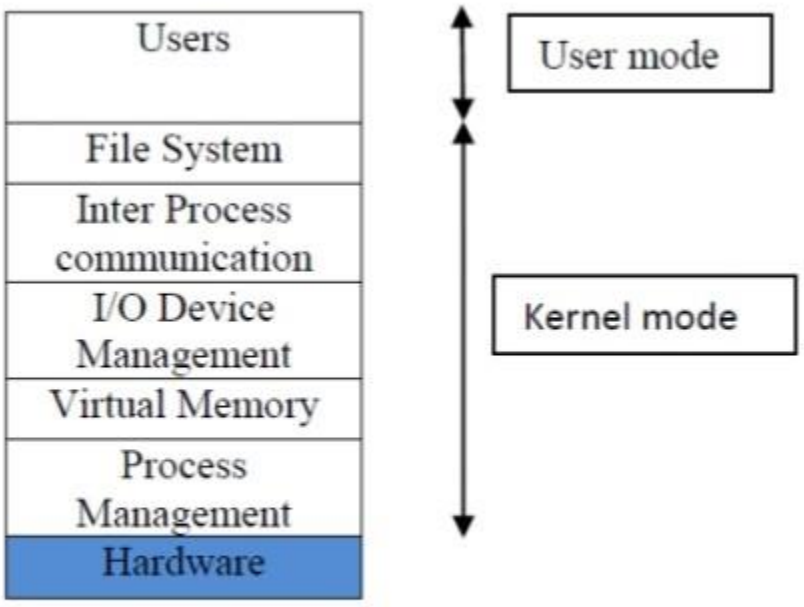


Figure 2.3 Layered OS

Advantages:

1: modularity

2: easy debugging

3: Easy update.

4: No direct access to hardware.

5: Abstraction:

Disadvantages:

1: Complex and careful implementation.

2: slower in execution.

Microkernel Operating System

1: A microkernel is a minimalistic software containing essential functions, data, and features to implement an operating system.

2: It provides a limited set of mechanisms sufficient to run basic OS functions and allows flexibility by not enforcing many policies.

3: Microkernels and their user environments are often implemented in C++ or C, with some assembly code.

4: Basic operations like memory management, process scheduling, and inter-process communication are fulfilled by the microkernel.

5: Unlike traditional kernels, only the microkernel executes at the privileged level, while other critical OS functionalities, such as device drivers, applications, file servers, and inter-process communication, are moved to the user mode for improved stability and security.

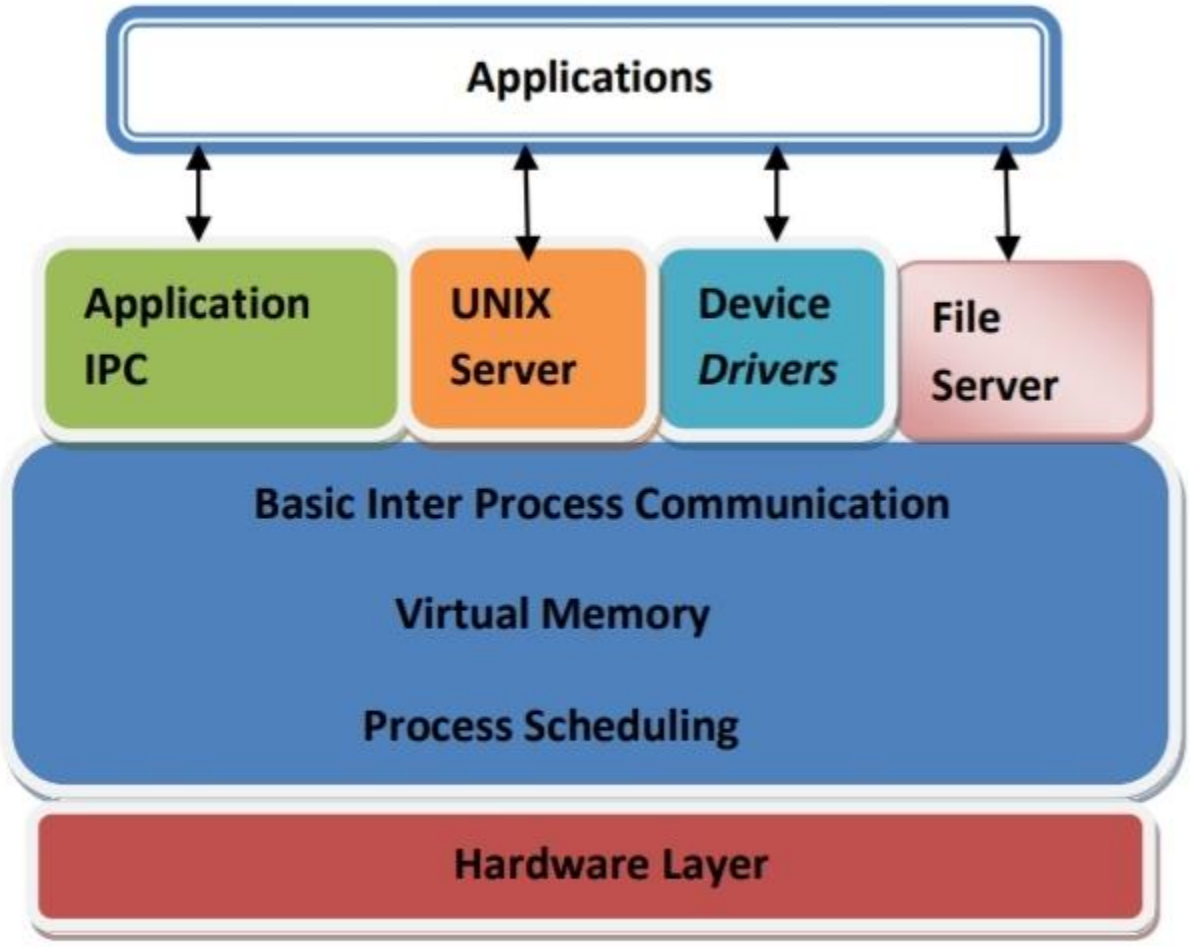


Figure 2.4 Microkernel structure

Advantages of Microkernel

1: Microkernel architecture is small and isolated therefore it can function better.

2: Microkernels are modular, and the different modules can be replaced, reloaded, modified without even touching the Kernel.

3:. Fewer system crashes when compared with monolithic systems.

Disadvantage of Microkernel

1: Providing services in a microkernel system are expensive compared to the normal monolithic system.

2: Context switch or a function call needed when the drivers are implemented as procedures or processes, respectively.

3: The performance of a microkernel system can be indifferent and may lead to some problems.

Exo-kernel Operating system

1: The Exo-kernel, a creation of MIT, grants applications control over hardware resources, enabling application-specific customization.

2: Unlike other kernels, it concentrates on safeguarding and multiplexing raw hardware without providing abstractions for applications.

3: This separation permits developers to efficiently utilize hardware for each program.

4: User-mode processes can directly access Kernel resources, including process tables.

5: Exo-kernels are compact but accompanied by library operating systems, providing conventional OS functionalities.

6: A key advantage is their ability to incorporate multiple library operating systems, allowing simultaneous execution of diverse APIs like Linux and Windows applications.

7: Compared to Micro-kernels and Monolithic Kernels, Exo-kernels are smaller and offer direct hardware access by eliminating unnecessary abstractions.

Virtual machine

1: Virtual Machines (VMs) abstract and isolate hardware components (CPU, disk drives, memory, NIC) into multiple execution environments to meet specific needs, creating a sense of distinct computers.

2: VMs, exemplified by VirtualBox, give the impression that each process runs on its virtual memory using CPU scheduling and virtual-memory techniques.

3: Although lacking certain functionalities like system calls and a file system, VMs provide a basic hardware interface.

4: They enable shared hardware utilization while supporting different operating systems concurrently.

5: However, disk space becomes a challenge when multiple VMs require more resources, necessitating the use of virtual disks.

6: Overall, VMs offer a partitioned approach for designing multi-user interactive systems in a manageable manner.

Advantages:

1: There are no protection problems because each virtual machine is completely isolated from all other virtual machines.

2: Virtual machine can provide an instruction set architecture that differs from real computers.

3: Easy maintenance, availability and convenient recovery.

Disadvantages:

1: When multiple virtual machines are simultaneously running on a host computer, one virtual machine can be affected by other running virtual machines, depending on the workload.

2: Virtual machines are not as efficient as a real one when accessing the hardware.

Booting

1: Booting is a start-up sequence that starts the operating system of a computer when it is turned on.

2: A boot sequence is the initial set of operations that the computer performs when it is switched on.

3: Every computer has a boot sequence.

4: The BIOS, operating system and hardware components of a computer

system should all be working correctly for it to boot.

5: If any of these elements fail, it leads to a failed boot sequence.

System boot process

1: During computer startup, the CPU initiates by generating clock ticks from the system clock.

2: It seeks the initial instruction in the ROM BIOS for the start-up program.

3: The BIOS triggers the Power-On Self-Test (POST) to check BIOS chip and CMOS RAM, proceeding with CPU initialization if no battery failures are found.

4: POST further examines hardware devices, secondary storage like hard drives, ports, mouse, and keyboard to ensure proper functionality.

5: Once confirmed, the BIOS locates the operating system, often from the C drive on the hard drive, using information from the CMOS chip.

6: Following the boot sequence, the BIOS loads the operating system's boot record into memory, allowing the operating system to take control of the boot process.

7: The operating system then verifies system memory and loads device drivers to manage peripheral devices.

Unit 3 (Information Management)

Information Management

1: Information management (IM) is the collection and management of information from one or more sources and the allocation of that information to one or more addresses.
2: The operating system manages the information from which it should be sent or received.
3: The file system in the operating system keeps track of information, its location and everything.
4: It also decides on which user should get resources first and puts into effect the protection requirements and also provides access routines.
5: Also**,** necessary information like opening a file, reading a file is taken care of by the file system and finally it retrieves the resources like closing a file.

Information Maintenance

1: Information maintenance is a system call that is used to maintain information.

2: Information maintenance in an operating system refers to the processes and mechanisms that ensure the accurate, consistent, and reliable management of data and information within the system.

3: This includes tasks such as file organization, data integrity checks, data synchronization, and managing access permissions to prevent unauthorized modifications.

4: Operating systems employ various techniques and algorithms to handle these tasks efficiently, ensuring the overall stability and functionality of the system.

5: examples of information maintenance, include getting system data, set time or date, get time or date, set system data, etc.

Floppy disk

1: A Floppy disk is made up of a round piece of plastic material, coated with a magnetized recording material.

2: The surface is made up of concentric circles which are called as tracks.

3: Data is recorded on these tracks in bit and it contains magnetized particles of metal is having north and South Pole.

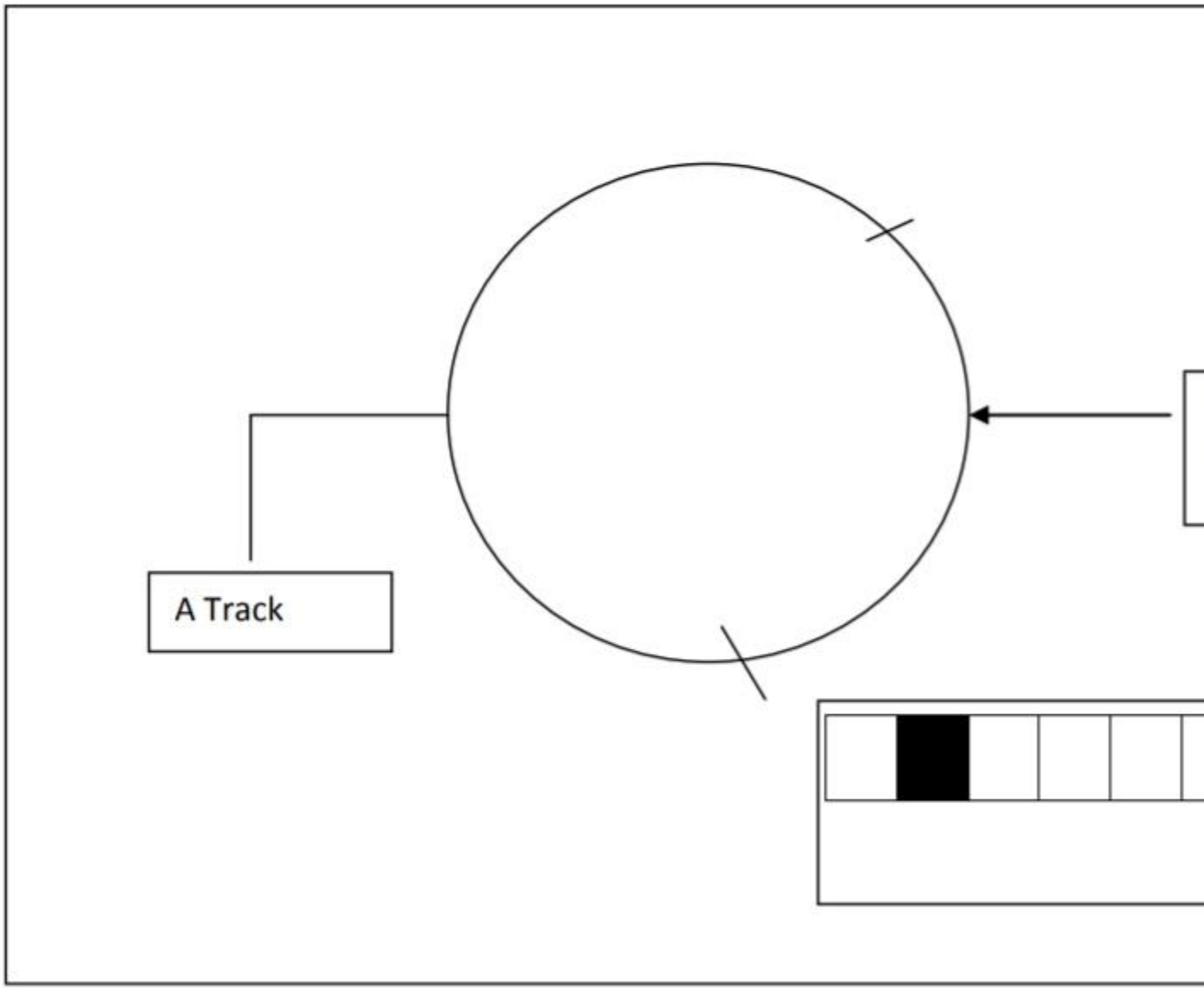
4: It is having only two directions so each

particle can acts as a binary values of 0 or 1 and 8 switches can record a character with the coding methods (ASCII).

5: A logical record which consists of different fields (data items), each

consisting of several characters which is stored in floppy disk.

Figure 3.3 Data Recording on the Disk



Magnetic disks

On top of magnetic disks, data is encoded as microscopic magnetized needles on the disk‘s surface. One can record and erase data on a magnetic disk any number of times, just as with a cassette tape. Magnetic disks come in a number of different forms:

Floppy disk: A typical 5 inch floppy disk can hold 360K or 1.2MB (megabytes). 3-inch floppies normally store 720K, 1.2MB or 1.44MB of data. Floppy disks or outdated today, and are found on older computer systems.

Hard disk: Hard disks can store anywhere from 20MB to more than 1-TB

(terabyte). Hard disks are also from 10 to 100 times faster than floppy disks.

Removable cartridge: Removable cartridges are hard disks covered in a metal or plastic cartridge, so you can remove them just like a floppy disk. Removable cartridges are very fast, though usually not as fast as fixed hard disks. GT

Optical disks

Optical disks record data by burning microscopic holes in the surface of the disk with a laser. To read the disk, another laser beam shines on the disk and detects the holes by changes in the reflection pattern.

Optical disks come in three basic forms:

CD-ROM: Most optical disks are read-only. When you purchase them, they are already filled with data. You can read the data from a CD-ROM, but you cannot modify, delete, or write new data.

WORM: Stands for write-once, read-many. WORM disks can be written on once and then read any number of times; however, you need a special WORM drive to write data onto a WORM disk.

Erasable optical (EO): EO disks can be read to, written to, and erased just like

magnetic disks.

Additional disk points

1: Accessing data from a disk is not as fast as accessing data from main memory, but disks are much cheaper.

2: And unlike RAM, disks hold on to data even when the computer is turned off.

3: Accordingly, disks are the storage medium of choice for most types of data.

4: Another storage medium is magnetic tape.

5: But tapes are used only for backup and archiving because they are sequential-access devices (to access data in the middle of a tape, the tape drive must pass through all the previous data).

A disk can be considered several surfaces, each of which consisting numb
is shown in figure 3.4. The tracks are normally numbered from 0 as outerm
number increasing inwards. Each track is divided into a number of sectors
equal size. And a sector can store up to 512 bytes. Double sided floppies
surfaces on which data can be recorded. So a given sector is specified by th
an address made up of surface number, track number and sector number.

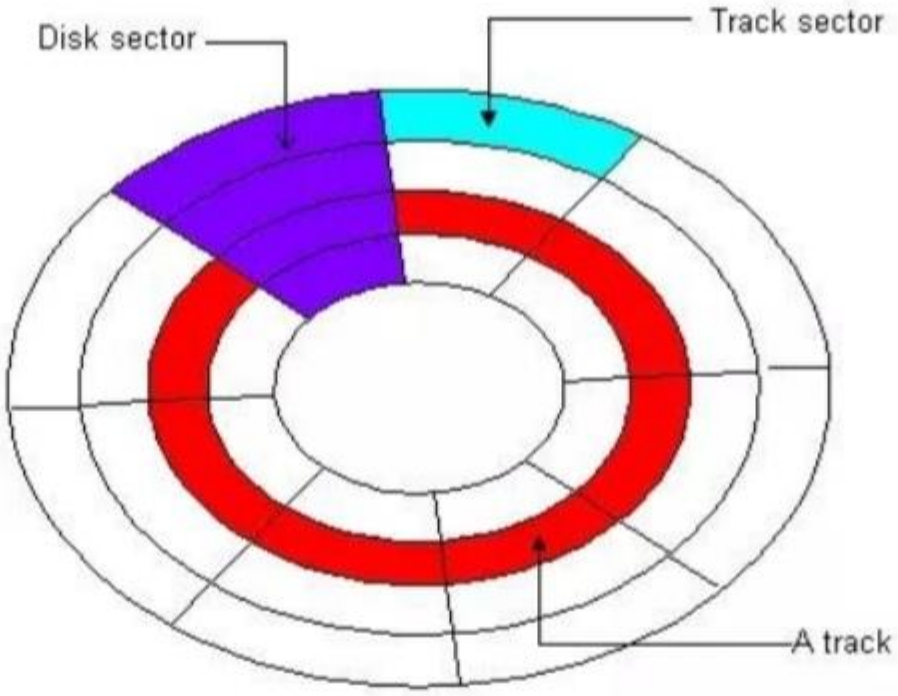


Figure 3.4 Tracks and Sectors (Source: installsetupconfig.c

Direct Memory Access (DMA)

1: Slow devices, like keyboards, generate interrupts for each transferred byte to the main CPU, while fast devices, such as disks, would overwhelm the CPU with interrupts.

2: To mitigate this, computers employ Direct Memory Access (DMA) hardware, allowing I/O modules to autonomously read from or write to memory.

3: DMA manages data exchange between main memory and I/O devices, with CPU involvement only at the transfer's start and end.

4: For instance, sound and video cards use DMA to access and process data, bypassing the CPU.

5: DMA requires a DMA controller (DMAC), programmed with pointers, counters, and settings to facilitate data transfers and manage system bus access.

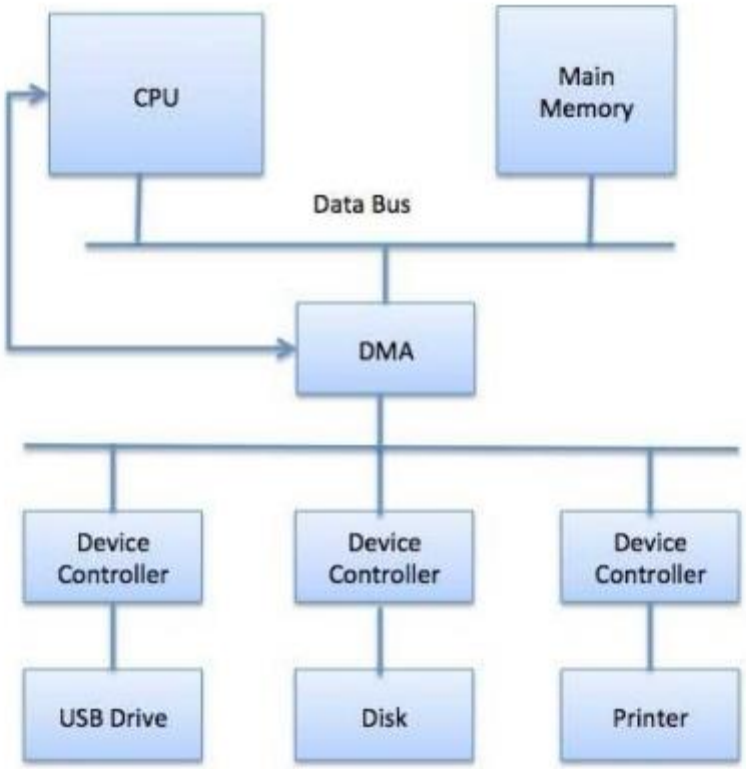


Figure 3.5 Direct Memory Access

3.3.1 Block diagram of DMA controller and DMA Operations

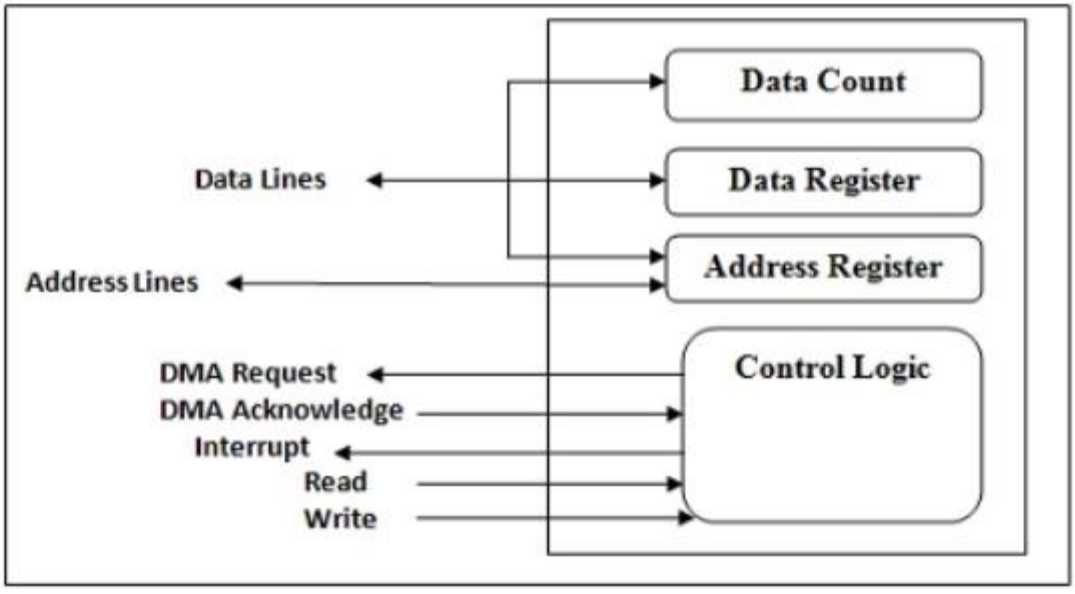


Figure 3.6 Block Diagram of DMA Controller

DMA Operation

1: Direct Memory Access involves transfer of data between I/O devices and memory by an external circuitry system called DMA controller without involving the microprocessor.

2: However, microprocessor itself initiates the DMA control process by providing starting address, size of data block and direction of data flow.

3: DMA contains a control unit to deals

with the control functions during DMA operations such as read, write and interrupt.

4: The address register of DMA controller is used to generate address and select I/O device to transfer the data block.

5: The Count registers counts and hold no. of data block transferred.

6: It also specifies direction of data transfer.

Steps of DMA Operation

a) For DMA operation to occur, the DMA controller first make a bus request (BR) by sending a control signal HOLD to the control line.

b) On receiving the BR through HOLD pin high, the microprocessor completes the current instruction execution and afterward it generates HLDA control signal and sends it to the DMA-Controller. This event switches over the control from microprocessor to DMA Controller. The microprocessor gets idle.

c) As soon as DMA controller receives HLDA (Hold Acknowledged) through Bus Grant (BG) line, it takes the control of system bus and start transferring the data blocks between memory and Input / Output devices, without involving the microprocessor.

d) On completion of data transfer, the DMA controller sends a low signal to the HOLD pin and hence microprocessor makes the HLDA pin low and takes the control over system bus.

DMA Operation Modes:

The DMA Controller operates under three modes:

Burst Mode: Here DMA controller switch over the control to the microprocessor only on completion of entire data transfer, irrespective of microprocessor requiring the bus. Microprocessor has to be idle during the data transfer.

Cycle-Stealing Mode: DMA controller hand over the control to microprocessor on transfer of every byte, thereby microprocessor gets the control and become able to process highly prioritized instruction. DMA need to make the BG request for each byte.

Transparent Mode: In this mode, DMA controller can transfer data blocks only when microprocessor are executing such instruction that does not requires system bus utilization.

Direct Memory Access Advantages and Disadvantages

Advantages:

1. Transferring the data without the involvement of the processor will speed up the read-write task.

2. DMA reduces the clock cycle requires to read or write a block of data.

3. Implementing DMA also reduces the overhead of the processor.

Disadvantages

1. As it is a hardware unit, it would cost to implement a DMA controller in the system.

2. Cache consistency problem can occur while using DMA controller.

---

block and block numbering scheme

1:In operating systems, a block is a fixed size data storage unit on devices like hard drives.

2:Each block is uniquely identified by a block number, crucial for managing data.

3: The block numbering scheme organizes blocks to build a logical structure on storage devices.

4: Files are divided into blocks, each assigned sequential block numbers.

5: This approach ensures efficient space allocation, rapid random access, and error handling.

6: However, it can lead to internal fragmentation when small files don't use an entire block.

7: This block-based storage foundation is fundamental to modern file systems, enabling effective data management and retrieval.

---

arrangements of logical records

1: Arrangements of logical records involve various methods like Relative Record Number (RRN), Relative Block Number (RBN), and Primary Record Number (PRN).

2: RRN assigns a unique number to each record within a file.

3: RBN uses block numbers as identifiers, aiding in organizing records across blocks.

4: PRN involves associating records with primary keys, facilitating rapid data retrieval.

5: These techniques enhance data organization and access efficiency in file systems.

example:

Imagine all the customer records put one after another like carpet shown in following f

3.8.

PRN=0	PRN = 1	.....	PRN = 9
BYTES 0-699	BYTES 700-1399		BYTES 6300-6

Figure 3.8 : The PRN and RBN

If 10 customer records are there the 700*10 = 7000 bytes will be written onto the cust

for PRN = 0 to 9. The operating system can calculate a Relative byte number (RBN

every record. This is the starting byte number for each record. RBN is calculated with re

to 0 as the starting byte number and put after other as logical records. Following f

shows the relationship between RRN and RBN for the records shown in above figure.

RRN	RBN
0	0
1	700
2	1400

---

arrangement of blocks

Arrangement of blocks refers to how data is organized and stored in blocks within a storage device, typically in the context of file systems and operating systems. Here's a concise explanation:

Determine File Size: Determine the size of the file you want to store. This helps in estimating the number of blocks needed for storage.

Choose Block Size: Select a suitable block size for your file system. Common block sizes are 512 bytes, 1 KB, 4 KB, etc. The chosen block size affects how efficiently space is utilized and how many blocks are allocated to a file.

Calculate Number of Blocks: Divide the file size by the chosen block size to calculate the number of blocks needed for the file. If the file size is not an exact multiple of the block size, there might be some unused space in the last block, known as internal fragmentation.

Allocate Blocks: Depending on the file system's allocation strategy, there are different methods to allocate blocks:

Relationship between the Operating System and DMS

1: The operating system can present records to an Application Program (AP) in the order they were written.

2: If another AP needs records sorted differently, it's the AP's responsibility to rearrange them for proper retrieval.

3: For selective sequential processing, maintaining an index becomes essential. Data Management Systems (DMS) handle these indexes, functioning separately from the OS.

4: Initially part of the OS, complex data functions led to separate DMS, including File Management Systems (FMS) and Database Management Systems (DBMS) like ORACLE, DB2, INFORMIX.

5: DMS bridges the gap between the OS and AP, enhancing data organization and retrieval efficiency.

File Directory Entry

1: Group of files combined is known as directory.

2: A directory contains all information about file, its attributes.

3: The directory can be viewed as a symbol table that translates file names

into their directory entries.

4: Directory itself can be organized in many ways.

5: For each file created by the operating system, the OS maintains a directory entry which is also known as Volume Table of Contents (VTOC) in IBM.

6: The logical records of VTOC or file directory entries are stored on the disk using some hashing algorithm on the file name.

Open/Close Operations in files

Open ( )

1: A file can be opened in one of the two modes, read mode or write mode.

2: In read mode, the OS does not allow anyone to alter data; file opened in read mode can be stored among several entities.

3: Write mode allows data modifications, file opened in write mode can be read but cannot be shared.

Close ( )

1: This is the most important operation from OS point of view.

2: When a request to close a file is generated the OS.

3: OS removes all the block (in shared mode)

4: Saves the data if altered to the secondary storage media.

5: Releases all the buffer and file handlers associated with a file.

Contiguous Allocation

1: In this scheme, each file occupies a contiguous set of blocks on the disk.

2: For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: b, b+1, b+2,.....b+n-1.

3: This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.

Advantages:

1: Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).

2: This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

Disadvantages:

1: This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.

2: Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

Linked List Allocation

1: In this scheme, each file is a linked list of disk blocks which need not be contiguous.

2: The disk blocks can be scattered anywhere on the disk.

3: The directory entry contains a pointer to the starting and the ending file block.

4: Each block contains a pointer to the next block occupied by the file.

Advantages:

1: This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.

2: This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

Disadvantages:

1: Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.

2: pointers required in the linked allocation incur some extra overhead.

Indexed Allocation

1: In this scheme, a special block known as the Index block contains the pointers to all the blocks occupied by a file.

2: Each file has its own index block.

3: The ith entry in the index block contains the disk address of the ith file block.

For files that are very large, a single index block may not be able to hold all the pointers. Following mechanisms can be used to resolve this:

Linked scheme: This scheme links two or more index blocks together for holding the pointers. Every index block would then contain a pointer or the address to the next index block.

Multilevel index: In this policy, a first level index block is used to point to the second level index blocks which in turn points to the disk blocks occupied by the file. This can be extended to 3 or more levels depending on the maximum file size.

The Combined Scheme: involves an I node (Information Node) that stores file details like name and size, along with addresses of Disk Blocks holding the file's actual data. Pointers in the I node refer to direct and indirect blocks. Indirect blocks can be single, double, or triple, serving as intermediaries to access the file data through stored addresses.

Directory Structure: User's View

1: By considering the user' s viewpoint, a directory is a file of files, i.e. a single file containing details about other files belonging to that directory.

2: A Directory is the collection of the correlated files on the disk.

3: In simple words, a directory is like a container which contains files and folders.

4: In a directory, we can store the complete file attributes or some attributes of the file.

5: A directory can be comprised of various files.

6: With the help of the directory, we can maintain the information related to the files.

Operations on Directory

1. creating: – In this operation, a directory is created. The name of the directory should be unique.

2. Deleting: – If there is a file that we don't need, then we can delete that file from the directory. We can also remove the whole directory if the directory is not required. An empty directory can also be deleted. An empty directory is a directory that only consists of dot and dot-dot.

3. Searching: – Searching operation means, for a specific file or another directory, we can search a directory.

4. List a directory: – In this operation, we can retrieve the entire files list in the directory. And we can also retrieve the content of the directory entry for every file present in the list.

Types of Directory Structure

1: Single-Level Directory: – Single-Level Directory is the easiest directory structure. There is only one directory in a single-level directory, and that directory is called a root directory. In a single-level directory, all the files are present in one directory that makes it easy to understand. In this, under the root directory, the user cannot create the subdirectories.

2: Two-Level Directory: Two-Level Directory is another type of directory structure. In this, it is possible to create an individual directory for each of the users. There is one master node in the two-level directory that includes an individual directory for every user. At the second level of the directory, there is a different directory present for each of the users. Without permission, no user can enter into the other user's directory.

3: tree-structured directory: there is an own directory of each user, and any user is not allowed to enter into the directory of another user. Although the user can read the data of root, the user cannot modify or write it. The system administrator only has full access to the root directory. In this, searching is quite effective and we use the current working concept. We can access the file by using two kinds of paths, either absolute or relative.

4: Acyclic-Graph Directory**:** In the tree-structure directory, the same files cannot exist in the multiple directories, so sharing the files is the main problem in the tree-structure directory. With the help of the acyclic-graph directory, we can provide the sharing of files. In the acyclic-graph directory, more than one directory can point to a similar file or subdirectory.

5. General-Graph Directory: The General-Graph directory is another vital type of directory structure. In this type of directory, within a directory we can create a cycle of the directory where we can derive the various directories with the help of more than one parent directory. The main issue in the general-graph directory is to calculate the total space or size, taken by the directories and the files.

Implementation of a Directory System

linked list

1: The basic method for implementing a directory involves using a linear list of file names with pointers to data blocks.

2: This approach is easy to program but slow to execute.

3: Creating a new file requires searching for name conflicts and adding a new entry. Deleting a file involves searching for the file and releasing its space.

4: Strategies to manage directory entries include marking as unused, attaching to a free list, or copying the last entry.

5: Linear searches make file finding slow, leading to caching and sorted lists for better access.

6: Sorted lists offer binary search advantages but complicate file creation and deletion.

7: More complex structures like B-trees can address this.

Hash Table

1: A hash table is a different file directory structure involving both a linear list and a hash data structure.

2: The hash table uses a value from the file name to point to the corresponding entry in the linear list, speeding up searches.

3: Insertion and deletion are manageable, though collisions need addressing.

4: Challenges include fixed size limitations and hash function dependence.

5: For instance, increasing entries means rehashing.

6: Chained-overflow hash tables use linked lists for entries, addressing collisions and enabling dynamic growth.

7: This method might slow lookups due to linked list traversal for name search.

---

device driver

1: A device driver, often referred to as a driver, is a computer program that controls a specific device connected to a computer.

2: It acts as a software bridge between hardware and operating systems, allowing programs to access hardware functions without intricate hardware knowledge.

3: Drivers interact with devices via computer buses, issuing commands and receiving data.

4: They handle interrupts and time-dependent hardware interactions, varying by hardware and operating system.

5: These software modules plug into operating systems to manage specific devices, encapsulating device-dependent code and providing standard interfaces.

6: Typically created by manufacturers, drivers are delivered with devices on CD-ROMs to facilitate hardware interaction.

Functions of Device Driver

1: It accepts abstract read and write requests from the device independent software above to it.

2: It controls the power requirement and log event.

3: It checks the status of the device. If device is in idle state then request is processed and if device is in busy state then it queue the request which will be processed later.

4: Interact with the device controller to take and give I/O and perform required error handling.

5: Making sure that the request is executed successfully

Path Management

1: Various controllers, with their substantial memory buffers and intricate electronics, incur high costs.

2: A cost-saving notion involves attaching different devices to a single controller.

3: Despite potential parallelism from overlapped seeks, each controller can manage only one device at a time, causing waiting if both devices request I/O.

4: Device drivers queue pending requests in a device request queue, using data structures and algorithms.

5: Complex mainframe controllers split into Channels and Control Units (CUs).

6: Channels, like small computers, handle specific I/O instructions and can manage diverse controller types.

7: Complex paths in multi-path systems increase driver routine complexity but enhance response time compared to single controllers per device.

The Sub modules of DD

file system and Device Drivers. Conceptually it is divided in four sub modules:

I/O Procedure

1: The I/O procedure manages path allocation and requests using data structures like Channel Control Block (CCB), Control Unit Control Block (CUCB), Device Control Block (DCB), and Input/output Request Block (IORB).

2: These structures reside in memory and are updated as requests arrive and are serviced.

3: Memory is partitioned for IORB records, each containing slots for devices, CUs, and channels.

4: Complex algorithms handle slot management, allocation, and linking IORBs to appropriate DCBs, CUCBs, and CCBs, constituting a critical part of the I/O procedure.

The I/O scheduler

1: The I/O scheduler addresses pending device I/O requests from different processes with varying priorities.

2: Although prioritizing by process importance seems rational, it's often not practical due to slow electromechanical I/O operations.

3: Instead, the focus is on minimizing disk arm movement and seek time, enhancing through output and response time.

4: This strategy may introduce a slight delay for higher-priority processes.

5: For instance, if two processes await a device, and one with lower priority seeks data from the same track where the read/write arms are, no seek time is needed.

Device Handler:

1: The device handler basically is a piece of software which prepares an I/O program for the channel or a controller, loads it for them and instructs the hardware to execute the actual I/O.

2: After this happens, the device handler goes to sleep.

3: The hardware performs the actual I/O and on completion, it generates an interrupt.

4: The ISR (Interrupt Service Routines) for this interrupt wakes up the device handler again.

5: It checks for any errors. If there is no error, the device handler instructs the DMA transfer the data to the memory.

Interrupt Service Routines (ISR)

1: An interrupt service routine (ISR) is a software routine that hardware invokes in response to an interrupt.

2: ISR examines an interrupt and determines how to handle it executes the handling, and then returns a logical interrupt value.

3: If no further handling is required the ISR notifies the kernel with a return value.

4: An ISR must perform very quickly to avoid slowing down the operation of the device and the operation of all lower-priority ISRs

Unit 4 (Process Management)

Process Management

1: Process management in operating systems encompasses a range of activities vital for efficient task execution.
2: It involves creating, scheduling, synchronizing, and terminating processes.
3: The operating system's scheduler determines which process gets CPU time, enhancing resource utilization.
4: Process synchronization mechanisms prevent conflicts arising from concurrent processes accessing shared resources.
5: Creation and termination mechanisms enable dynamic process generation and cleanup, optimizing system resources.
6: By orchestrating these tasks, process management maintains system stability, responsiveness, and resource allocation.
7: It's a crucial facet of modern operating systems, ensuring seamless multitasking, resource allocation, and overall system efficiency.

Multiprogramming in Operating System

1: Multiprogramming in an operating system as the name suggests multi means more than one and programming means the execution of the program.

2: when more than one program can execute in an operating system then this is termed a multiprogramming operating system.

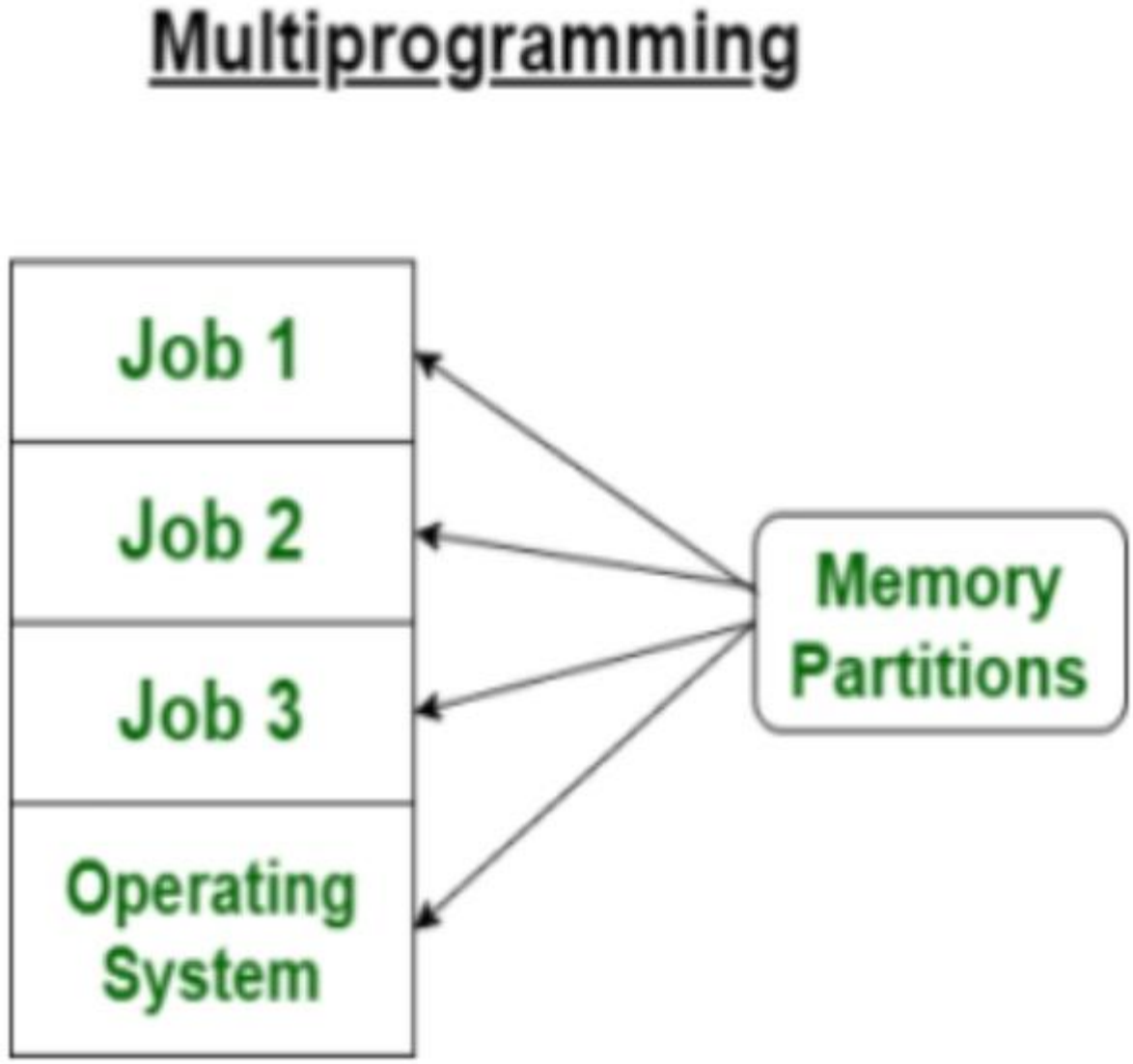
3: The major task of multiprogramming is to maximize the utilization of resources.

4: Multiprogramming is broadly classified into two types namely:

Multi-user operating system and Multitasking operating system.

5: Multitasking is an operating system that allows you to run more than one program simultaneously.

6: A multi-user operating system allows many users to share processing time on a powerful central computer on different terminals.



Advantages of Multiprogramming

1: Need Single CPU for implementation.

2: Context switch between process.

3: Switching happens when current process undergoes waiting state.

4: CPU idle time is reduced.

5: High resource utilization.

6: High Performance.

Disadvantages of Multiprogramming

1: Prior knowledge of scheduling algorithms is required.

2: If it has a large number of jobs, then long-term jobs will have to require a long wait.

3: Memory management is needed in the operating system because all types of tasks are stored in the main memory.

4: Using multiprogramming up to a larger extent can cause a heat-up issue.

Context Switch

1: Context switching in an operating system involves saving the context or state of a running process so that it can be restored later, and then loading the context or state of another process and run it.

2: Context Switching refers to the process/method used by the system to change the process from one state to another using the CPUs present in the system to perform its job.

3: Example – Suppose in the OS there (N) numbers of processes are stored in a Process Control Block(PCB). like The process is running using the CPU to do its job. While a process is running, other processes with the highest priority queue up to use the CPU to complete their job.

Need for Context switch

1: One process does not directly switch to another within the system. Context switching makes it easier for the operating system to use the CPU’s resources to carry out its tasks and store its context while switching between multiple processes.

2: Context switching enables all processes to share a single CPU to finish their execution and store the status of the system's tasks. The execution of the process begins at the same place where there is a conflict when the process is reloaded into the system.

3: Context switching only allows a single CPU to handle multiple processes requests parallelly without the need for any additional processors.

Process States

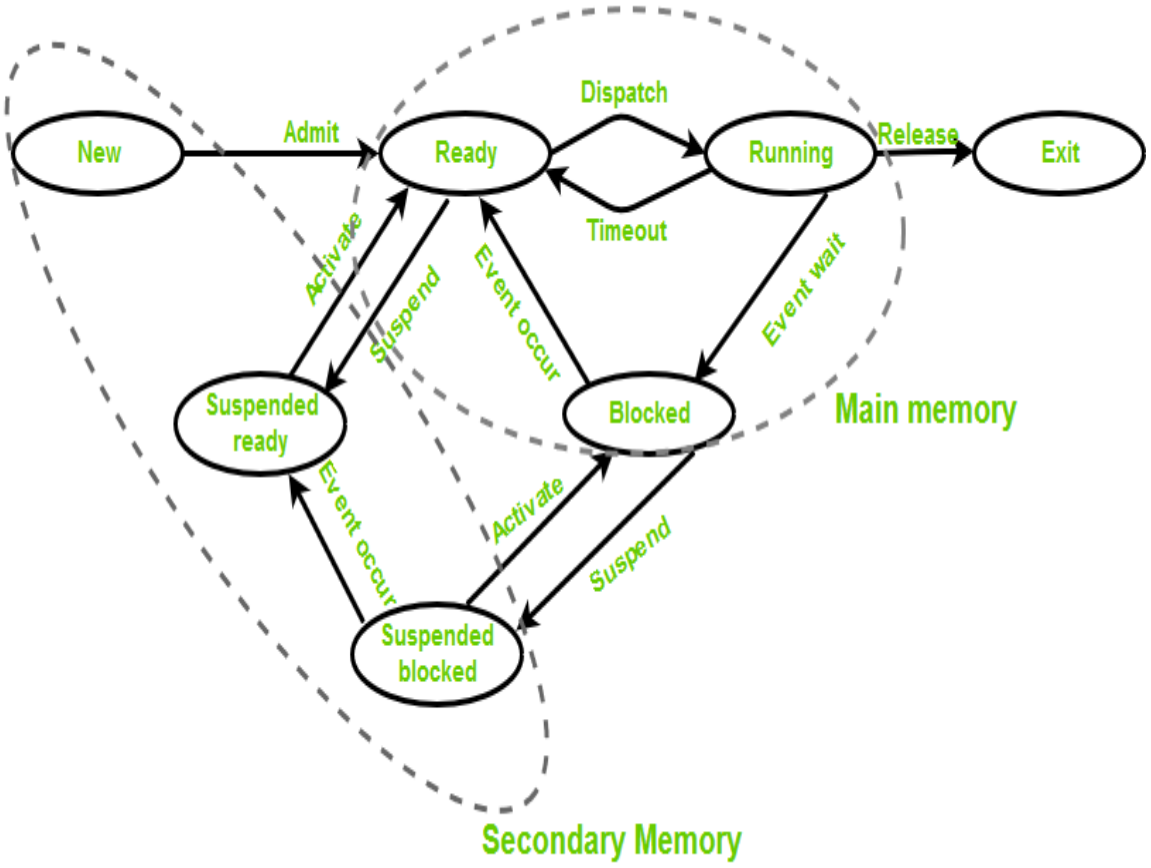
1: New (Create): In this step, the process is about to be created but not yet created. It is the program that is present in secondary memory.

2: Ready: New -> Ready to run. After the creation of a process, the process enters the ready state i.e. the process is loaded into the main memory.

3: Run: The process is chosen from the ready queue by the CPU for execution and the instructions within the process are executed by any one of the available CPU cores.

4: Terminated or Completed: Process is killed as well as PCB is deleted. The resources allocated to the process will be released or deallocated.

5: Suspend Ready: Process that was initially in the ready state but was swapped out of main memory and placed onto external storage by the scheduler is said to be in suspend ready state.



Process Transitions

The working of Process is explained in following steps:

1: User-running: Process is in user running.

2: Kernel-running: Process is allocated to kernel and hence, is in kernel mode.

3: Ready to run in memory: Further, after processing in main memory process is rescheduled to the kernel.

4: Asleep in memory: Process is sleeping but resides in main memory. It is waiting for the task to begin.

5: Ready to run, swapped: Process is ready to run and be swapped by the processor into main memory, thereby allowing kernel to schedule it for execution.

6: Sleep, Swapped: Process is in sleep state in secondary memory, making space for execution of other processes in main memory. It may resume once the task is fulfilled.

7: Pre-empted: Kernel preempts an on-going process for allocation of another process, while the first process is moving from kernel to user mode.

8: Created: Process is newly created but not running. This is the start state for all processes.

9: Zombie: Process has been executed thoroughly and exit call has been enabled. The process, thereby, no longer exists. But, it stores a statistical record for the process. This is the final state of all processes.

Process control Block

1: While creating a process the operating system performs several operations.

2: To identify the processes, it assigns a process identification number (PID) to each process.

3: As the operating system supports multi-programming, it needs to keep track of all the processes.

4: For this task, the process control block (PCB) is used to track the process's execution status.

5: Each block of memory contains information about the process state, program counter, stack pointer, status of opened files, scheduling algorithms, etc.

6: All this information is required and must be saved when the process is switched from one state to another.

7: When the process makes a transition from one state to another, the operating system must update information in the process's PCB.

8: A process control block (PCB) contains information about the process, i.e. registers, quantum, priority, etc.

Process hierarchy

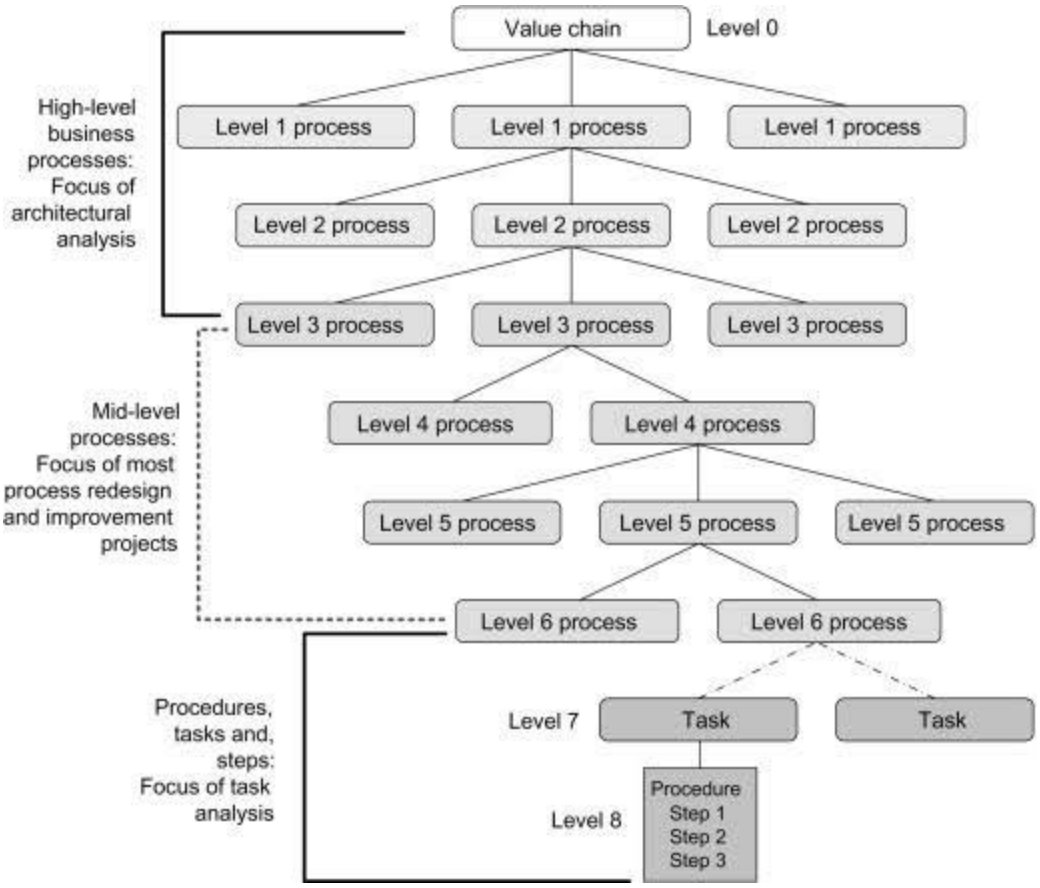
1: Process hierarchy refers to the organizational structure established by different processes operating within a computer system.

2: A primary process, often referred to as the parent, initiates the execution of secondary processes, known as children, to carry out designated tasks.

3: These child processes can themselves generate additional subprocesses, creating a layered, tree-like arrangement.

4: This hierarchy serves as a vital mechanism for orchestrating task allocation, optimizing resource utilization, and preserving the stability of the entire system.

5: By establishing a clear relationship between processes and their dependencies, process hierarchy contributes to efficient multitasking and streamlined system management.



Operation on a Process

1: Creation:

This is the initial step of the process execution activity. Process creation means the construction of a new process for execution. This might be performed by the system, the user, or the old process itself.

2: Scheduling/Dispatching

The event or activity in which the state of the process is changed from ready to run. It means the operating system puts the process from the ready state into the running state. Dispatching is done by the operating system when the resources are free or the process has higher priority than the ongoing process.

3: Blocking

When a process invokes an input-output system call that blocks the process, and operating system is put in block mode. Block mode is basically a mode where the process waits for input-output. Hence on the demand of the process itself, the operating system blocks the process and dispatches another process to the processor.

4: Preemption

When a timeout occurs that means the process hadn’t been terminated in the allotted time interval and the next process is ready to execute, then the operating system preempts the process.

5: Process Termination

Process termination is the activity of ending the process. In other words, process termination is the relaxation of computer resources taken by the process for the execution.

Block/Time Up/Wake Up a Process

Blocking a Process: This operation involves putting a process in a state where it waits for a specific event to occur before it can proceed. It's commonly used when a process needs to wait for input, data, or a signal.

Time Up a Process: This term isn't as commonly used. However, it might refer to setting a time limit or deadline for a process to complete its execution.

Waking Up a Process: This refers to bringing a blocked or suspended process back to an active state so that it can continue its execution. It's usually done when the event or condition the process was waiting for has occurred.

Suspend/Resume Operations

Suspending a Process: This operation involves temporarily pausing the execution of a process. The process's current state and data are preserved, but it doesn't consume CPU resources while suspended. This can be useful for resource management or when a process needs to be put on hold.

Resuming a Process: Resuming a suspended process means allowing it to continue its execution from where it was suspended. This is often used when the conditions that led to the suspension have been resolved

---

Process scheduling

1: Process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.

2: Process scheduling is an essential part of a Multiprogramming operating system.

3: Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

4: Scheduling falls into one of two categories:

Non-preemptive: In this case, a process's resource cannot be taken before the process has finished running. When a running process finishes and transitions to a waiting state, resources are switched.

Preemptive: In this case, the OS assigns resources to a process for a predetermined period of time. The process switches from running state to ready state or from waiting for state to ready state during resource allocation.

Time slice

1: It is timeframe for which process is allotted to run in preemptive multitasking CPU.

2: The scheduler runs each process every single time-slice.

3: The period of each time slice can be very significant and crucial to balance CPUs performance and responsiveness.

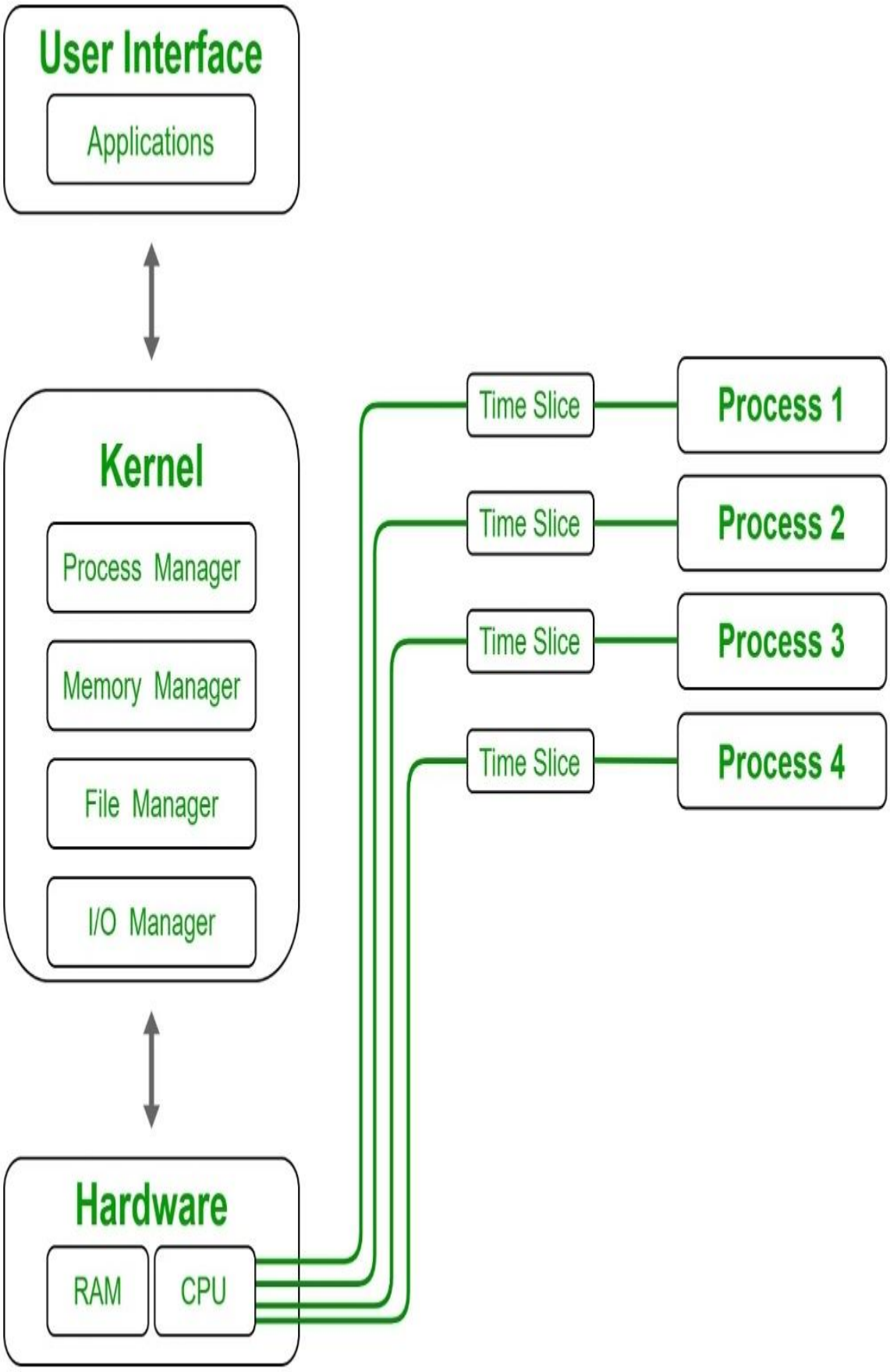
4: If time slice is quite short, scheduler will take more processing time.

5: In contrast, if the time slice is too long, scheduler will again take more processing time.

6: When process is allotted to CPU, clock timer is set corresponding to time slice.

7: If the process finishes its burst before time slice, CPU simply swaps it out like conventional FCFS calculation.

8: If the time slice goes off first, CPU shifts it out to back of ongoing queue.


User Interface
Applications
Kernel
Process Manager
Memory Manager
File Manager
I/O Manager
Hardware
RAM
CPU
Time Slice
Process 1
Time Slice
Process 2
Time Slice
Process 3
Time Slice
Process 4

Advantages of time slice

1: Fair allocation of CPU resources.

It deals all process with equal priority.

2: Easily implementable on the system.

3: Context switching method used to save states of preempted processes gives best performance in terms of average processing time.

Disadvantages of time slice

1: If the slicing time is short, processor output will be delayed.

2: It spends time on context switching.

3: Performance depends heavily on time quantum.

4: Priorities can’t be fixed for processes.

5: No priority to more important tasks.

6: Finding an appropriate time quantum is quite difficult.

Scheduling philosophies

1: First-Come, First-Served (FCFS): Processes are executed in the order they arrive in the ready queue. It's a simple approach, but it can lead to the "convoy effect" where a long process holds up shorter processes behind it.

2: Shortest Job Next (SJN) / Shortest Job First (SJF): Processes with the smallest execution time are given priority. This minimizes the average waiting time but can lead to starvation for longer processes.

3: Round Robin (RR): Each process is assigned a fixed time slice (quantum) to execute. When the time slice expires, the process is moved to the back of the queue, allowing other processes to run.

4: Priority Scheduling: Processes are assigned priority levels, and the scheduler executes processes with higher priority first. It can lead to starvation for lower-priority processes if not properly managed.

Scheduling levels

Typically, there are several scheduling levels, each corresponding to a different priority or class of processes. These levels can include:

1: Real-time Scheduling: This level is dedicated to processes that have strict timing constraints, where response times are critical. Real-time processes are given the highest priority to ensure they meet their deadlines.

2: Interactive Scheduling: Processes at this level are those that interact with users directly, such as user interface components. The goal is to provide responsive and interactive performance.

3: Batch Scheduling: Batch processes are those that are submitted in groups and can be executed without immediate user interaction. These processes are often assigned a lower priority and run during periods of lower system activity.

4: Background Scheduling**:** Background processes, like system maintenance tasks or non-urgent updates, are typically assigned a lower priority to avoid affecting the performance of more critical tasks.

Multi Threading Models

Multithreading involves the simultaneous execution of multiple threads within a process. Two threading models exist: user-level threads and kernel-level threads.

User-level threads: Managed by a user-level thread library within the application, the OS doesn't directly support threads. The library handles scheduling on available processors, granting flexibility and portability. However, user-level threads are less efficient than kernel-level threads as they rely on the application for scheduling.

Kernel-level threads: Directly supported by the OS as part of the kernel, each thread operates independently and can be scheduled by the OS. Kernel-level threads offer better performance and scalability, but they're less flexible and portable than user-level threads.

Hybrid models like the "two-level model" combine user-level and kernel-level threads for improved performance and resource management.

User-level threads Advantages and Disadvantages

Advantages:

Greater flexibility and control: User-level threads provide more control over thread management, as the thread library is part of the application. This allows for more customization and control over thread scheduling.

Portability: User-level threads can be more easily ported to different operating systems, as the thread library is part of the application.

Disadvantages:

Lower performance: User-level threads rely on the application to manage thread scheduling, which can be less efficient than kernel-level thread scheduling. This can result in lower performance for multithreaded applications.

Limited parallelism: User-level threads are limited to a single processor, as the application has no control over thread scheduling on other processors.

Kernel-level threads Advantages and Disadvantages

Advantages:

Better performance: Kernel-level threads are managed by the operating system, which can schedule threads more efficiently. This can result in better performance for multithreaded applications.

Greater parallelism: Kernel-level threads can be scheduled on multiple processors, which allows for greater parallelism and better use of available resources.

Disadvantages:

Less flexibility and control: Kernel-level threads are managed by the operating system, which provides less flexibility and control over thread management compared to user-level threads.

Less portability: Kernel-level threads are more tightly coupled to the operating system, which can make them less portable to different operating systems.

Hybrid models Advantages and Disadvantages

Advantages:

Combines advantages of both models: Hybrid models combine the advantages of user-level and kernel-level threads, providing greater flexibility and control while also improving performance.

More scalable: Hybrid models can scale to larger numbers of threads and processors, which allows for better use of available resources.

Disadvantages:

More complex: Hybrid models are more complex than either user-level or kernel-level threading, which can make them more difficult to implement and maintain.

Requires more resources: Hybrid models require more resources than either user-level or kernel-level threading, as they require both a thread library and kernel-level support.

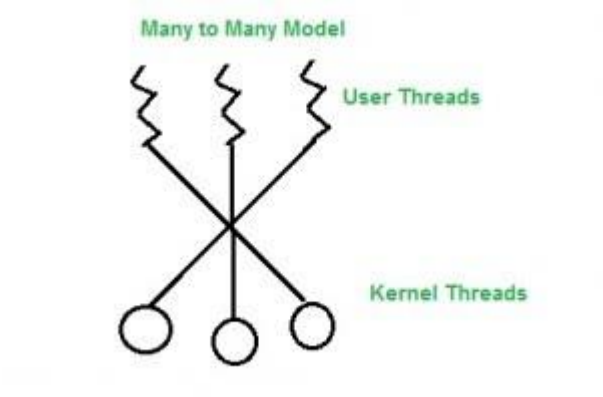
Many to Many Model

1: In this model, we have multiple user threads multiplex to same or lesser number of kernel level threads.

2: Number of kernel level threads are specific to the machine, advantage of this model is if a user thread is blocked we can schedule others user thread to other kernel thread.

3: Thus, System doesn’t block if a particular thread is blocked.

4:It is the best multi threading model.



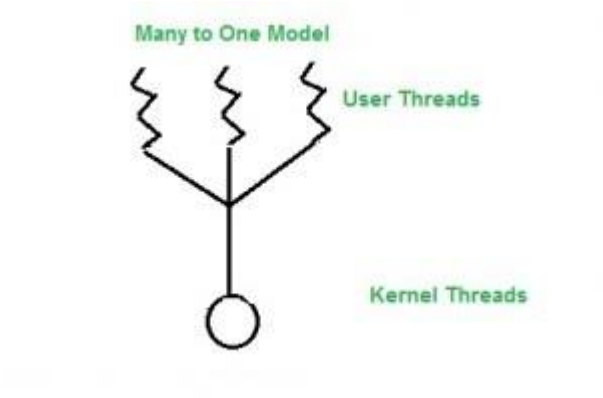
Many to One Model

1: In this model, we have multiple user threads mapped to one kernel thread.

2: In this model when a user thread makes a blocking system call entire process blocks.

3: As we have only one kernel thread and only one user thread can access kernel at a time, so multiple threads are not able access multiprocessor at the same time.

4: The thread management is done on the user level so it is more efficient.



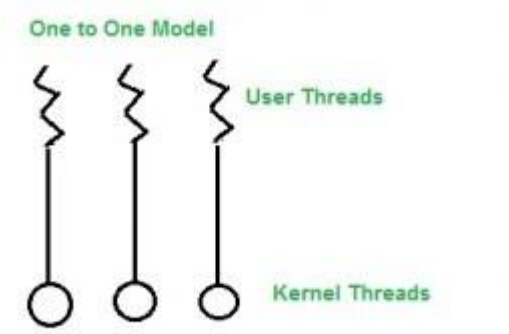
One to One Model

1:In this model, one to one relationship between kernel and user thread.

2: In this model multiple thread can run on multiple processor.

3: Problem with this model is that creating a user thread requires the corresponding kernel thread.

4: As each user thread is connected to different kernel , if any user thread makes a blocking system call, the other user threads won't be blocked.



Unit 5 (Inter Process Communication)

The Producer-Consumer Problems

1: The Producer-Consumer problem is a classic problem this is used for multi-process synchronization i.e. synchronization between more than one processes In the producer consumer problem, there is one Producer that is producing something and there is one Consumer that is consuming the products produced by the Producer.
2: The producers and consumers share the same memory buffer that is of fixed-size.
3: The job of the Producer is to generate the data, put it into the buffer, and again start generating data.
4: While the job of the Consumer is to consume the data from the buffer.
What's the problem here?
The following are the problems that might occur in the Producer-Consumer:

1: The producer should produce data only when the buffer is not full. If the buffer is full, then the producer shouldn't be allowed to put any data into the buffer.
2: The consumer should consume data only when the buffer is not empty. If the buffer is empty, then the consumer shouldn't be allowed to take any data from the buffer.
3: The producer and consumer should not access the buffer at the same time.

What's the solution? The above three problems can be solved with the help of semaphores. In the producer consumer problem, we use three semaphore variables:

1. Semaphore S: This semaphore variable is used to achieve mutual exclusion between processes. By using this variable, either Producer or Consumer will be allowed to use or access the shared buffer at a particular time. This variable is set to 1 initially.
2. Semaphore E: This semaphore variable is used to define the empty space in the buffer. Initially, it is set to the whole space of the buffer i.e. "n" because the buffer is initially empty.
3. Semaphore F: This semaphore variable is used to define the space that is filled by the producer. Initially, it is set to "0" because there is no space filled by the producer initially.

The following is the pseudo-code for the producer:

```
void producer() {
while(T) {
produce()
wait(E)
wait(S)
append()
signal(S)
signal(F)
```

```
}
}
```
The above code can be summarized as:

· while() is used to produce data, again and again, if it wishes to produce, again and again.

· produce() function is called to produce data by the producer.

· wait(E) will reduce the value of the semaphore variable "E" by one i.e. when the producer produces something then there is a decrease in the value of the empty space in the buffer. If the buffer is full i.e. the vale of the semaphore variable "E" is "0", then the program will stop its execution and no production will be done.

· wait(S) is used to set the semaphore variable "S" to "0" so that no other process can enter into the critical section.

· append() function is used to append the newly produced data in the buffer.

· signal(s) is used to set the semaphore variable "S" to "1" so that other processes can come into the critical section now because the production is done and the append operation is also done.

· signal(F) is used to increase the semaphore variable "F" by one because after adding the data into the buffer, one space is filled in the buffer and the variable "F" must be updated.

The following is the code for the consumer:

```
while(T) {
wait(F)
wait(S)
take()
signal(S)
signal(E)
use()
}
}
```
The above code can be summarized as:

· while() is used to consume data, again and again, if it wishes to consume, again and again.

· wait(F) is used to decrease the semaphore variable "F" by one because if some data is consumed by the consumer then the variable "F" must be decreased by one. wait(S) is used to set the semaphore variable "S" to "0" so that no other process can enter into the critical section.

· take() function is used to take the data from the buffer by the consumer. signal(S) is used to set the semaphore variable "S" to "1" so that other processes can come into the critical section now because the consumption is done and the take operation is also done.

· signal(E) is used to increase the semaphore variable "E" by one because after taking the data from the buffer, one space is freed from the buffer and the variable "E" must be increased.
· use() is a function that is used to use the data taken from the buffer by the process to do some operation.

Race condition

1: Race conditions occur when multiple processes or threads access shared data simultaneously, leading to unpredictable outcomes due to timing.

2: Operating systems manage shared resources, like memory or files, and must coordinate processes' operations to avoid conflicts.

3: Inter Process Communication (IPC) facilitates process communication.

4: A scenario illustrates a race condition: two processes, A and B, increment a shared variable n in a file.

5: If interruptions occur during the steps, n might end up with unexpected values due to unsynchronized access.

6: This applies to threads too, like kernel and user threads, extending to Java's thread safety.

7: Preventing race conditions requires ensuring processes complete critical steps separately to maintain data integrity.

Shielding interruption

1: In the earlier example, process B accessed the critical section due to process A's interruption therein.

2: To prevent this, an approach could be having process A disable interrupts upon entering the critical section, only responding after leaving it.

3: However, this notion has limitations.

4: With multiple CPUs, process A can't disable interrupts across all CPUs, allowing other CPUs to access the critical section.

5: Additionally, there are power and system stability concerns.

6: Granting user processes interrupt-disabling power could lead to system hang-ups, as masking interrupts entirely could compromise the operating system's functionality. Thus, while the concept is sound, its practical implementation faces complexities.

Mutual Exclusion and Critical Zone

To prevent race conditions, it's essential to establish mutual exclusion, ensuring that when one process accesses shared data, others cannot simultaneously do the same. For instance, a critical section can be defined around the program fragment encompassing steps 1-3. This signifies a sensitive area where shared data or files are manipulated. By controlling access to this critical section, and ensuring that multiple processes cannot enter it concurrently, race conditions can be averted. This approach guarantees that conflicting operations won't overlap, effectively safeguarding data integrity.

An example to understand race conditions in real life

I have $500 in my account. Someone transfers $200 to me at the same time that I withdraw $50.

Now, if the bank doesn't handle race conditions properly, they will do the following (assuming the transactions are handled manually, of course) Clerk A will see the request to add $200 to my balance, and note that my balance is currently $500. Clerk B will see the request to subtract $50 from my balance, and note that my balance is currently $500 (clerk A hasn't yet transferred the money).

Clerk A finishes the paperwork and sets my account balance to $700 (500 + the 200 he was supposed to add). And then, a minute later (because clerk B just had to grab a cup of coffee), clerk B finishes up the other transaction and sets my balance to $450 (the 500 I had when he checked, minus the 50 he was meant to subtract).

My balance is now $450, when it should have been $650, because of a race condition. The outcome depended on the order in which different parts of the two transactions were performed.

That's the general description of how race conditions are bad. Now say that instead of clerks, we have our application processing two separate tasks at the same time (that's

your 'threads of execution'), and just like above, they both read a value, modify the value that they read, and then write it back. One of the modifications may then be lost if this happens in the order shown above. That should relate it to the specific problems in your app.

Lock variables

1: The use of lock variables can address race conditions.

2: By initializing a lock flag to 0, a process intending to enter a critical section checks the lock's value.

3: If it's 0, the process sets it to 1, enters the critical section, and resets it to 0 upon exit.

4: However, a problem remains—race conditions may occur.

5: For instance, two processes might read the lock as 0 simultaneously.

6: Before one sets it to 1, the other process does the same, allowing both to access the critical section concurrently.

7: This highlights the need for more sophisticated synchronization mechanisms to ensure exclusive access and prevent such conflicts.

Peterson algorithm

1: Peterson’s Algorithm synchronizes two processes using two variables: a boolean array 'flag' of size 2 and an integer 'turn.'

2: For the Consumer (i) and Producer (j), the flags start as false.

3: To access their critical sections, a process sets its flag to true and 'turn' to the other process's index, indicating willingness to yield control.

4: Busy waiting occurs until the other process completes its critical section.

5: The current process then enters its critical section, modifying a shared buffer.

6: Afterward, it sets its flag to false, signaling disinterest.

7: The program operates for a defined duration, adjustable via the RT macro value.

TSL directive

1: The Test-and-Set-And-Clear (TSL) directive, a synchronization primitive in operating systems and concurrent programming, guarantees mutual exclusion for shared resources.

2: Working on a memory location, often called a "lock" or "flag," it's used to implement locks and semaphores.

3: TSL involves two key operations: Test-and-Set (TAS), which reads the memory value and sets it atomically, blocking concurrent modifications; and Clear, which resets the memory value after resource use.

4: TAS prevents simultaneous reads or changes, aiding data integrity.

5: Clear allows subsequent processes to access shared resources.

6: However, modern synchronization options, like atomic operations and mutexes, offer efficiency improvements over TSL.

Be busy waiting

1: In fact, the Peterson algorithm and TSL method mentioned above all have one characteristic: they are busy waiting when they are waiting to enter the critical region, which we often call spin.

2: Its disadvantage is that it wastes CPU time slices and can lead to priority inversion Questions.

3: Consider that a computer has two processes, H priority is higher and L priority is lower.

4: Scheduling rules stipulate that H can run as long as H is in a ready state.

5: At some point, L is in the critical region, and H is in the ready state, ready to run.

6: But H needs to be busy waiting, while L can not be scheduled because of its low priority, so it can not leave the critical area, so H will always be busy waiting, while L

can not leave the critical area. This situation is called priority inversion problem（ priority inversion problem）

Blocking and awakening of process

1: The operating system provides some primitives, sleep and wakeup.

2: The process or function provided by the kernel to an out-of-core call becomes a primitive, and the primitive is not allowed to interrupt during execution.

3: Sleep is a system call that calls a blocked process until another process calls the wakeup method, which takes the blocked process as a parameter and wakes it up.

4: The biggest difference between blocking and busy waiting is that when a process is blocked, the CPU will not allocate time slices to it, while busy waiting is always idle, consuming time slices of the CPU.

Semaphores

Semaphores are integer variables that are used to solve the critical section problem by using two atomic operations, wait and signal that are used for process synchronization.

Wait

The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

```
wait(S)
{
   while (S<=0);
   S--;
}
```

Signal
The signal operation increments the value of its argument S.

```
signal(S)
{
   S++;
}
```

Types of Semaphores

There are two main types of semaphores i.e. counting semaphores and binary semaphores. Details about these are given as follows −

Counting SemaphoresThese are integer value semaphores and have an unrestricted value domain. These semaphores are used to coordinate the resource access, where the semaphore count is the number of available resources. If the resources are added, semaphore count is automatically incremented and if the resources are removed, the count is decremented.

Binary SemaphoresThe binary semaphores are like counting semaphores but their value is restricted to 0 and 1. The wait operation only works when the semaphore is 1 and the signal operation succeeds when semaphore is 0. It is sometimes easier to implement binary semaphores than counting semaphores.

Advantages of semaphore

1: The mutual exclusion principle is followed when you use semaphores because semaphores allow only one process to enter into the critical section.

2: Here, you need not verify that a process should be allowed to enter into the critical section or not. So, processor time is not wasted here.

Disadvantages of semaphore

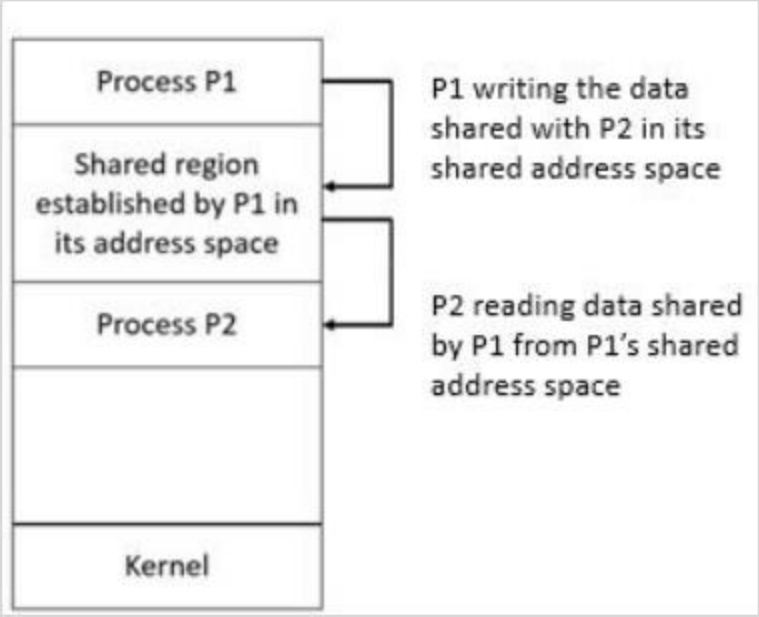
1: While using semaphore, if a low priority process is in the critical section, then no other higher priority process can get into the critical section. So, the higher priority process has to wait for the complete execution of the lower priority process.

2: The wait() and signal() functions need to be implemented in the correct order. So, the implementation of a semaphore is quite difficult.

Shared Memory Method

In the Shared Memory system, the cooperating processes communicate, to exchange data with each other. Because of this, the cooperating processes establish a shared region in their memory. The processes share data by reading and writing the data in the shared segment of the processes.

Let us discuss it by considering two processes. The diagram is shown below:



Let the two cooperating processes P1 and P2. Both the processes P1 and P2, have their different address spaces. Now let us assume, P1 wants to share some data with P2. So, P1 and P2 will have to perform the following steps −

Step 1 − Process P1 has some data to share with process P2. First P1 takes initiative and establishes a shared memory region in its own address space and stores the data or information to be shared in its shared memory region.

Step 2 − Now, P2 requires the information stored in the shared segment of P1. So, process P2 needs to attach itself to the shared address space of P1. Now, P2 can read out the data from there.

Step 3 − The two processes can exchange information by reading and writing data in the shared segment of the process.

Message passing method

Message Passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space.

For example − chat programs on World Wide Web.

Now let us discuss the message passing step by step.

Step 1 − Message passing provides two operations which are as follows −

Send message

Receive message

Messages sent by a process can be either fixed or variable size.

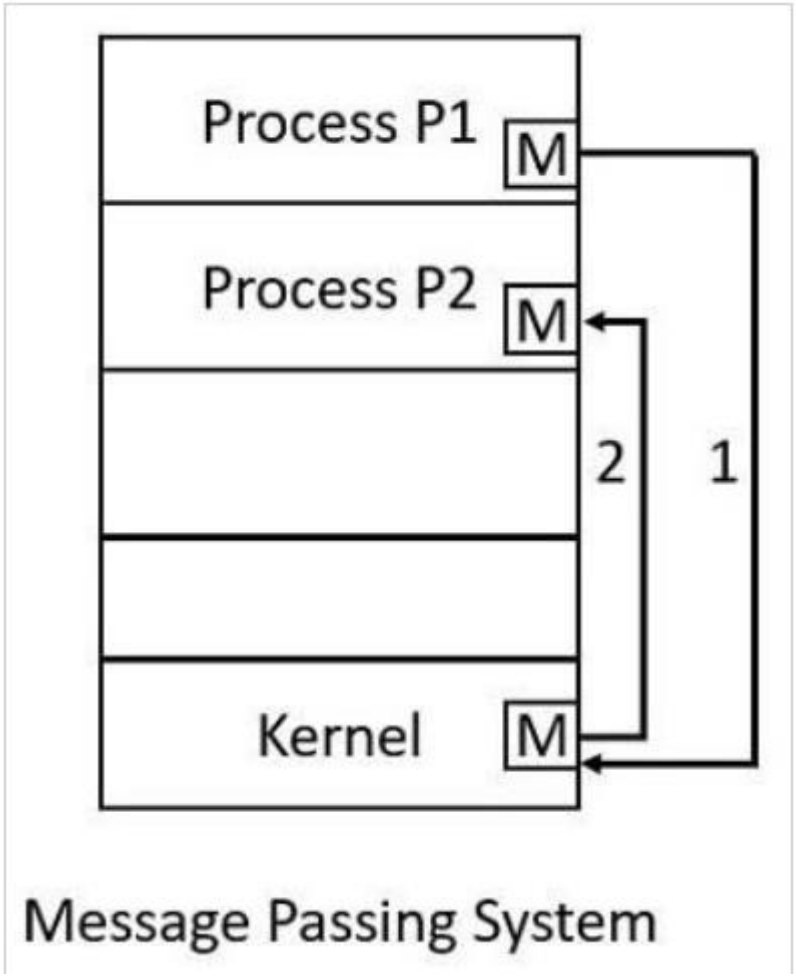
Step 2 − For fixed size messages the system level implementation is straight forward. It makes the task of programming more difficult.

Step 3 − The variable sized messages require a more system level implementation but the programming task becomes simpler.

Step 4 − If process P1 and P2 want to communicate they need to send a message to and receive a message from each other that means here a communication link exists between them.

Step 5 − Methods for logically implementing a link and the send() and receive() operations.

Given below is the structure of message passing technique −



Message Passing through Communication Link.

1: Message passing via communication links involves transmitting data or messages between processes in distributed or parallel computing.

2: This occurs through channels like sockets, pipes, or message queues, ensuring synchronized, reliable data exchange.

3: Unlike shared memory, it's suited for separate processes or machines, enhancing scalability and fault tolerance.

4: Messages carry information, instructions, or data, aiding coordination and resource sharing.

5: This approach fosters decentralized, efficient parallel processing and modular design, vital for distributed computing systems.

Message Passing through Exchanging the Messages.

1: Message passing through message exchange in an operating system involves processes sharing data or instructions by sending and receiving messages.

2: This communication method relies on predefined mechanisms like message queues, channels, or sockets.

3: Processes transmit messages, which are then received, processed, and acted upon by the intended recipient.

4: This enables synchronization, resource sharing, and collaboration in distributed or parallel computing environments.

5: It's particularly valuable when processes need to operate independently on separate machines while still needing to interact and coordinate effectively.

Examples of IPC systems

1. Posix : uses shared memory method.

2. Mach : uses message passing

3. Windows XP : uses message passing using local procedural calls

---

Bounded Buffer Problem

1: The Bounded Buffer Problem is a classic synchronization challenge in concurrent programming, involving the coordination of producer and consumer processes sharing a fixed-size buffer or queue.

2: The goal is to ensure that producers don't overflow the buffer and consumers don't underflow it, preventing data loss or access conflicts.

3: Producers add items to the buffer when it's not full, and consumers retrieve items when the buffer is not empty.

4: Synchronization mechanisms like semaphores or mutexes are used to control access to the buffer, preventing simultaneous access by multiple processes and avoiding race conditions.

solution

Assume that there are n buffers, each capable of holding a single item. We use three semaphores: empty and full to count the empty and full buffers and mutex to provide mutual exclusion for operations on the buffer pool.

mutex is initialized to 1, empty is initialized to n and full is initialized to 0.

The Dining Philosopher's problem

1: The dining philosophers problem states that there are 5 philosophers sharing a circular table and they eat and think alternatively.

2: There is a bowl of rice for each of the philosophers and 5 chopsticks.

3: A philosopher needs both their right and left chopstick to eat.

4: A hungry philosopher may only eat if there are both chopsticks available. **5:** Otherwise a philosopher puts down their chopstick and begin thinking again.

Solution to Dining Philosopher's problem

A solution to the Dining Philosophers Problem is to use a semaphore to represent a chopstick. A chopstick can be picked up by executing a wait operation on the semaphore and released by executing a signal semaphore.

The structure of the chopstick is shown below −

semaphore chopstick [5];

Initially the elements of the chopstick are initialized to 1 as the chopsticks are on the table and not picked up by a philosopher.

The structure of a random philosopher i is given as follows −

```
do {
```

```
   wait( chopstick[i] );
   wait( chopstick[ (i+1) % 5] );
   . .
   . EATING THE RICE
   .
   signal( chopstick[i] );
   signal( chopstick[ (i+1) % 5] );
   .
   . THINKING
   .
} while(1);
```

In the above structure, first wait operation is performed on chopstick[i] and chopstick[(i+1) % 5]. This means that the philosopher i has picked up the chopsticks on his sides. Then the eating function is performed.
After that, signal operation is performed on chopstick[i] and chopstick[(i+1) % 5]. This means that the philosopher i has eaten and put down the chopsticks on his sides. Then the philosopher goes back to thinking.

The readers-writers problem

1: The readers-writers problem relates to an object such as a file that is shared between multiple processes.

2: Some of these processes are readers i.e. they only want to read the data from the object and some of the processes are writers i.e. they want to write into the object.

3: The readers-writers problem is used to manage synchronization so that there are no problems with the object data.

4: For example - If two readers access the object at the same time there is no problem.

5: However, if two writers or a reader and writer access the object at the same time, there may be problems.

solution

1: The Readers-Writers Problem involves coordinating multiple readers and writers accessing a shared resource.

2: To address this, a solution employs semaphores: a "Mutex" semaphore ensures exclusive access for both readers and writers, and a "ReaderCount" semaphore keeps track of the number of readers.

3: Writers gain exclusive access by waiting on the "Mutex" semaphore.

4: Readers modify the "ReaderCount" under "Mutex" protection, allowing multiple readers simultaneous access.

5: This solution maintains data integrity and synchronization, though advanced solutions can address potential issues like writer starvation.

6: Careful implementation of the solution is essential to prevent conflicts and ensure fair resource utilization.

Peterson’s Algorithm in Process Synchronization

Peterson's Algorithm is a classic approach to achieving mutual exclusion in process synchronization. It's designed for scenarios with two processes, typically referred to as "P0" and "P1," that share a critical section of code. This algorithm ensures that only one process can access the critical section at a time, preventing race conditions.

Here's a concise description of Peterson's Algorithm:

Initialization: Initialize two flags, flag[2], where flag[i] is set to true when process i wants to enter the critical section. Also, set turn to the opposite of the current process.

Entering the Critical Section: When a process (P0 or P1) wants to enter the critical section, it sets its flag to true and lets the other process have the turn.

Checking for Conflicts: Before entering the critical section, a process checks if the other process is interested (flag[j] is true) and has the turn (turn == j). If both conditions are met, the process waits until the other process exits the critical section.

Exiting the Critical Section: After leaving the critical section, the process sets its flag to false, allowing the other process to enter.

Unit 6 (I/O Management and Deadlock)

I/O software

I/O software is often organized in the following layers −

1: User Level Libraries − this provides simple interface to the user program to perform input and output. For example, stdio is a library provided by C and C++ programming languages.

2: Kernel Level Modules − this provides device driver to interact with the device controller and device independent I/O modules used by the device drivers

3: Hardware − this layer includes actual hardware and hardware controller which interact with the device drivers and makes hardware alive.

4: A key concept in the design of I/O software is that it should be device independent where it

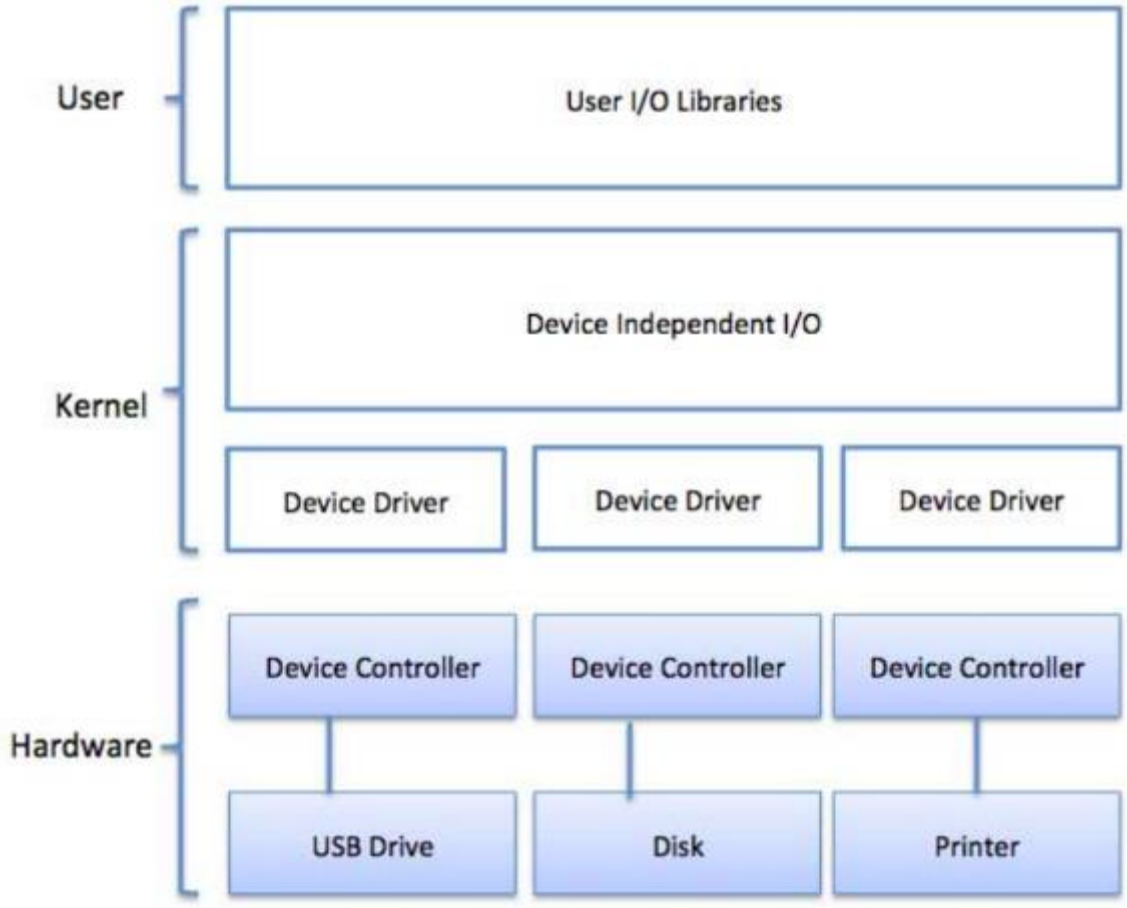


Fig. Operating System interface

Device Drivers

1: Device drivers are software modules that can be plugged into an OS to handle a particular device.

2: Operating System takes help from device drivers to handle all I/O devices.

3: Device drivers encapsulate device-dependent code and implement a standard interface in such a way that code contains device-specific register reads/writes. Device driver, is generally written by

the device's manufacturer and delivered along with the device on a CD-ROM.

A device driver performs the following jobs −

- To accept request from the device independent software above to it.
- Interact with the device controller to take and give I/O and perform required error handling.
- Making sure that the request is executed successfully.

Interrupt handlers

1: An interrupt handler, also known as an ISR, is a software callback within an OS or device driver triggered by hardware interrupts.

2: Interrupts are used in computer systems to handle events that need immediate attention, such as hardware events, input from devices, or errors.

3: Upon activation, the handler manages the interrupt's tasks, updates data structures, and awakens processes awaiting specific events.

4: An address, found in an interrupt vector table, directs the CPU to the appropriate handler.

5: By efficiently responding to external events without constant monitoring, interrupt handlers enhance system performance and responsiveness in multitasking environments.

Device-Independent I/O Software

1: 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.

2: Though it is difficult to write completely device independent software but we can write some modules which are common among all the devices.

3: Following is a list of functions of device-

independent I/O Software −

1. Uniform interfacing for device drivers.

2. Device naming - Mnemonic names mapped to Major and Minor device numbers.

3. Device protection.

4. Providing a device-independent block size.

5. Buffering because data coming off a device cannot be stored in final destination.

6. Storage allocation on block devices.

7. Allocation and releasing dedicated devices.

8. Error Reporting.

User-Space I/O Software

1: These are the libraries which provide richer and simplified interface to access the functionality of the kernel or ultimately interact with the device drivers.

2: Most of the user- level I/O software consists of library procedures with some exception like spooling system which is a way of dealing with dedicated I/O devices in a multiprogramming system.

3: I/O Libraries (e.g., stdio) are in user-space to provide an interface to the OS resident device-independent I/O SW.

4: For example putchar(), getchar(), printf() and scanf() are example of user level I/O library studio available in C programming.

Kernel I/O Subsystem

1: The Kernel I/O Subsystem manages various I/O services.

2: It handles scheduling by arranging I/O requests for optimal execution order, enhancing system efficiency and response time.

3: Buffering involves using a memory buffer to accommodate speed differences in data transfer between devices or applications.

4: Caching is maintained with cached copies in fast memory for efficient data access.

5: Spooling stores output for devices like printers, transferring queued files sequentially.

6: Error handling in protected memory OS guards against hardware and app errors.

7: Overall, the subsystem optimizes I/O operations for improved system performance and error resilience.

Terminal I/O(Terminal Hardware, Terminal):

An essential role of an Operating System (OS) is managing diverse I/O devices like keyboards, disks, printers, USB devices, and network connections. The OS handles application I/O requests, transmitting them to physical devices and relaying responses back to applications. I/O devices fall into two categories:

1. Block Devices: These devices communicate via entire blocks of data. Examples include hard disks, USB cameras, and disk-on-key devices. The OS manages the communication and data transfer involving these block devices.

2. Character Devices: These devices use single characters (bytes) for communication. Serial and parallel ports, sound cards, and similar devices fall into this category. The OS facilitates the exchange of individual characters with these devices.

Device Controllers

1: Device drivers are software modules that enable an OS to manage specific devices, assisting with I/O operations.

2: Device controllers serve as intermediaries between devices and drivers, with I/O units having mechanical and electronic components, the latter termed the controller.

3: Each device necessitates both a controller and driver for OS interaction.

4: Controllers can manage multiple devices, functioning as interfaces that convert serial bits to byte blocks and perform error correction.

5: Devices link to computers via plug and socket, connected to a controller.

6: A model integrates CPU, memory, controllers, and I/O devices through a shared communication bus.

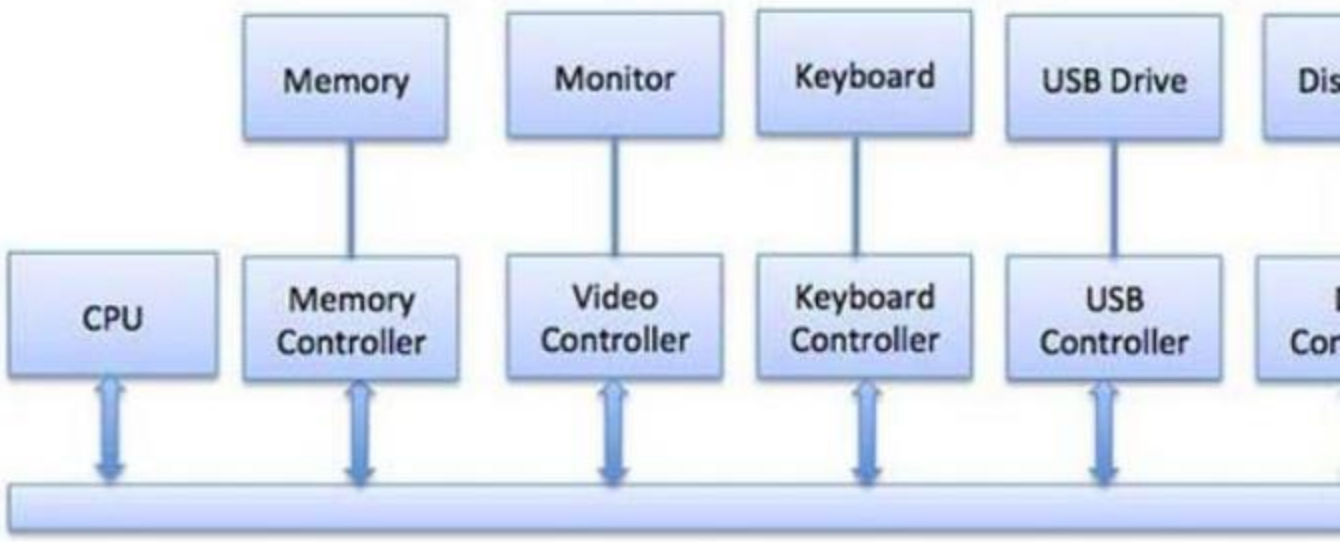


Fig. Device Controllers

Synchronous vs asynchronous I/O

Synchronous I/O involves a blocking mechanism, where a program waits for the I/O operation to complete before moving on to other tasks. The program halts its execution until the I/O operation finishes, which can potentially lead to inefficiencies if the operation takes a significant amount of time. Synchronous I/O is straightforward but can result in reduced overall system performance when dealing with multiple I/O operations.

Asynchronous I/O employs a non-blocking approach. The program initiates an I/O operation and continues executing without waiting for the operation to finish. It uses callbacks, events, or notifications to handle the completion of the I/O operation. This approach is more efficient when dealing with multiple I/O operations simultaneously.

Memory mapped I/O

1: When using memory-mapped I/O, the same address space is shared by memory and I/O devices.

2: The device is connected directly to certain main memory locations so that I/O device can transfer block of data to/from memory without going through CPU.

3: memory mapped IO, OS allocates buffer in memory and informs I/O device to use that buffer to send data to the CPU.

4: I/O device operates asynchronously with CPU, interrupts CPU when finished.

5: The advantage to this method is that every instruction which can access memory can be used to manipulate an I/O device.

6: Memory mapped IO is used for most high-speed I/O devices like disks, communication interfaces.

Direct Memory Access

1. DMA stands for direct memory access, which is a computer system feature that allows hardware subsystems to access the main system memory independently of the CPU.

2. DMA allows data to be sent directly from an attached device to the computer's main memory, freeing up the CPU from involvement with the data transfer.

3. This speeds up overall computer operation and enhances system performance by offloading data transfer tasks from the CPU.

4. The process is managed by a chip known as a DMA controller (DMAC).

5. DMA can also allow the network device to move packet data directly to the system's memory, reducing CPU utilization.

Interrupt and Polling I/O

Interrupt: Interrupt is a hardware mechanism in which, the device notices the CPU that it requires its attention. Interrupt can take place at any time. So when CPU gets an interrupt signal through the indication interrupt-request line, CPU stops the current process and respond to the interrupt by passing the control to interrupt handler which services device.

Polling is a protocol, not hardware, where the CPU checks if an I/O device needs attention. The device's "command-ready" bit indicates whether a command is ready to be read by hardware (bit=1) or not (bit=0), enabling the CPU to process accordingly.

Difference between Interrupt and Polling

1. The main difference between interrupt and polling is how the CPU and system communicate:

2. The system informs the CPU that it needs attention.

3. Interrupts can happen at any time.

4. When the CPU receives an interrupt signal, it stops the current process and passes control to the interrupt handler.

5. The CPU constantly inspects the status of the system to find whether it needs attention.

6. Polling is a protocol that allows the CPU to check if a device needs attention.

7. In summary, interrupts are event-driven and allow the processor to respond immediately to external events, while polling involves the processor actively checking the status of devices at regular intervals.

Reading a CD

Reading a CD is done by shining a laser at the discand detecting changing reflections

patterns.

1 = change in height (land to pit or pit to land)

0 = a ―fixed‖ amount of time between 1‘s

Data in Cd-ROM

1: The CD-ROM's fundamental data unit is a frame, with 24 bytes containing input data.

2: One byte occupies 17 bits, including 14 bits for source data and 3 merge bits.

3: Error correction uses 180 bits.

4: A frame holds 588 bits.

5: Frames compose sectors, with 2352 data bytes (98 frames) and 882 bytes for error correction.

6: CD surfaces have 17,000+ tracks, each holding 32 sectors.

7: Sectors include user data and service info totaling 9996 bits, divided into 49 segments of 204 bits.

8: Key segments mark sector start, clock adjustment, service data, while segments 4-48 store data.

9: Error correction ensures 10-10 bit error probability.

10: Segments 4 through 48 are used for data.

General principles of writing/reading

1: CD recording and reading processes rely on geometric phenomena (light reflection, absorption, transmission, refraction) and wave optics (light interference, diffraction, polarization).

2: A laser alters a CD's working layer to create a signal pattern representing logical values.

3: Optical characteristics change, and during reading, the laser interprets these changes as a digital signal.

4: Data density is determined by the laser's wavelength and label size.

5: Optical methods include ablation (single-write, laser heats material to remove it), bubble (expanded recording layer scatters reading laser), and multi-recording (laser alters phase/color for bit data).

6: These methods enhance data density, utilizing a transparent substrate for protection.

An introduction to the DVD

1: The DVD, a high capacity optical medium developed in the mid-1990s by companies like Philips and Sony, is poised to replace CD-ROMs and laser disks, possibly even VHS tapes for videos.

2: It shares a similar flat disk design with CDs, measuring 4.7 inches in diameter and 0.05 inches thick.

3: Data is stored in narrow spiral tracks using a shorter wavelength laser beam than CDs, allowing for increased storage capacity.

4: DVD drives, available in various versions, may read and write discs and often include MPEG-2 decoders for optimal DVD video playback.

5: Despite not replacing hard disks, DVDs are positioned to succeed CD-ROMs.

DVD RAM

1: Three writable technologies are present at the market: Pioneer has a DVD-Recordable technology placing 3.95 GB per disk. DVD-RAM is a RW-disk from Hitachi and Matsushiti.

2: The 1. generation disks hold 3.6 GB, while the 2. generation hold 4.7 GB.

3: The so-called DVD+RW, supported by HP, Sony, Philips, Yamaha, Ricoh and Mitsubishi holds up to 4.7 GB per disk.

4: None of the three products are compatible. However, the companies behind DVD+RW control 75% of the market.

5: It appears that the DVD-RAM disks are extremely sensitive to greasy fingers and other contaminants. Therefore they must be handled in special cassettes.

Deadlocks

1: A deadlock happens in an operating system when two or more processes need some resource to complete their execution that is held by the other process.

2: A deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process.

3: In the below diagram, the process 1 has resource 1 and needs to acquire resource 2.

4: Similarly process 2 has resource 2 and needs to acquire resource 1. Process 1 and process 2 are in deadlock as each of them needs the other‘s resource to complete their execution but neither of them is willing to relinquish their resources.

Coffman Conditions

A deadlock occurs if the four Coffman conditions hold true. But these conditions are not mutually exclusive. The Coffman conditions are given as follows −

Mutual ExclusionThere should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.

Hold and WaitA process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.

No PreemptionA resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.

Circular Wait

A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a

resource held by the first process. This forms a circular chain. For example: Process 1 is allocated to Resource 2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.

Deadlock Detection

1: A deadlock can be detected by a resource scheduler as it keeps track of all the resources that are allocated to different processes.

2: After a deadlock is detected, it can be resolved using the

following methods −

a: All the processes that are involved in the deadlock are terminated. This is not a good approach as all the progress made by the processes is destroyed.

b: Resources can be preempted from some processes and given to others till the deadlock is resolved.

Unit 7 (Memory Management)

Memory management

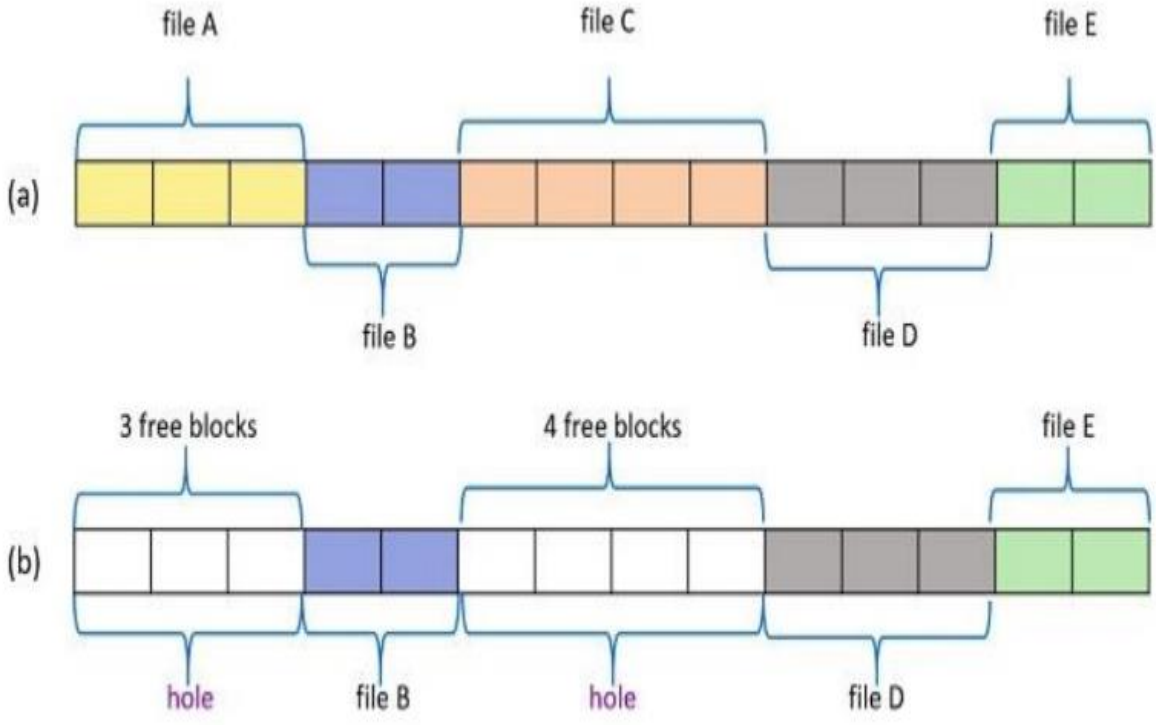
1: Memory management involves allocating, deallocating, and organizing a computer system's memory resources.
2: It assigns memory to processes, manages protection between them, and utilizes techniques like paging, segmentation, and virtual memory to optimize memory usage.
3: Effective memory management prevents conflicts, ensures efficient multitasking, and safeguards against memory leaks.
4: It addresses issues like fragmentation by compacting or swapping data to disk.
5: While not replacing hard disks, memory management plays a crucial role in maintaining system stability, performance, and responsiveness by efficiently distributing and coordinating memory for various processes and programs.

Contiguous Memory Allocation

1: Contiguous memory allocation is a memory allocation method that allocates a single contiguous section of memory to a process or a file.

2: This method takes into account the size of the file or a process and also estimates the maximum size, up to what the file or process can grow.

3: Taking into account the future growth of the file and its request for memory, the operating system allocates sufficient contiguous memory blocks to that file. **4:** Considering this future expansion and the file‘s request for memory, the operating system will allocate those many contiguous blocks of memory to that file.



(a) Contiguous memory allocation of 5 files
(b) When the file A and C terminates and release the memory creating hole

---

How holes are created in the memory?

1: Holes in memory, termed memory holes or gaps, emerge due to memory allocation and deallocation processes.

2: External fragmentation arises as memory blocks are allocated and freed, leaving scattered small gaps that hinder contiguous memory allocation.

3: Internal fragmentation results from memory blocks being larger than necessary, causing wasted space within allocated regions.

4: Both forms of fragmentation contribute to inefficient memory utilization, complicating the allocation of contiguous memory blocks.

5: Techniques such as memory compaction and virtual memory help alleviate these issues, optimizing memory management and enhancing overall system performance.

Block Allocation List

1: Block allocation list maintains two tables.

2: One table contains the entries of the blocks that are allocated to the various files.

3: The other table contains the entries of the holes that are free and

can be allocated to the process in the waiting queue.

4: Now, as we have entries of free blocks which one must be chosen can be decided using either of these strategies: first-fit, best-fit, worst-fit strategies.

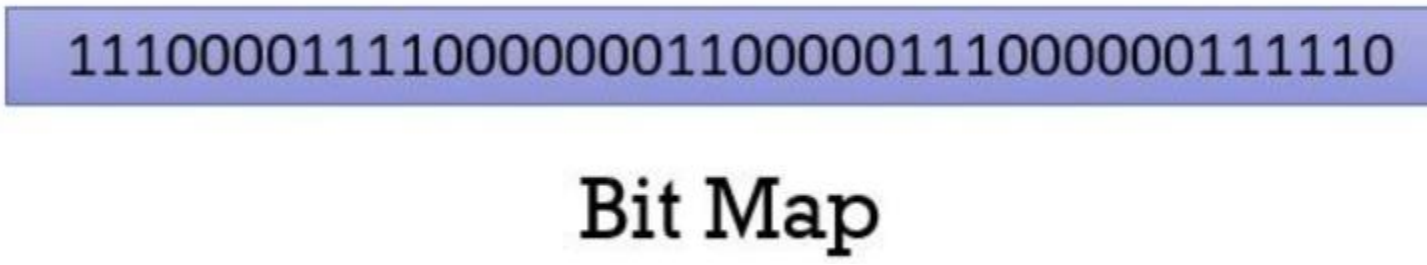
First-fit

Here, the searching starts either at the beginning of the table or from where the previous first- fit search has ended. While searching, the first hole that is found to be large enough for a process to accommodate is selected.

Best-fitThis method needs the list of free holes to be sorted according to their size. Then the smallest hole that is large enough for the process to accommodate is selected from the list of free holes. This strategy reduces the wastage of memory as it does not allocate a hole of larger size which leaves some amount of memory even after the process accommodates the space.

Worst-fitThis method requires the entire list of free holes to be sorted. Here, again the largest hole among the free holes is selected. This strategy leaves the largest leftover hole which may be useful for the other process.

Bit MapThe bit map method only keeps track of the free or allocated block. One block is represented by one bit, bit 0 resembles the free block and bit 1 resembles that the block is allocated to a file or a process.



Fragmentation

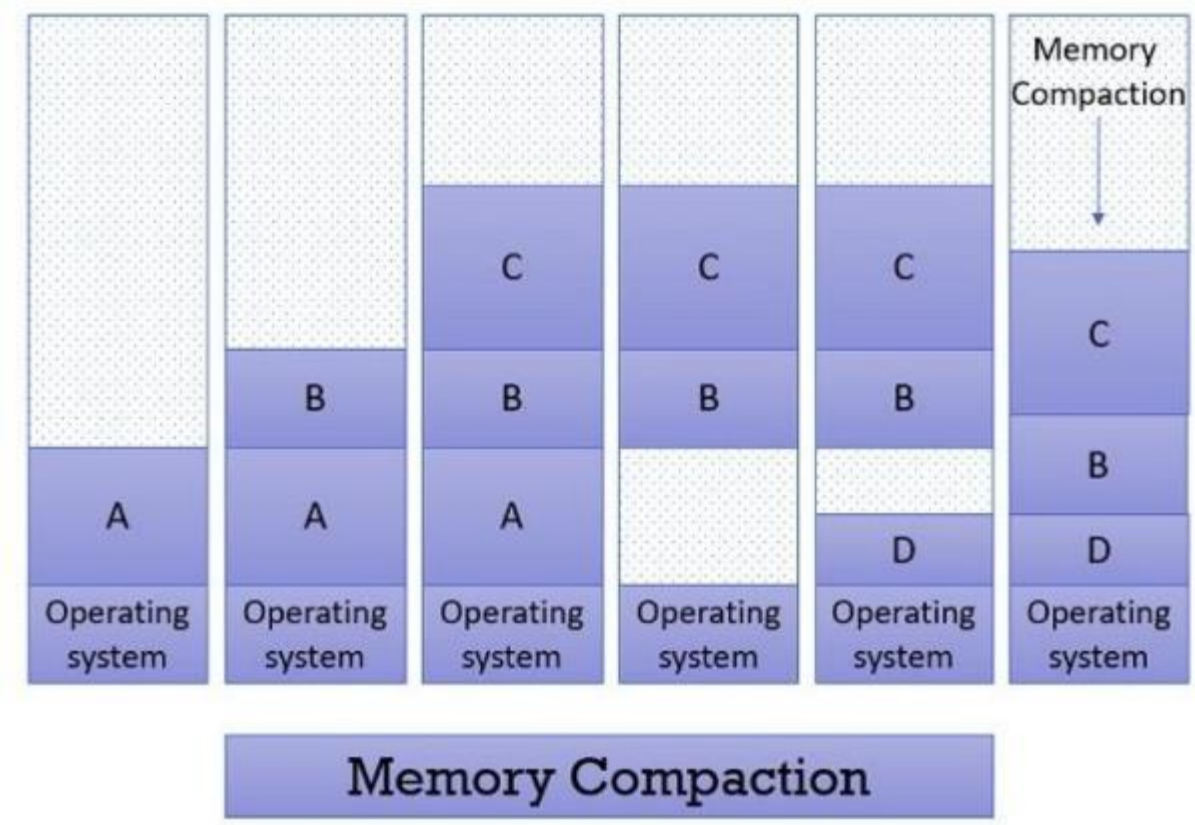
1: Fragmentation can either be external fragmentation or internal fragmentation.

2: External fragmentation is when the free memory blocks available in memory are too small and even non-contiguous.

3: Whereas, the internal fragmentation occurs when the process does not fully utilize the memory allocated to it.

4: The solution to the problem of external fragmentation is memory compaction. Here, all the memory contents are shuffled to the one area of memory thereby creating a large block of memory which can be allocated to one or more new processes.

5: As in the figure below, you can see at the last process C, B, D are moved downwards to make a large hole.



Disadvantages

1: The main disadvantage of contiguous memory allocation is memory wastage and inflexibility.

2: As the memory is allocated to a file or a process keeping in mind that it will grow during the run.

3: But until a process or a file grows many blocks allocated to it remains unutilized.

4: And they even they cannot be allocated to the other process leading to wastage of memory.

5: In case, the process or the file grows beyond the expectation i.e. beyond the allocated memory block, then it will abort with the message —'No disk space' leading to inflexibility.

Advantages

1: The advantage of contiguous memory allocation is it increases the processing speed.

2: As the operating system uses the buffered I/O and reads the process memory blocks consecutively it reduces the head movements.

3: This speed-ups the processing.

Fixed Partitioning

1: This is the oldest and simplest technique used to put more than one process in the main memory.

2: In this partitioning, the number of partitions (non-overlapping) in RAM are fixed but the size of each partition may or may not be the same.

3: As it is contiguous allocation, hence no spanning is allowed.

4: Here partition are made before execution or during system configure.

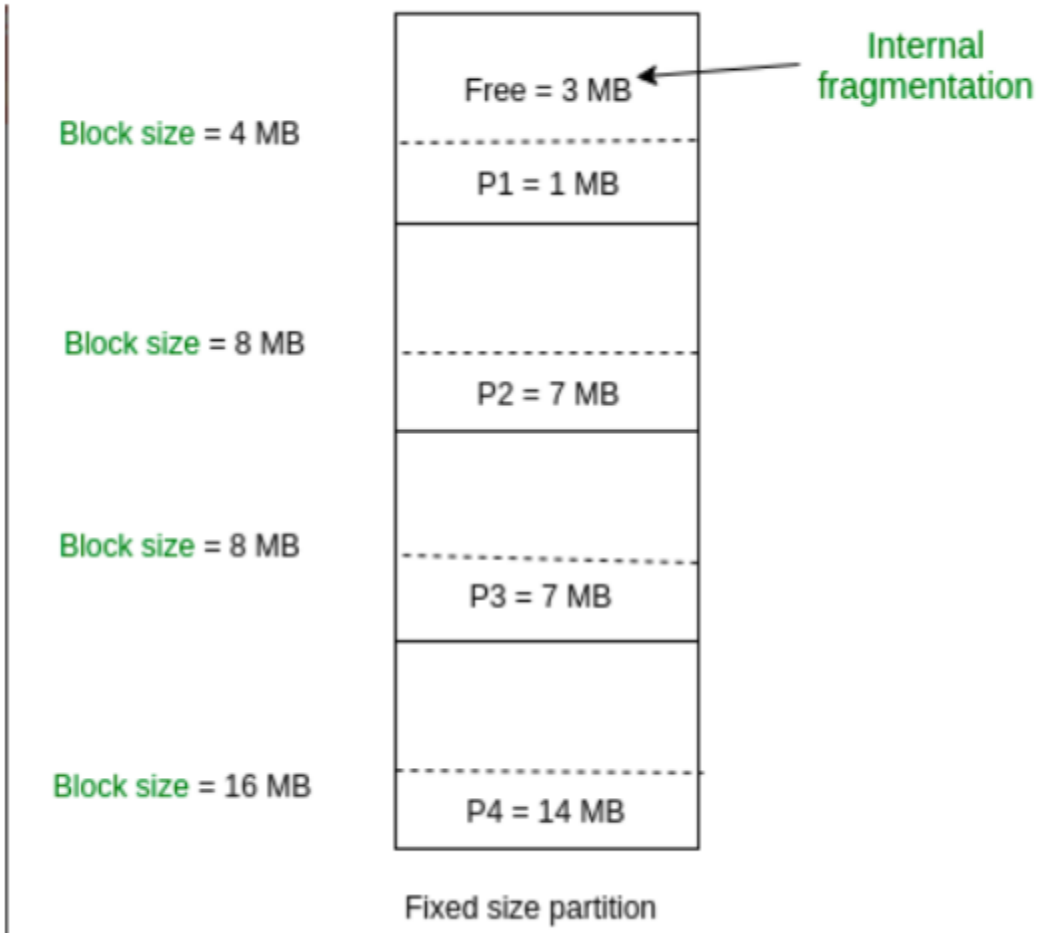
5: As illustrated in below figure, first process is only consuming 1MB out of 4MB in the main memory.

6: Hence, Internal Fragmentation in first block is (4-1) = 3MB.

7: Sum of Internal Fragmentation in every block = (4-1)+(8-7)+(8-7)+(16-14)= 3+1+1+2 = 7MB.

8: Suppose process P5, with a size of 7MB, arrives. However, this process cannot be accommodated despite the presence of available free space due to contiguous allocation (as spanning is not allowed).

9: As a result, the 7MB becomes part of external fragmentation.



Advantages of Fixed Partitioning

1: Algorithms needed to implement Fixed Partitioning are easy to implement. It simply requires putting a process into certain partition without focussing on the emergence of Internal and External Fragmentation.

2: Processing of Fixed Partitioning require lesser excess and indirect computational power.

Disadvantages of Fixed Partitioning

1: Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This can cause internal fragmentation.

2: The total unused space (as stated above) of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form (as spanning is not allowed).

3: Process of size greater than size of partition in Main Memory cannot be

accommodated. Partition size cannot be varied according to the size of incoming

process‘s size. Hence, process size of 32MB in above stated example is invalid.

Dynamic Partitioning

Dynamic partitioning is a computer memory management technique that allows the system to allocate and deallocate memory resources as needed, creating new partitions or resizing existing ones at runtime, in order to optimize memory utilization and improve system performance.

The process of dynamic partitioning involves the following steps −

1: The operating system maintains a table of all memory partitions in the system, including their sizes and status (allocated or free).

2: When a process requires memory, the operating system searches the table for a free partition of the required size.

3: If a free partition of the required size is found, the operating system allocates the partition to the requesting process.

4: If no free partition of the required size is available, the operating system may either split an existing larger partition into two smaller ones, or create a new partition of the required size.

5: When a process no longer needs a memory partition, the operating system deallocates the partition and marks it as free in the memory table.

Relocation

1: Relocation in memory management involves adjusting program addresses during loading to account for varying memory locations.

2: A base address, often stored in a relocation register, serves as a reference point.

3: Relative addresses within the program are added to this base address, resulting in actual physical addresses for execution.

4:This process ensures that multiple processes can coexist in memory without conflicts, enhancing memory protection and system stability.

5: Relocation supports multitasking by allowing processes to be loaded independently and contributes to overall security by preventing unauthorized memory access.

Key Takeaways

1: Memory allocation can be done either by a fixed-sized partition scheme or by variable-sized partition scheme.

2: Block allocation list has three methods to select a hole from the list of free holes first-fit, best-fit and worse-fit.

3: Bit map keeps track of free blocks in memory, it has one bit for one memory block, bit 0 shows that the block is free and bit 1 shows the block is allocated to some file or a process.

4: Contiguous memory allocation leads to fragmentation. Further fragmentation can either be external or internal.

5: Contiguous memory allocation leads to memory wastage and inflexibility. If the operating system uses buffered I/O during processing, then contiguous memory allocation can enhance processing speed.

Paging

1: Paging is a memory management approach that eliminates the need for continuous allocation of physical memory, focusing on non-contiguous allocation.

2: Paging involves retrieving process pages from secondary storage to main memory.

3: It divides processes into pages and uses frames to divide main memory, enabling non-contiguous physical address space.

4: Physical memory is split into fixed-size page frames, matching process page size.

5: Logical address space is divided into pages too.

6: When memory is requested, the OS assigns page frames, mapping logical pages to physical frames using a page table maintained by the memory management unit.

7: This table translates logical to physical addresses, aiding memory access.

Segmentation

1: Segmentation, a memory management technique in Operating Systems, involves dividing memory into variable-sized segments allocated to processes.

2: Segment details are stored in a segment table, potentially residing in one of the segments.

3: The table holds base and limit information.

4: Types of segmentation:

Virtual Memory Segmentation: Processes are divided into segments, possibly not all at once. Segmentation might occur during program runtime.

Simple Segmentation: Processes are divided into segments, all loaded into memory during runtime, not necessarily contiguously.

Advantages of Segmentation

1: No Internal fragmentation.

2: Segment Table consumes less space in comparison to Page table in paging.

3: As a complete module is loaded all at once, segmentation improves CPU utilization.

4: Segmentation provides a level of protection between segments, preventing one process from accessing or modifying another process's memory segment.

Disadvantages of Segmentation

1: segmentation can lead to external fragmentation as memory becomes divided into smaller segments. This can lead to wasted memory and decreased performance.

2: Using a segment table can increase overhead and reduce performance. Each segment table entry requires additional memory, and accessing the table to retrieve memory locations can increase the time needed for memory operations.

3: Segmentation can be more complex to implement and manage than paging. In particular, managing multiple segments per process can be challenging, and the potential for segmentation faults can increase as a result.

Segmented Paging

1: Pure segmentation is not very popular and not being used in many of the operating systems.

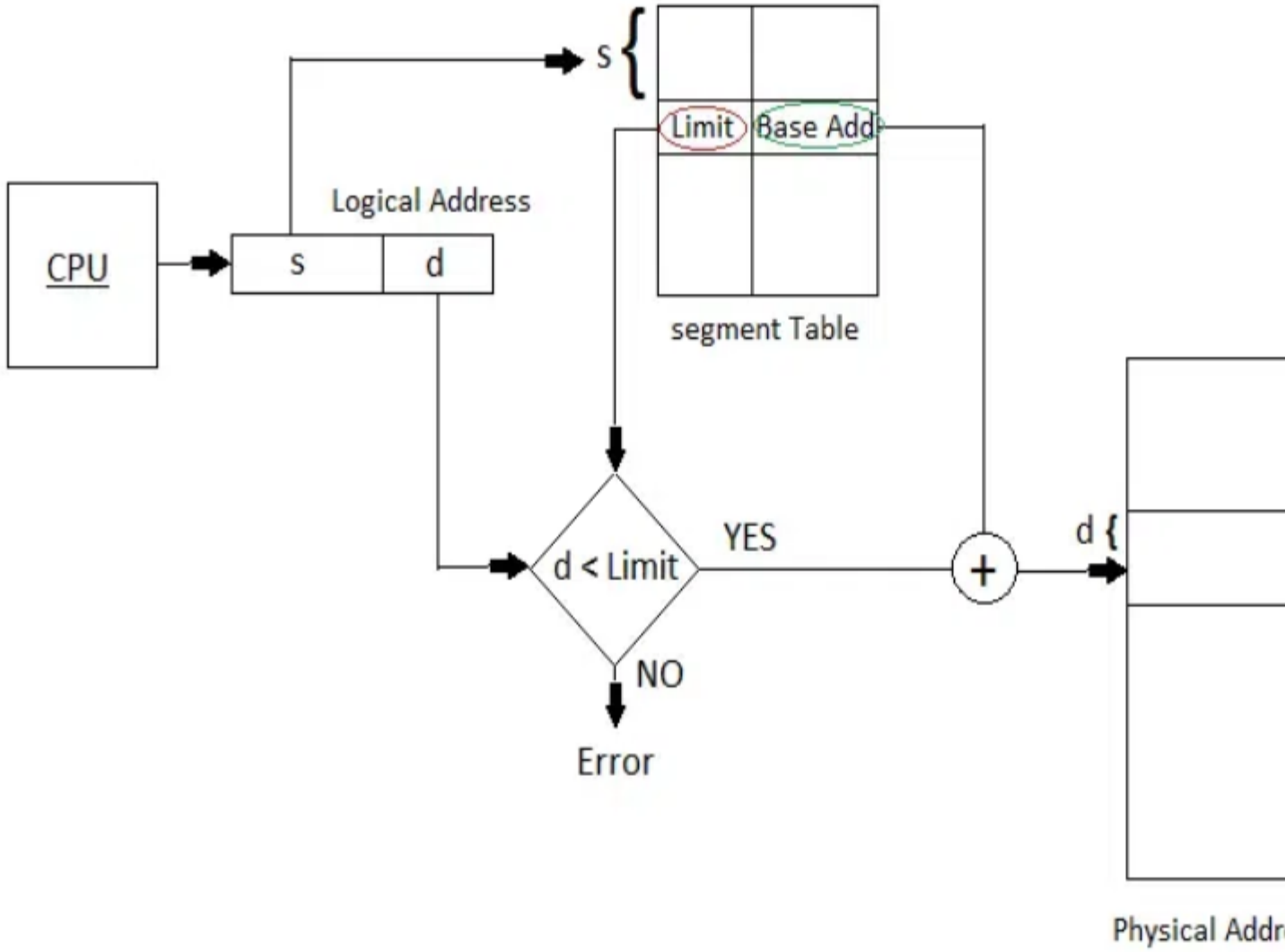
2: However, Segmentation can be combined with Paging to get the best features out of both the techniques.

3: In Segmented Paging, the main memory is divided into variable size segments which are further divided into fixed size pages.

4: Each Page table contains the various information about every page of the segment.

5: The Segment Table contains the information about every segment. Each segment table entry points to a page table entry and every page table entry is mapped to one of the page within a segment.

Translation of Two-dimensional Logical Address to Dimensional Physical Address.



Advantages of Segmented Paging

1. It reduces memory usage.

2. Page table size is limited by the segment size.

3. Segment table has only one entry corresponding to one actual segment.

4. External Fragmentation is not there.

5. It simplifies memory allocation.

Disadvantages of Segmented Paging

1. Internal Fragmentation will be there.

2. The complexity level will be much higher as compare to paging.

3. Page Tables need to be contiguously stored in the memory.

Virtual memory

1: Virtual Memory is a storage mechanism which offers user an illusion of having a very big main memory.

2: It is done by treating a part of secondary memory as the main memory.

3: In Virtual memory, the user can store processes with a bigger size than the available main memory.

4: Therefore, instead of loading one long process in the main memory, the OS loads the various parts of more than one process in the main memory.

5: Virtual memory is mostly implemented with demand paging and demand segmentation.

Demand paging

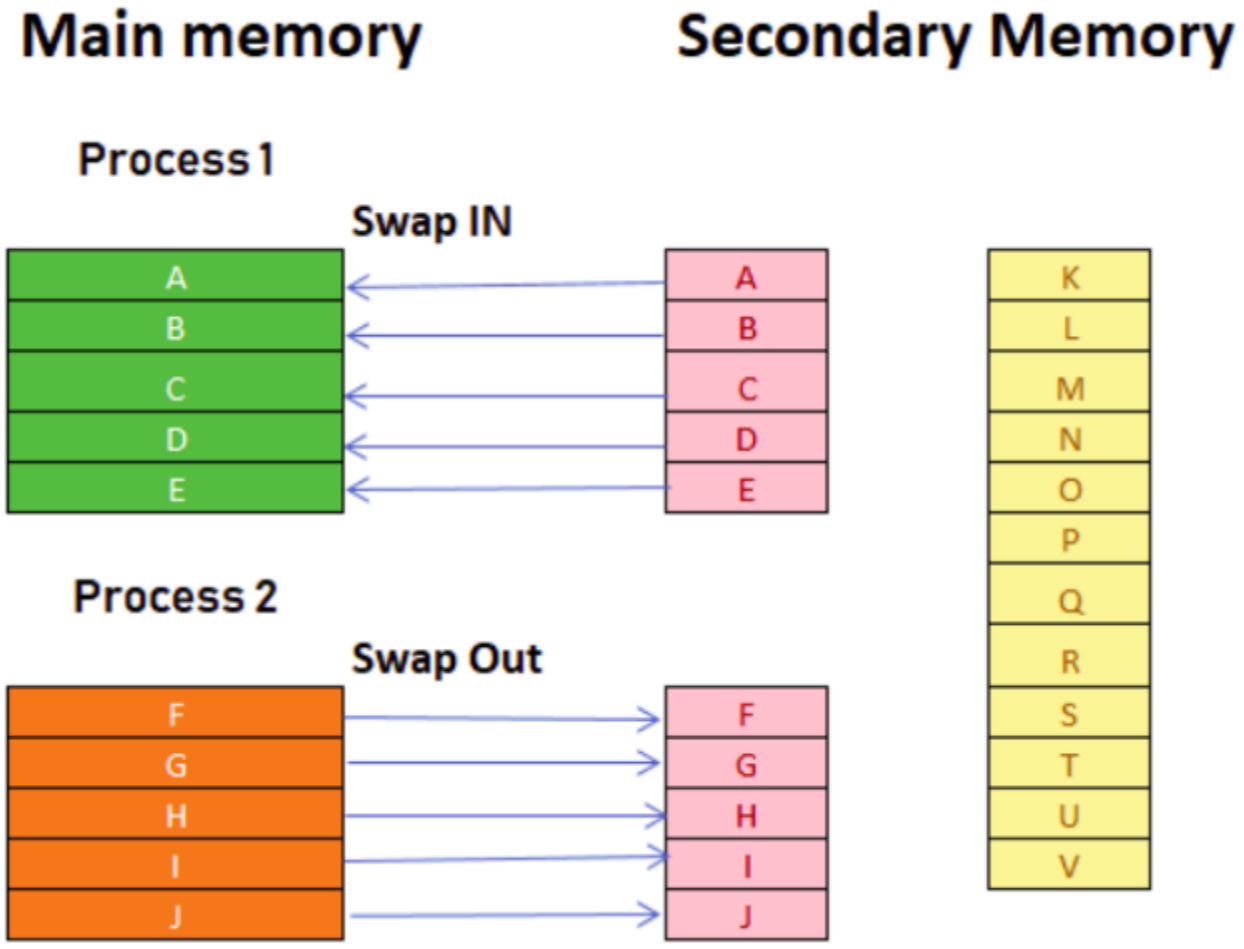
1: Demand paging is a memory management approach where processes are stored in secondary memory, and pages are loaded into main memory only when required, not in advance.

2: During context switches, the operating system doesn't copy all pages from disk to RAM.

3: Instead, the new program begins execution with the first page loaded, and subsequent pages are fetched on demand as they're referenced.

4: If a referenced page isn't in RAM due to swapping, a page fault occurs, causing the OS to retrieve the needed page from disk into memory.

5: This strategy optimizes memory utilization by loading only necessary pages, reducing unnecessary data transfer and enhancing efficiency.



FIFO Page Replacement

1: FIFO (First-in-first-out) is a simple implementation method.

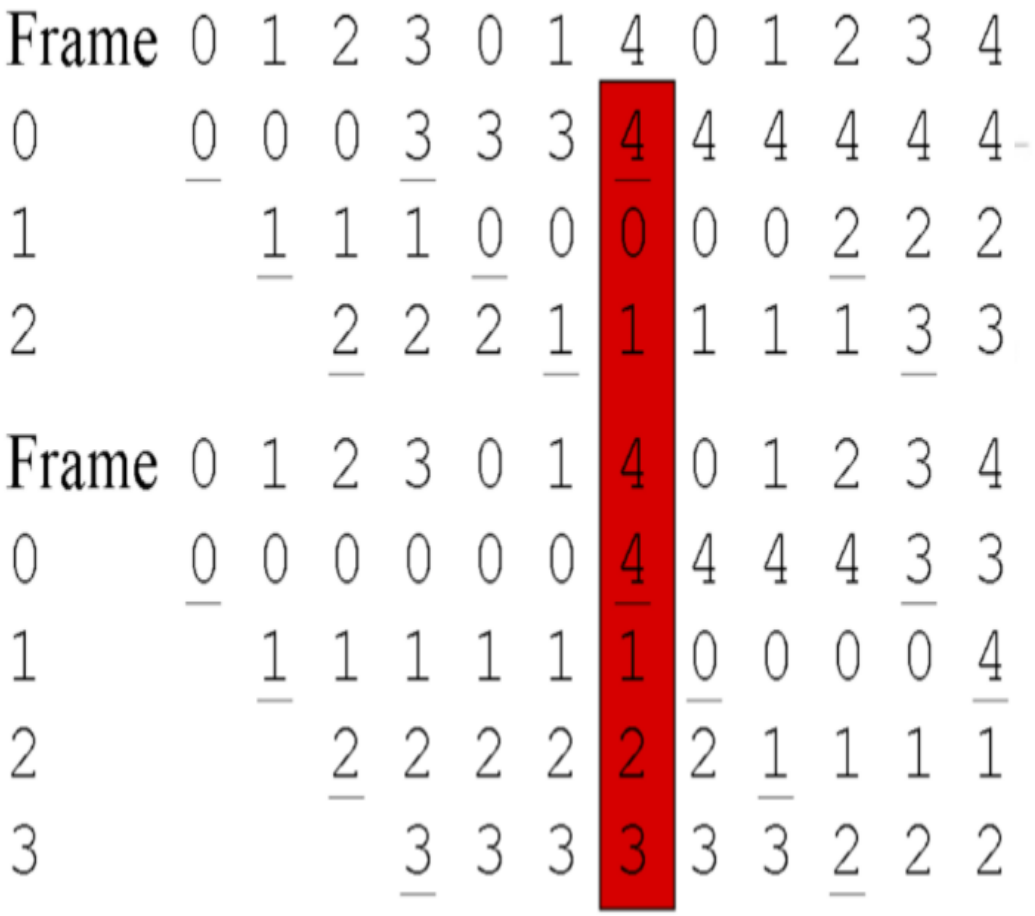
2: In this method, memory selects the page for a replacement that has been in the virtual address of the memory for the longest time.

3: On a page fault, the frame that has been in memory the longest is replaced.

4: Whenever a new page loaded, the page recently comes in the memory is removed.

5: So, it is easy to decide which page requires to be removed as its identification number is always at the FIFO stack.

6: The oldest page in the main memory is one that should be selected for replacement first.



Frame	0	1	2	3	0	1	4	0	1	2	3	4
0	0	0	0	3	3	3	4	4	4	4	4	4
1		1	1	1	0	0	0	0	0	2	2	2
2			2	2	2	1	1	1	1	1	3	3

Frame	0	1	2	3	0	1	4	0	1	2	3	4
0	0	0	0	0	0	0	4	4	4	4	3	3
1		1	1	1	1	1	1	0	0	0	0	4
2			2	2	2	2	2	2	1	1	1	1
3				3	3	3	3	3	3	2	2	2

Optimal Algorithm

1: The optimal page replacement method selects that page for a replacement for which the time to the next reference is the longest.

2: The Belady‘s optimal algorithm cheats. It looks forward in time to see which frame to replace on a page fault.

3: Thus it is not a real replacement algorithm.

4: It gives us a frame of reference for a given static frame access sequence.

5: Optimal algorithm results in the fewest number of page faults. This algorithm is

difficult to implement.

6: An optimal page-replacement algorithm method has the lowest page-fault rate of all algorithms. This algorithm exists and which should be called MIN or OPT.

LRU Page Replacement

1: LRU (Least Recently Used) is a page replacement algorithm in virtual memory.

2: It evicts the least recently accessed page when memory is full.

3: Pages are ranked by access time, discarding the oldest.

4: LRU helps optimize memory by retaining frequently used pages, but its implementation complexity can be high due to tracking access order.

5: While addressing Belady's Anomaly, where more frames lead to more page faults, LRU's computational cost might be significant.

6: Variants like Approximate LRU balance accuracy and efficiency, accommodating real-world scenarios with improved practicality.

Advantages of Virtual Memory

Here, are pros/benefits of using Virtual Memory:

1: Virtual memory helps to gain speed when only a particular segment of the program is required for the execution of the program.

2: It is very helpful in implementing a multiprogramming environment.

3: It allows you to run more applications at once.

4: It helps you to fit many large programs into smaller programs.

5: Common data or code may be shared between memory.

6: Process may become even larger than all of the physical memory.

7: Data / code should be read from disk whenever required.

Disadvantages of Virtual Memory

1: Applications may run slower if the system is using virtual memory.

2: Likely takes more time to switch between applications.

3: Offers lesser hard drive space for your use.

4: It reduces system stability.

5: It allows larger applications to run in systems that don't offer enough physical RAM

alone to run them.

Unit 8 (Protection and Security)

need for protection and security

1: Protection and security in computers are of paramount importance due to the increasing reliance on digital technologies and the interconnected nature of modern computing environments.
2: Computers store vast amounts of personal, financial, and sensitive information. Proper security measures ensure that this data is kept private and only accessible to authorized individuals or entities.
3: Protection and security measures prevent unauthorized individuals or malicious software from gaining access to computer systems, networks, and data. Unauthorized access can lead to data breaches, identity theft, and financial loss.
4: Cyberattacks, such as distributed denial of service (DDoS) attacks and phishing attempts, can disrupt services, steal data, and compromise systems.

COMPUTER SECURITY CONCEPTS

1: Confidentiality: This term covers two related concepts:

◦ Data confidentiality: Assures that private or confidential information is not made available or disclosed to unauthorized individuals.

◦ Privacy: Assures that individuals control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed.

2: Integrity: This term covers two related concepts:

◦ Data integrity: Assures that information and programs are changed only in a specified and authorized manner.

◦ System integrity: Assures that a system performs its intended function in an

unimpaired manner, free from deliberate or inadvertent unauthorized manipulation of the system.

3: Availability: Assures that systems work promptly and service is not denied to authorized users.

Threats and Attacks

Unauthorized disclosure is a threat to confidentiality. The following types of attacks can result in this threat consequence:

1: Exposure: This can be deliberate, as when an insider intentionally releases sensitive information, such as credit card numbers, to an outsider. It can also be the result of a human, hardware, or software error, which results in an entity gaining unauthorized knowledge of sensitive data. There have been numerous instances of this, such as universities accidentally posting student confidential information on the Web.

2: Interception: Interception is a common attack in the context of communications. On a shared local area network (LAN), such as a wireless LAN or a broadcast Ethernet, any device attached to the LAN can receive a copy of packets intended for another device. On the Internet, a determined hacker can gain access to e-mail traffic and other data transfers. All of these situations create the potential for unauthorized access to data.

3: Inference: An example of inference is known as traffic analysis, in which an adversary is able to gain information from observing the pattern of traffic on a network, such as the amount of traffic between particular pairs of hosts on the network. Another example is the inference of detailed information from a database by a user who has only limited access; this is accomplished by repeated queries whose combined results enable inference.

4: Intrusion: An example of intrusion is an adversary gaining unauthorized access to sensitive data by overcoming the system‘s access control protections.

5: Masquerade: One example of masquerade is an attempt by an unauthorized user to gain access to a system by posing as an authorized user; this could happen if the unauthorized user has learned another user‘s logon ID and password. Another example is malicious logic, such as a Trojan horse, that appears to perform a useful or desirable function but actually gains unauthorized access to system resources or tricks a user into executing other malicious logic.

Threat to hardware

1: A major threat to computer system hardware is the threat to availability.

Hardware is the most vulnerable to attack and the least susceptible to automated controls.

2: Threats include accidental and deliberate damage to equipment as well as theft.

3: The proliferation of personal computers and workstations and the widespread use of LANs increase the potential for losses in this area.

4: Theft of CD-ROMs and DVDs can lead to loss of confidentiality.

5: Physical and administrative security measures are needed to deal with these

threats.

Threat to software

1: Software includes the operating system, utilities, and application programs.

2: A key threat to software is an attack on availability. Software, especially application software, is often easy to delete.

3: Software can also be altered or damaged to render it useless.

4: Careful software configuration management, which includes making backups of the most recent version of software, can maintain high availability.

5: A more difficult problem to deal with is software modification that results in a program that still functions but that behaves differently than before, which is a threat to integrity/authenticity.

---

passive attack

1: A passive attack is a network attack in which a system is monitored and sometimes scanned for open ports and vulnerabilities.

2: The purpose of a passive attack is to gain information about the system being targeted; it does not involve any direct action on the target.

3: Passive attacks include active reconnaissance and passive reconnaissance.

4: Active reconnaissance: The intruder engages with the target system to gather information about vulnerabilities. Attackers often use methods such as port scanning to learn which ports are open and what services are running on them.

5: Passive reconnaissance: The intruder monitors the system for vulnerabilities without interaction for the sole purpose of gaining information. Often the attacker monitors a user's web session and then uses information retrieved from that session to conduct a future attack.

Types of passive attacks

Passive attacks can take various forms, including the following:

Traffic analysis: This involves analyzing network traffic as it moves to and from the target systems. These types of attacks use statistical methods to analyze and interpret the patterns of communication exchanged over the network. These attacks can be performed on encrypted network traffic, but they are more common on unencrypted traffic.

Eavesdropping: Eavesdropping occurs when an attacker intercepts sensitive information by listening to phone calls or reading unencrypted messages exchanged in a communication medium. Although eavesdropping is similar to snooping, snooping is limited to gaining access to data during transmission.

Footprinting: This is the process of gathering as much information as possible about the target company's network, hardware, software and employees. Footprinting gathers information on the target, such as IP address, domain name system information and employee ID. Footprinting is also the first step in gathering information for a penetration test.

Active attack

1: An active attack in cybersecurity refers to a type of cyberattack where an unauthorized entity actively interacts with a computer system or network with the intention of altering, disrupting, or manipulating its normal operations.

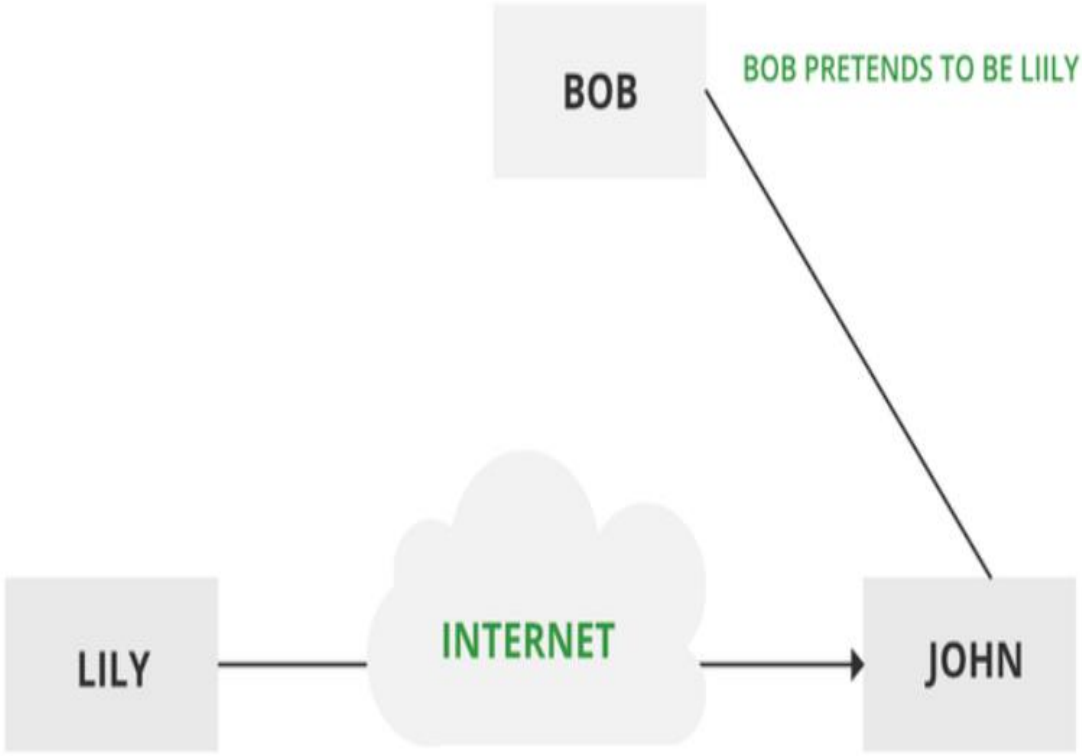
2: Unlike passive attacks that focus on intercepting and eavesdropping on data, active attacks involve taking direct actions to compromise the target system's integrity, confidentiality, or availability.

3: These attacks are more overt and can have a more immediate and noticeable impact on the targeted system.

Types of active attacks

1: Masquerade -

Masquerade is a type of cybersecurity attack in which an attacker pretends to be someone else in order to gain access to systems or data. This can involve impersonating a legitimate user or system to trick other users or systems into providing sensitive information or granting access to restricted areas.



Masquerade Attack

Modification of messages –It means that some portion of a message is altered or that message is delayed or reordered to produce an unauthorized effect. Modification is an attack on the integrity of the original data. It basically means that unauthorized parties not only gain access to data but also spoof the data by triggering denial-of-service attacks, such as altering transmitted data packets or flooding the network with fake data. Manufacturing is an attack on authentication.

Denial of Service – Denial of Service (DoS) is a type of cybersecurity attack that is designed to make a system or network unavailable to its intended users by overwhelming it with traffic or requests. In a DoS attack, an attacker floods a target system or network with traffic or requests in order to consume its resources, such as bandwidth, CPU cycles, or memory, and prevent legitimate users from accessing it.

Intruders

1: One of the two most publicized threats to security is the intruder (the other is viruses), often referred to as a hacker.

2: They have immense knowledge and an in-depth understanding of technology and security. Intruders breach the privacy of users and aim at stealing the confidential information of the users.

3: The stolen information is then sold to third-party, which aim at misusing the information for their own personal or professional gains.

4: three classes of intruders:

Masquerader: An individual who is not authorized to use the computer and who penetrates a system‘s access controls to exploit a legitimate user‘s account

Misfeasor: A legitimate user who accesses data, programs, or resources for which such access is not authorized, or who is authorized for such access but misuses his or her privileges

Clandestine user: An individual who seizes supervisory control of the system and uses this control to evade auditing and access controls or to suppress audit collection.

Examples of intrusion

1: Performing a remote root compromise of an e-mail server.

2: Defacing a Web server.

3: Guessing and cracking passwords.

4: Copying a database containing credit card numbers.

5: Viewing sensitive data, including payroll records and medical information, without authorization.

6: Running a packet sniffer on a workstation to capture usernames and passwords Using a permission error on an anonymous FTP server to distribute pirated software and music files.

7: Using an unattended, logged-in workstation without permission.

Intruder Behavior Patterns

Insider attack

1: An insider attack in cybersecurity refers to a type of threat where a person or entity with authorized access to an organization's systems, networks, or sensitive information intentionally or unintentionally exploits that access to compromise the confidentiality, integrity, or availability of data, systems, or resources.

2: Insider attacks can be particularly damaging because the attackers have legitimate access and often possess a deep understanding of the organization's operations and security mechanisms.

3: Insider attacks are generally classified into two main categories:

Malicious Insiders: These are individuals within the organization who deliberately engage in harmful activities. Malicious insiders might be motivated by financial gain, revenge, ideology, or personal reasons. They use their knowledge of the organization's systems and security practices to bypass defenses and carry out attacks.

Accidental Insiders: These individuals are not intentionally malicious but inadvertently contribute to security breaches due to mistakes, negligence, or lack of awareness about security practices. Accidental insiders may unknowingly introduce vulnerabilities or expose sensitive information, leading to breaches.

Solution to insider attack

Although IDS and IPS facilities can be useful in countering insider attacks, other more direct approaches are of higher priority. Examples include the following:

1: Enforce least privilege, only allowing access to the resources employees need to do their job.

2: Set logs to see what users access and what commands they are entering.

3: Protect sensitive resources with strong authentication.

4: Upon termination, delete employee‘s computer and network access.

5: Upon termination, make a mirror image of employee‘s hard drive before reissuing it.

That evidence might be needed if your company information turns up at a competitor.

Virus

1: A computer virus is a piece of software that can ―infect other programs by modifying them; the modification includes injecting the original program with a routine to make copies of the virus program, which can then go on to infect other programs.

2: computer virus carries in its instructional code the recipe for making perfect copies of itself.

3: The typical virus becomes embedded in a program on a computer.

4: Then, whenever the infected computer comes into contact with an

uninfected piece of software, a fresh copy of the virus passes into the new program.

5: Thus, the infection can be spread from computer to computer by unsuspecting users who either swap disks or send programs to one another over a network.

6: In a network environment, the ability to access applications and system services on other computers provides a perfect culture for the spread of a virus.

Four phases of a typical virus

1: Dormant phase: The virus is idle. The virus will eventually be activated by some event, such as a date, the presence of another program or file, or the capacity of the disk exceeding some limit. Not all viruses have this stage.

2: Propagation phase: The virus places an identical copy of itself into other programs or into certain system areas on the disk. Each infected program will now contain a clone of the virus, which will itself enter a propagation phase.

3: Triggering phase: The virus is activated to perform the function for which it was intended. As with the dormant phase, the triggering phase can be caused by a variety

of system events, including a count of the number of times that this copy of the virus has made copies of itself.

4: Execution phase: The function is performed. The function may be harmless, such as a message on the screen, or damaging, such as the destruction of programs and data files.

Viruses Classification

1: Encrypted virus: A typical approach is as follows. A portion of the virus creates a random encryption key and encrypts the remainder of the virus. The key is stored with the virus. When an infected program is invoked, the virus uses the stored random key to decrypt the virus. When the virus replicates, a different random key is selected.

Because the bulk of the virus is encrypted with a different key for each instance, there is no constant bit pattern to observe.

2: Stealth virus: A form of virus explicitly designed to hide itself from detection by antivirus software. Thus, the entire virus, not just a payload, is hidden.

3: Polymorphic virus: A virus that mutates with every infection, making detection by the ―"signature" of the virus impossible.

4: Metamorphic virus: As with a polymorphic virus, a metamorphic virus mutates with every infection. The difference is that a metamorphic virus rewrites itself completely at each iteration, increasing the difficulty of detection. Metamorphic viruses may change their behavior as well as their appearance.

Macro virus

1: Macro viruses are platform-independent and commonly infect Microsoft Office documents.

2: They exploit macros in applications like Word and Excel for automation.

3: Infecting documents, not code, they spread via methods like email and transcend hardware and operating systems supporting these apps.

4: As macros automate tasks, attackers misuse them.

5: Traditional file access controls have limited utility against them due to targeting documents rather than system programs.

6: Macros are essentially embedded executable programs that save keystrokes by automating tasks.

7: They often spread through email, utilizing the Basic programming language and are triggered by specific keystroke sequences or function keys.

E-mail virus

1: A more recent development in malicious software is the e-mail virus. The first rapidly spreading e-mail viruses, such as Melissa, made use of a Microsoft Word macro embedded in an attachment.

2: If the recipient opens the email attachment, the Word macro is activated. Then the e-mail virus sends itself to everyone on the mailing list in the user‘s e-mail package and the virus does local damage on the user‘s system.

3: The virus propagates itself as soon as it is activated (either by opening an e-mail attachment or by opening the e-mail) to all of the e-mail addresses known to the infected host.

4: As a result, whereas viruses used to take months or years to propagate, they now do so in hours.

5: This makes it very difficult for antivirus software to respond before much damage is done.

6: Ultimately, a greater degree of security must be built into Internet utility and application software on PCs to counter the growing threat.

---

worms

1: Worms are self-replicating malicious software in cybersecurity that can spread independently across networks and systems without requiring user interaction.

2: They exploit vulnerabilities in software or network protocols to propagate and can cause significant damage to both individual computers and entire networks.

3: Unlike viruses, worms don't need to attach themselves to existing files or programs to spread.

4: Due to their ability to spread quickly across networks, worms can cause widespread disruptions, compromise sensitive information, and lead to financial losses.

5: Worms remain a significant threat in cybersecurity due to their ability to rapidly spread across networks. Employing a combination of proactive security measures, regular updates, and user education is crucial to mitigating their impact.

Worm Propagation Model

1: Worm Propagation Model describes a model for worm propagation based on an analysis of recent worm attacks.

2: The speed of propagation and the total number of hosts infected depend

on a number of factors, including the mode of propagation, the vulnerability or vulnerabilities exploited, and the degree of similarity to preceding attacks.

3: For the latter factor, an attack that is a variation of a recent previous attack may be countered more effectively than a more novel attack.

4: Following Figure shows the dynamics for one typical set of parameters. **5:** Propagation proceeds through three phases.

6: In the initial phase, the number of hosts increases exponentially.

7: To see that this is so, consider a simplified case in which a worm is launched from a single host and infects two nearby hosts.

8: Each of these hosts infects two more hosts, and so on.

9: This results in exponential growth.

10: After a time, infecting hosts waste some time attacking already infected

hosts, which reduces the rate of infection.

11: During this middle phase, growth is approximately linear, but the rate of infection is rapid.

12: When most vulnerable computers have been infected, the attack enters a slow finish phase as the worm seeks out those remaining hosts that are difficult to identify.

State of Worm Technology

The state of the art in worm technology includes the following:

1: Multiplatform: Newer worms are not limited to Windows machines but can attack a variety of platforms, especially the popular varieties of UNIX.

2: Multiexploit: New worms penetrate systems in a variety of ways, using exploits against Web servers, browsers, e-mail, file sharing, and other network-based applications.

3: Ultrafast spreading: One technique to accelerate the spread of a worm is to conduct a prior Internet scan to accumulate Internet addresses of vulnerable machines.

4: Polymorphic: To evade detection, skip past filters, and foil real-time analysis, worms adopt the virus polymorphic technique. Each copy of the worm has new code. Generated on the fly using functionally equivalent instructions and encryption techniques.

5: Metamorphic: In addition to changing their appearance, metamorphic worms have a repertoire of behavior patterns that are unleashed at different stages of propagation.

6: Transport vehicles: Because worms can rapidly compromise a large number of systems, they are ideal for spreading other distributed attack tools, such as distributed denial of service bots.

7: Zero-day exploit: To achieve maximum surprise and distribution, a worm should exploit an unknown vulnerability that is only discovered by the general network community when the worm is launched.

Bots

1: Bots, short for "robots," are software applications or scripts that perform automated tasks on computers and computer networks.

2: These tasks can range from simple and repetitive actions to more complex and sophisticated operations.

3: Bots can serve both legitimate and malicious purposes, making them a diverse and sometimes controversial element in the digital landscape.

Types of Bots

1: Crawlers or Web Scrapers: Used by search engines to index websites and gather information.

2: Chatbots: Used for automated customer service interactions and online assistance.

3: Social Media Bots: Used to automate interactions on social media platforms, both for legitimate marketing and malicious purposes (e.g., spreading misinformation or manipulating trends).

4: IoT Bots: Infected Internet of Things (IoT) devices that can be harnessed for botnets.

5: Spambots: Send spam emails or messages to a large number of recipients.

6: Trading Bots: Used in financial markets to automate trading decisions.

7: Game Bots: Used in online games to automate gameplay, sometimes violating game rules.

Uses of Bots

1: Distributed denial-of-service attacks: A DDoS attack is an attack on a computer

system or network that causes a loss of service to users.

2: Spamming: With the help of a botnet and thousands of bots, an attacker is able to

send massive amounts of bulk e-mail (spam).

3: Sniffing traffic: Bots can also use a packet sniffer to watch for interesting clear-text

data passing by a compromised machine. The sniffers are mostly used to retrieve

sensitive information.

4: Keylogging works by monitoring and recording the keystrokes entered by a user on a computer's keyboard. The recorded keystrokes can include text, passwords, usernames, credit card numbers, and other sensitive information.

Remote Control Facility

1: A bot stands apart from a worm due to its remote control feature.

2: While a worm self-propagates and activates, a bot remains governed by a central facility, at least initially.

3: Typically, this control is achieved through an IRC server where bots congregate in a designated channel, interpreting incoming messages as directives.

4: Contemporary botnets favor concealed communication via protocols like HTTP over traditional IRC methods.

5: To enhance resilience, distributed control mechanisms are employed to avoid a single point of failure.

6: Once communication is established between the control module and bots, activation occurs.

7: This involves issuing commands, ranging from executing predefined routines to downloading files for diverse applications.

Constructing the Attack Network

The first step in a botnet attack is for the attacker to infect a number of machines with bot software that will ultimately be used to carry out the attack. The essential ingredients in this phase of the attack are the following:

1. Software that can carry out the attack: The software must be able to run on a large number of machines, must be able to conceal its existence, must be able to communicate with the attacker or have some sort of time-triggered mechanism, and

must be able to launch the intended attack toward the target.

2. A vulnerability in a large number of systems: The attacker must become aware of a vulnerability that many system administrators and individual users have failed to patch and that enables the attacker to install the bot software.

Scanning

1: In the scanning process, the attacker first seeks out a number of vulnerable machines and infects them.

2: Then, typically, the bot software that is installed in the infected machines repeats the same scanning process, until a large distributed network of infected machines is created.

3: the following types of scanning strategies:

Random: Each compromised host probes random addresses in the IP address space, using a different seed. This technique produces a high volume of Internet traffic, which may cause generalized disruption even before the actual attack is launched.

Topological: This method uses information contained on an infected victim machine to find more hosts to scan.

Local subnet: The host uses the subnet address structure to find other hosts that would otherwise be protected by the firewall.

Authentication

1: Authentication in computers refers to the process of verifying the identity of a user, system, or application attempting to access a resource or system.

2: It ensures that the entity requesting access is who they claim to be.

3: This can involve various methods such as passwords, biometrics, two-factor authentication, and digital certificates.

4: The goal is to prevent unauthorized access and enhance security.

Means of Authentication

There are four general means of authenticating a user‘s identity, which can be used alone or in combination:

1: Something the individual knows: Examples includes a password, a personal identification number (PIN), or answers to a prearranged set of questions.

2: Something the individual possesses: Examples include electronic keycards, smart cards, and physical keys. This type of authenticator is referred to as a token.

3: Something the individual is (static biometrics): Examples include recognition by fingerprint, retina, and face.

4: Something the individual does (dynamic biometrics): Examples include

recognition by voice pattern, handwriting characteristics, and typing rhythm.

Password-Based Authentication

1: The password system is a crucial defense against intruders in various environments like multiuser systems, servers, and web-based services.

2: Users provide both a name (ID) and a password for access.

3: The password is compared to a stored one, authenticating the user's ID.

4: The ID's role encompasses authorization, determining who can access the system, and privileges, dictating what actions a user can perform.

5: Additionally, the ID facilitates discretionary access control, allowing users to grant file access permissions to others.

6: Some users may have elevated status, while others might have limited privileges.

7: Overall, this combination of ID and password enhances security and access control.

Salt value

1: A salt value is a random and unique string of characters that is added to a password before hashing it for storage.

2: It enhances security by preventing attackers from using precomputed tables (rainbow tables) to quickly guess passwords.

3: When a salt is used, each password hash is unique, even if the original passwords are the same.

4: This makes it significantly more challenging for attackers to crack passwords using methods like dictionary attacks or brute force.

5: Salting is a common practice in password security to mitigate the risks associated with password hashing and improve overall system security.

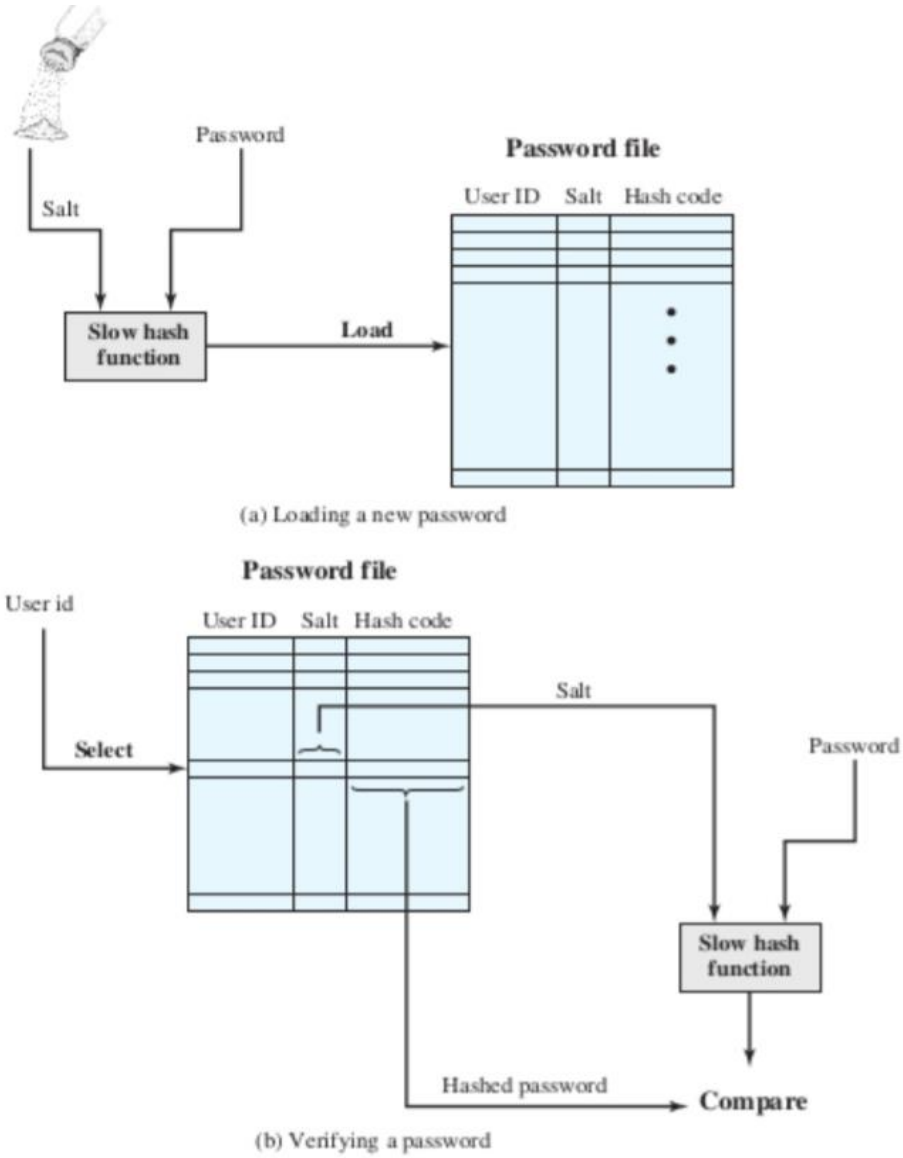


Figure 6. UNIX Password Scheme

UNIX Implementation

1: Since the inception of UNIX, a common password system has been utilized.

2: Users create a password up to eight characters long, which is then transformed into a 56-bit value through an encryption process.

3: This method employs the crypt(3) hash routine, utilizing a 12-bit salt value, and the DES algorithm.

4: The algorithm operates on a 64-bit zero data block, undergoing two encryption stages repeated 25 times.

5: The outcome is a 64-bit result, converted into an 11-character sequence.

6: This adjusted DES algorithm acts as a one-way hash function, enhancing security.

7: The crypt(3) routine is deliberately crafted to deter guessing attacks.

Token based authentication

1: Tokens used for user authentication, such as bank card-sized objects, are discussed in this section.

2: Two prevalent token types are examined: memory cards and bank cards with magnetic stripes.

3: Memory Cards: These cards store data but don't process it. The most familiar example is the bank card with a magnetic stripe that contains a simple security code.

4: Unfortunately, these stripes can be read and reprogrammed by inexpensive card readers.

5: Some memory cards also have internal electronic memory.

6: For user authentication, memory cards are often used alongside a password or PIN.

7: They find use in scenarios like physical access control (e.g., hotel rooms) and computer authentication (e.g., ATMs).

Disadvantages of Token based authentication

Requires special reader: This increases the cost of using the token and creates the requirement to maintain the security of the reader‘s hardware and software.

Token loss: A lost token temporarily prevents its owner from gaining system access. Thus there is an administrative cost in replacing the lost token. In addition, if the

token is found, stolen, or forged, then an adversary now need only determine the PIN to gain unauthorized access.

User dissatisfaction: Although users may have no difficulty in accepting the use of a memory card for ATM access, its use for computer access may be deemed inconvenient.

Smart tokens

The term "smart tokens" encompasses a range of devices, which can be categorized based on three characteristics:

1. Physical Characteristics: Smart tokens can have an embedded microprocessor. Those resembling bank cards are termed smart cards, while others might resemble calculators, keys, or small objects.

2.Interface: Smart tokens can have manual interfaces, like keypads and displays, for human interaction. Some possess electronic interfaces that communicate with compatible reader/writer devices.

3.Authentication Protocol: Smart tokens serve user authentication purposes, employing three main authentication protocols:

Static.

Dynamic Password Generator.

Challenge-Response.

Biometric Authentication

1: A biometric authentication system attempts to authenticate an individual based on his or her unique physical characteristics.

2: These include static characteristics, such as fingerprints, hand geometry, facial characteristics, and retinal and iris patterns; and dynamic characteristics, such as voiceprint and signature.

3: In essence, biometrics is based on pattern recognition.

4: Compared to passwords and tokens, biometric authentication is both technically complex and expensive.

5: While it is used in a number of specific applications, biometrics has yet to mature as a standard tool for user authentication to computer systems.

Types of biometric authentication

1: Facial characteristics: The most common approach is to define characteristics based on relative location and shape of key facial features, such as eyes, eyebrows, nose, lips, and chin shape. An alternative approach is to use an infrared camera to produce a face thermogram that correlates with the underlying vascular system in the human face.

2: Fingerprints : A fingerprint is the pattern of ridges and furrows on the surface of the fingertip. Fingerprints are believed to be unique across the entire human population. In practice, automated fingerprint recognition and matching system extract a number of features from the fingerprint for storage as a numerical surrogate for the full fingerprint pattern.

3: Hand geometry: Hand geometry systems identify features of the hand, including

shape, and lengths and widths of fingers.

4: Retinal pattern: The pattern formed by veins beneath the retinal surface is unique.and therefore suitable for identification. A retinal biometric system obtains a digital image of the retinal pattern by projecting a low-intensity beam of visual or infrared light into the eye.

5: Iris: Another unique physical characteristic is the detailed structure of the iris.