OPERATING SYSTEMS
Unit – I

1.1 Introduction

An operating system (OS) is a fundamental software component that serves as an intermediary between computer hardware and user applications. It plays a crucial role in managing and coordinating various system resources to ensure efficient and secure execution of tasks. Here's a basic introduction to operating systems:

1. Definition and Purpose:

An operating system is a software layer that acts as an interface between the computer hardware and user applications. Its primary purpose is to provide a convenient and efficient environment for users to interact with the computer while managing hardware resources.

2. Key Functions:

- **Process Management:**

OS manages processes, which are instances of executing programs. It allocates resources, schedules tasks, and ensures smooth execution.

- **Memory Management:**

OS controls and organizes the computer's memory, allocating space to programs and data, and handling memory access to prevent conflicts.

- **File System Management:**

It provides a hierarchical structure for organizing and storing files, ensuring data persistence and accessibility.

- **Device Management:**

OS interacts with hardware devices, managing communication and data transfer between the computer and peripherals.

- **Security and Protection:**

Operating systems implement security measures to protect data and resources, controlling access to system functions and sensitive information.

3. User Interface:

- **Command-Line Interface (CLI):**Users interact with the system through text commands.
- **Graphical User Interface (GUI):**

Utilizes graphical elements like icons and windows, providing a more user-friendly experience.

4. Types of Operating Systems:

Single-User, Single-Tasking: Supports one user and one task at a time.
Single-User, Multi-Tasking: Allows a single user to run multiple tasks simultaneously.
Multi-User: Supports multiple users accessing the system concurrently.
Real-Time Operating System (RTOS): Designed for systems with strict timing requirements, such as embedded systems and control systems.

5. Examples of Operating Systems:

Microsoft Windows: Widely used in personal computers.
MacOS: Found on Apple Macintosh computers.
Linux:An open-source OS kernel used in various distributions (distros).

Unix:A powerful, multi-user, and multitasking OS influencing Linux and macOS.
Android: An OS designed for mobile devices.

6. Evolution of Operating Systems:

Early computers had no operating systems, requiring users to interact directly with hardware.
Batch processing systems allowed for the execution of predefined sequences of jobs without user intervention.
Time-sharing systems enabled multiple users to interact with the computer simultaneously.
Modern operating systems incorporate features like multitasking, virtual memory, and graphical user interfaces.

Understanding operating systems is crucial for anyone working with computers, as it forms the foundation for software development, system administration, and computer science in general.

1.2 COMPUTER SYSTEM ARCHITECTURE:

Computer system architecture refers to the design and structure of the various components that make up a computer and how they interact with each other. It encompasses both hardware and system software components, detailing their organization and functionality. Here's an overview of computer system architecture:

1. Basic Components:

Central Processing Unit (CPU): Often referred to as the brain of the computer, the CPU executes instructions stored in memory, performs calculations, and manages data flow.
Memory: Stores data and instructions that the CPU can quickly access. There are two main types: RAM (Random Access Memory) for temporary storage and ROM (Read-Only Memory) for permanent storage of essential system instructions.
Input Devices: Enable users to provide data or commands to the computer, such as keyboards, mice, and touchscreens.
Output Devicess: Display or present information processed by the computer, including monitors, printers, and speakers.
Storage Devices: Store data permanently, such as hard drives, solid-state drives, and optical drives.

2. Bus System:

- A bus is a communication system that allows data transfer between components. It consists of address buses for specifying memory locations and data buses for transferring information.
- The bus architecture determines how data flows between the CPU, memory, and other peripherals.

3. Registers:

- Registers are small, high-speed storage locations within the CPU used to store temporary data and control information.
- They play a crucial role in the execution of instructions and data manipulation.

4. Instruction Set Architecture (ISA):

- ISA defines the set of instructions that a CPU understands and can execute.
- It includes operations like arithmetic, logic, data movement, and control transfer instructions.

5. Pipelines and Caches:

- Modern CPUs often use pipelines to parallelize instruction execution, allowing for higher throughput.
- Caches are small, high-speed memory units that store frequently accessed data to reduce memory access times.

6. System Software:

- The operating system and other system software manage and control the computer's resources.
- Device drivers facilitate communication between the operating system and hardware components.

7. Input-Output (I/O) System:

- Manages the flow of data between the CPU, memory, and peripheral devices.
- I/O controllers handle specific types of devices, ensuring proper data transfer and communication.

8. Clock and Synchronization:

- The system clock synchronizes the operations of various components, defining the pace at which instructions are executed.
- Synchronization mechanisms prevent conflicts and ensure data consistency.

9. Parallelism and Multiprocessing:

- Some systems use multiple processors or cores to perform tasks concurrently, enhancing overall system performance.
- Parallel processing involves breaking down tasks into smaller subtasks that can be executed simultaneously.

Understanding computer system architecture is essential for computer scientists, engineers, and programmers to design efficient systems, optimize performance, and troubleshoot issues at both hardware and software levels. It provides a foundational knowledge base for working with computers and developing advanced computing systems.

1.3 OPERATIONS OF OS SERVICES OF OS

Refer Notes

1.4 EVOLUTION OF OS

The evolution of operating systems can be traced back to the 1950s when computers were first developed. Here is a brief overview of the key stages in the evolution of operating systems:

1. 1950s-1960s: The Batch Processing Era

- *Early computers used batch processing systems, where users would submit jobs on punch cards to be processed in batches by the computer.*
- *Operating systems were basic and mainly managed the execution of jobs and the allocation of resources.*

2. 1960s-1970s: The Time-Sharing Era

- Time-sharing operating systems were developed to allow multiple users to interact with a computer system simultaneously.
- IBM's OS/360 and the Unix operating system (developed at Bell Labs) were significant developments during this period.

3. 1970s-1980s: The Personal Computing Era

- The introduction of microprocessors and personal computers led to the development of operating systems for individual users.
- Microsoft's MS-DOS and Apple's Macintosh operating system were among the early personal computer operating systems.

4. 1980s-1990s: The Graphical User Interface (GUI) Era

- Graphical user interfaces became popular, making computers more user-friendly.
- Microsoft Windows and Apple's MacOS emerged as dominant GUI-based operating systems.

5. 1990s-Present: The Networked Computing Era

- The rise of the internet and networking led to the development of operating systems with built-in networking capabilities.
- Linux, developed as a free and open-source operating system, gained popularity, particularly in server environments.
- Microsoft Windows and MacOS continued to dominate the personal computer market, with Windows becoming the dominant OS for desktop PCs.

6. 2000s-Present: The Mobile Computing Era

- The proliferation of smartphones and tablets led to the development of mobile operating systems such as iOS and Android.
- These operating systems are designed for touchscreens and provide a different user experience compared to traditional desktop operating systems.

7. 2010s-Present: The Cloud Computing Era

- Cloud computing has become increasingly popular, leading to the development of operating systems optimized for cloud environments.
- Operating systems like Chrome OS and various Linux distributions have gained traction in this era.

Throughout this evolution, operating systems have continued to evolve to meet the changing needs of users and the advancements in computer technology.

1.5 TYPES OF OS

There are several types of operating systems, each designed to serve different purposes and cater to specific computing environments. Here are some of the main types:

1. Single-User, Single-Tasking Operating Systems: *These operating systems are designed to allow only one user to run one program at a time. Examples include early versions of MS-DOS.*

2. Single-User, Multi-Tasking Operating Systems: *These operating systems allow a single user to run multiple programs simultaneously. Examples include modern versions of Windows, MacOS, and Linux.*

3. Multi-User Operating Systems: ***These operating systems allow multiple users to access a computer system concurrently. They are commonly used in server environments. Examples include Unix and Linux distributions.***

4. Real-Time Operating Systems (RTOS): *RTOS are designed to respond to input or events within a guaranteed time frame. They are used in applications where timing is critical, such as industrial automation, robotics, and aerospace systems.*

5. Mobile Operating Systems: *These operating systems are designed for mobile devices such as smartphones and tablets. Examples include Android, iOS, and Windows Phone.*

6. Embedded Operating Systems: *Embedded operating systems are used in embedded systems, which are specialized computing devices designed for specific tasks. Examples include operating systems used in routers, digital cameras, and microwave ovens.*

7. Network Operating Systems: *These operating systems are designed to support networking functionalities, such as file sharing, printer sharing, and network security. Examples include Windows Server, Linux distributions optimized for servers, and Novell NetWare.*

8. Distributed Operating Systems: *Distributed operating systems manage a group of independent computers and make them appear to be a single computer system. They are used in large-scale computing environments such as data centers and cloud computing.*

These are just some of the main types of operating systems, and there are many variations and specialized operating systems designed for specific purposes and environments.
Also Refer Notes

UINT – II

CPU AND PROCESS SCHEDULING

2.1 Process Concepts:

Process concepts are fundamental to understanding how operating systems manage programs running on a computer. Here are some key concepts related to processes:

1. **Process :** A process is an instance of a program in execution. It represents a running program along with its current state, including program counter, registers, and variables. Each process has its own memory space and resources allocated by the operating system.

2. **Process State:** The state of a process indicates its current condition or stage in its execution. Common process states include:

- **New:** The process is being created.
- **Ready:** The process is ready to run and waiting to be assigned to a processor.
- **Running:** The process is currently being executed by a processor.
- **Blocked (or Waiting):** The process is waiting for an event or resource, such as I/O completion.
- **Terminated:** The process has finished execution.

3**. Process Control Block (PCB):** PCB is a data structure maintained by the operating system for each process. It contains information such as process state, program counter, CPU registers, memory allocation, and accounting information.

4. **Process Scheduling:** Process scheduling is the mechanism by which the operating system decides which process to run next on the CPU. It involves selecting a process from the ready queue and allocating the CPU to that process.

5**. Context Switching:** Context switching is the process of saving the state of a process that is currently running, loading the state of a new process, and switching the CPU from the old process to the new process. It is an essential part of multitasking operating systems.

6**. Process Creation:** Process creation is the mechanism by which new processes are created. This typically involves allocating memory for the process, initializing the process control block, and loading the program into memory.

7. **Process Termination:** Process termination is the process by which a process is ended either by completing its execution or by being forcefully terminated by the operating system.

Understanding these process concepts is essential for developing and managing programs in a multitasking environment, where multiple processes run concurrently on a single processor.

2.2 Process States:

Process states represent the different stages a process goes through during its execution. The exact states and their names may vary slightly depending on the operating system, but the basic concepts remain similar. Here is a detailed explanation of the common process states:

1. New: In this state, a new process is being created. The operating system creates a process control block (PCB) for the process, allocates memory, and initializes the process's data structures. However, the process has not yet started execution.

2. Ready: A process in the ready state is prepared to execute but is waiting for the CPU to be assigned to it. The process is placed in a ready queue, where it waits until the operating system's scheduler selects it to run on the CPU.

3. Running: When the scheduler selects a process from the ready queue and assigns the CPU to it, the process enters the running state. In this state, the process's instructions are being executed on the CPU. A process typically remains in the running state until it is interrupted or voluntarily gives up the CPU.

4. Blocked (or Waiting): A process in the blocked state is unable to proceed until a specific event occurs, such as the completion of an I/O operation or the availability of a resource. When a process is blocked, it is temporarily removed from the CPU and placed in a waiting queue. Once the event occurs, the process is moved back to the ready state.

5. Terminated: The terminated state indicates that a process has finished execution. This could happen because the process has completed its task, explicitly called an exit system call, or has been terminated by the operating system due to an error or other reasons. When a process is terminated, its resources are released, and its PCB is removed.

In addition to these primary states, some operating systems may include additional states or variations. For example, some systems may have a "Suspended" state, where a process is temporarily removed from memory to free up space but can later be resumed. Understanding process states is crucial for operating system designers, developers, and administrators to manage processes effectively and ensure efficient use of system resources.

2.3 Process Control Block:

The Process Control Block (PCB) is a data structure used by operating systems to manage information about a process. It contains various pieces of information that the operating system needs to manage the process effectively. Here are some key components typically found in a PCB:

1. **Process ID (PID):** A unique identifier assigned to each process by the operating system.

2. **Process State:** The current state of the process, such as new, ready, running, blocked, or terminated.

3. **Program Counter (PC):** The address of the next instruction to be executed for the process.

4**. CPU Registers:** The values of CPU registers (e.g., accumulator, stack pointer, etc.) associated with the process.

5**. CPU Scheduling Information:** Information used by the scheduler to determine the priority and scheduling of the process, such as the priority level, scheduling queue pointers, etc.

6. **Memory Management Information:** Information about the process's memory allocation, including base and limit registers, page tables, etc.

7. **I/O Status Information:** Information about I/O devices allocated to the process, including a list of open files, I/O devices in use, etc.

8. **Accounting Information:** Information used for accounting purposes, such as CPU usage, execution time, etc.

9. **Pointer to Parent Process:** A pointer to the process that created this process (if applicable).

The PCB is created and maintained by the operating system for each process and is typically stored in the operating system's memory. When a process is scheduled to run, the operating system loads the PCB of that process into memory and uses it to manage the process's execution. As the process executes, the operating system updates the PCB with the latest information about the process's state and resource usage.

The PCB is a critical data structure in operating systems, as it allows the operating system to manage and control processes effectively, allocate system resources efficiently, and ensure proper coordination between processes.

2.4 Process communication:

Process communication in an operating system refers to the mechanisms and techniques used by processes to exchange information and synchronize their actions. Efficient process communication is essential for coordinating activities between processes running concurrently on a computer system. Here are some common methods of process communication:

1. **Shared Memory:** In shared memory communication, processes share a region of memory that is accessible to all of them. Processes can read from and write to this shared memory region, allowing them to exchange data quickly. However, shared memory requires careful synchronization to avoid conflicts between processes accessing the same memory area simultaneously.

2**. Message Passing:** Message passing involves sending messages between processes using inter-process communication (IPC) mechanisms provided by the operating system. There are two main types of message passing:

-**Direct Communication:** In direct communication, processes must explicitly name each other to send and receive messages. This method requires processes to know each other's identities and can be less flexible.

-Indirect Communication: In indirect communication, messages are sent and received through a shared system mailbox or message queue. Processes do not need to know each other's identities, making the communication more flexible. The operating system manages the message queues and ensures that messages are delivered correctly.

3. Pipes and FIFOs: Pipes and FIFOs (First In, First Out) are used for communication between processes on the same system. Pipes are used for one-way communication, while FIFOs allow bidirectional communication. Pipes and FIFOs are typically used for communication between a parent process and its child processes.

4. Sockets: Sockets are communication endpoints that allow processes to communicate over a network. They are widely used in client-server applications and allow processes running on different machines to exchange data.

5. Signals: Signals are software interrupts used to notify a process that a particular event has occurred. For example, a process can receive a signal when another process completes execution. Signals can be used for simple communication and synchronization between processes.

6. Semaphores: Semaphores are used to synchronize access to shared resources between processes. They can be used to control access to critical sections of code or to coordinate the execution of multiple processes.

These are some of the common methods of process communication in operating systems. The choice of communication method depends on the specific requirements of the application and the desired level of synchronization and coordination between processes.

2.5 Threads:

Threads are a fundamental component of modern operating systems, allowing programs to perform multiple tasks concurrently. A thread is a lightweight process that can be independently scheduled and executed within a process. Here's a detailed explanation of threads in operating systems:

1. Thread Creation: Threads are created within a process using system calls provided by the operating system. When a new thread is created, it shares the same memory space as the parent process, including code, data, and open files. However, each thread has its own stack, which stores local variables and function call information.

2. Thread Execution: Threads within a process can execute concurrently on multiple processors or cores, if available. This concurrency allows for parallelism and can improve the overall performance of the program. The operating system's scheduler is responsible for allocating CPU time to each thread, ensuring that they make progress.

3. Thread States: Threads can be in one of several states, including:

- **Ready:** The thread is ready to execute but is waiting for the scheduler to assign it CPU time.

- **Running:** The thread is currently executing on a CPU.

- **Blocked:** The thread is waiting for a particular event to occur, such as I/O completion or a lock becoming available.

4. Thread Synchronization: Since threads within a process share the same memory space, they can access and modify shared data. This can lead to race conditions and other synchronization issues. Operating systems provide synchronization mechanisms such as mutexes, semaphores, and condition variables to ensure that threads can safely access shared resources.

5. Thread Communication: Threads within a process can communicate with each other using shared variables or other forms of inter-thread communication. This allows threads to coordinate their actions and exchange data.

6. Thread Termination: Threads can terminate either voluntarily or involuntarily. Voluntary termination occurs when a thread reaches the end of its execution or calls an exit system call. Involuntary termination can occur if a thread encounters an error or if the process is terminated.

7. Benefits of Threads: Threads provide several benefits, including:

- **Improved Performance:** Threads can take advantage of multiple processors or cores, leading to faster execution.

- **Resource Sharing:** Threads within a process can share resources such as memory, files, and other system resources.

- **Simplified Programming:** Threads can simplify the programming of concurrent tasks compared to using multiple processes.

Overall, threads are a powerful mechanism for achieving concurrency in operating systems, allowing programs to perform multiple tasks simultaneously and efficiently utilize system resources.

2.6 Process Scheduling:

Process scheduling in an operating system is the process of selecting the next process to run on a CPU from the ready queue. The goal of process scheduling is to maximize CPU utilization, minimize waiting time, and ensure fair allocation of CPU time among processes. Here is an explanation of the key concepts and algorithms used in process scheduling:

1. Scheduling Queues: In a multitasking operating system, processes are typically organized into several queues based on their current state. The main queues include:

- Job Queue: Contains all processes in the system, including both those waiting to be admitted and those currently being executed.

- Ready Queue: Contains processes that are ready to run but are waiting for CPU time.

- Waiting Queue: Contains processes that are waiting for some event to occur, such as I/O completion or a semaphore signal.

2. Scheduling Criteria: Process scheduling algorithms are evaluated based on various criteria, including:

- CPU Utilization: The percentage of time the CPU is busy executing processes.

- Throughput: The number of processes completed per unit of time.

- Turnaround Time: The total time taken to execute a process from submission to completion.

- Waiting Time: The total time a process spends waiting in the ready queue.

- Response Time: The time taken for a process to respond to a user input.

3. Scheduling Algorithms: There are several scheduling algorithms used by operating systems to determine the order in which processes are executed. Some common algorithms include:

- First-Come, First-Served (FCFS): Processes are executed in the order they arrive in the ready queue. This algorithm is simple but can lead to poor performance, especially for long-running processes.

- Shortest Job Next (SJN) or Shortest Job First (SJF): The process with the smallest expected CPU burst is selected for execution. This algorithm minimizes average waiting time but requires knowledge of the burst times, which is often not available.

- **Priority Scheduling:** Each process is assigned a priority, and the process with the highest priority is selected for execution. Priority can be static or dynamic based on various factors. This algorithm can suffer from starvation if low-priority processes never get a chance to execute.

- **Round Robin (RR):** Each process is assigned a fixed time slice (quantum) during which it can execute. If a process does not complete within its time slice, it is moved to the end of the ready queue. RR is simple and ensures fairness, but it may lead to high context-switching overhead.

4. Multilevel Queue Scheduling: This approach divides the ready queue into multiple queues, each with its own scheduling algorithm. Processes are assigned to a queue based on their characteristics, and each queue has a different priority level. This approach is used to prioritize different types of processes and provide better service to high-priority processes.

5. Multilevel Feedback Queue Scheduling: This is an extension of multilevel queue scheduling where processes can move between queues based on their behavior. For example, a process that uses too much CPU time may be moved to a lower-priority queue to give other processes a chance to run.

Process scheduling is a critical component of operating systems, as it directly impacts system performance, responsiveness, and resource utilization. Operating systems use a variety of scheduling algorithms and techniques to balance these factors and provide efficient CPU scheduling for processes.

2.7 Schedulers:

Schedulers in operating systems play a crucial role in managing resources efficiently and executing tasks in a timely manner. There are several types of schedulers used in operating systems, each with its own characteristics and objectives:

1. **Long-Term Scheduler (Job Scheduler):** This scheduler selects processes from the pool of new processes and loads them into the ready queue for execution. Its main goal is to control the degree of multiprogramming to optimize system performance.

2**. Short-Term Scheduler (CPU Scheduler):** The short-term scheduler selects a process from the ready queue and allocates CPU time to it for execution. It aims to achieve fairness, minimize response time, and maximize throughput.

3. **Medium-Term Scheduler:** This scheduler is responsible for swapping processes between main memory and secondary storage (e.g., disk). It helps in managing memory

resources efficiently and handling processes that are ready but cannot be accommodated in main memory.

4. **Real-Time Scheduler:** Real-time schedulers are designed for time-critical systems where tasks must meet strict deadlines. They prioritize tasks based on their deadlines and ensure timely execution.

5. **Multi-Level Feedback Queue Scheduler:** This scheduler uses multiple queues with different priority levels and employs feedback mechanisms to adjust priorities based on process behavior. It aims to balance response time and throughput for interactive and batch processes.

6. **Round Robin Scheduler:** In this scheduler, processes are assigned CPU time slices, and they are executed in a circular order. It ensures fairness by allowing each process to get a share of CPU time.

7. **Priority-Based Scheduler:** Priority-based schedulers assign priorities to processes, and higher priority processes are given preference in CPU allocation. However, they need mechanisms to prevent starvation of lower priority processes.

These schedulers work together to manage system resources effectively, balance workload, and meet performance objectives based on the specific requirements of the operating system and the applications running on it.

2.8 Process Synchronization:

Process synchronization in operating systems refers to the coordination and control of concurrent processes to ensure that they access shared resources in a mutually exclusive and orderly manner. This is essential to prevent race conditions, data inconsistency, and other concurrency-related issues. Here are some common mechanisms used for process synchronization:

1. Mutual Exclusion: Ensuring that only one process at a time can access a critical section of code or a shared resource. This is typically achieved using locks, semaphores, or other synchronization primitives**.**

2. Semaphore: A semaphore is a synchronization primitive that maintains a count to control access to resources. It can be binary (mutex) or counting semaphores. Processes can perform operations like wait (P) and signal (V) on semaphores to synchronize access to shared resources.

3. Mutex (Mutual Exclusion Lock): A mutex is a locking mechanism used to ensure that only one process can access a resource or critical section at a time. It provides exclusive access and is often used to protect shared data structures.

4. Monitors: Monitors are high-level synchronization constructs that encapsulate data and procedures into a single entity. They provide mutual exclusion and condition variables to synchronize access to shared data within the monitor.

5. Condition Variables: Condition variables are used in conjunction with mutexes to manage the execution flow of processes based on certain conditions. Processes can wait on a condition variable until a condition is met, and other processes can signal the variable to notify waiting processes.

6. Spinlock: A spinlock is a synchronization primitive where a process repeatedly checks for the lock until it becomes available. This can be efficient in situations where the lock is expected to be held for a short duration.

7. Barrier: A barrier is a synchronization construct that ensures all participating processes reach a certain point in code before any of them can proceed further. It is commonly used in parallel computing and synchronization of parallel tasks.

These synchronization mechanisms help maintain order, prevent data corruption, and ensure the correct execution of concurrent processes in operating systems. The choice of synchronization technique depends on factors such as the type of shared resource, the level of concurrency, and performance considerations.

2.9 Critical Section Problem:

The critical section problem is a fundamental issue in operating systems and concurrent programming. It revolves around ensuring that multiple processes or threads can access shared resources or critical sections of code in a mutually exclusive and synchronized manner to avoid race conditions and maintain data integrity. Here are the key components and concepts related to the critical section problem:

1. Critical Section: This refers to a segment of code or a region in a program where shared resources (e.g., variables, data structures) are accessed and manipulated. It is essential to ensure that only one process or thread can execute within the critical section at any given time to prevent concurrent access issues.

2. Mutual Exclusion: Achieving mutual exclusion is a core objective of solving the critical section problem. It means that only one process can be executing in the critical section at a time. This prevents multiple processes from simultaneously modifying shared data and causing data corruption or inconsistencies.

3. Progress: Processes should not be prevented from entering their critical sections indefinitely, even if other processes are currently executing in their critical sections. Progress ensures that processes waiting to enter the critical section eventually get a chance to do so.

4. Bounded Waiting: Bounded waiting implies that there exists a limit on the number of times other processes can enter their critical sections after a process requests entry to its critical section. This prevents a process from being starved indefinitely by ensuring fairness in accessing shared resources.

To address the critical section problem, various synchronization mechanisms and algorithms have been developed. Some of the commonly used solutions include:

Locks and Mutexes: Using locks (binary semaphores) or mutexes to provide mutual exclusion by allowing only one process to acquire the lock and enter the critical section while others wait.

Semaphores: Employing counting semaphores to control access to critical sections, where the semaphore value indicates the number of available resources or slots**.**

Monitors: Using monitors, which are high-level synchronization constructs that encapsulate data and procedures, to enforce mutual exclusion and provide condition variables for synchronization.

Spinlocks: Using spinlocks, where a process repeatedly checks for the lock (busy-waiting) until it becomes available. Spinlocks are efficient for short critical sections or on systems with low contention.

Software-based solutions: Implementing algorithms such as Peterson's algorithm, Dekker's algorithm, or the bakery algorithm, which are designed to provide mutual exclusion, progress, and bounded waiting properties.

Each of these solutions has its advantages, disadvantages, and suitability for different scenarios based on factors like concurrency level, resource utilization, and performance requirements. Choosing the appropriate synchronization mechanism is crucial for ensuring correct and efficient handling of the critical section problem in operating systems.

2.11 Classic Problems of Synchronization:

There are several classic synchronization problems in operating systems and concurrent programming that illustrate common challenges and solutions in managing shared resources and coordinating concurrent processes. These problems often serve as foundational examples for understanding synchronization techniques and algorithms. Here are some of the classic synchronization problems:

1. The Producer-Consumer Problem:

- Description: In this problem, there are two types of processes, producers, and consumers, that share a common, fixed-size buffer or queue. Producers generate data items and add them to the buffer, while consumers consume (remove) data items from the buffer.

- Challenge: The challenge is to synchronize the producers and consumers to ensure that producers do not add items to a full buffer and consumers do not remove items from an empty buffer, avoiding buffer overflow or underflow.

- Solution: This problem can be solved using synchronization primitives such as semaphores or mutexes to control access to the buffer and manage the buffer's state (empty, full, or with available slots).

2. The Readers-Writers Problem:

- Description: In this problem, multiple processes (readers and writers) access a shared data resource. Readers only read the data without modifying it, while writers both read and modify the data.

- Challenge: The challenge is to allow multiple readers to access the data simultaneously for read operations while ensuring exclusive access for writers during write operations to maintain data consistency.

- Solution: Various solutions exist, including using a readers-writers lock that allows multiple readers but only one writer at a time. Read operations can proceed concurrently unless a writer is active, in which case readers are blocked.

3. The Dining Philosophers Problem:

- Description: This problem involves a group of philosophers sitting around a dining table with a bowl of rice and chopsticks. Each philosopher alternates between thinking and eating. To eat, a philosopher needs both the chopsticks to their left and right.

- Challenge: The challenge is to avoid deadlock (where each philosopher holds one chopstick and waits indefinitely for the other) and starvation (where a philosopher is always unable to acquire both chopsticks).

- Solution: Solutions include using techniques like resource hierarchy (philosophers acquire chopsticks in a predefined order), limiting the number of philosophers allowed to eat simultaneously, or implementing timeouts to prevent deadlock.

4. The Sleeping Barber Problem:

- Description: In this problem, there is a barber shop with a barber chair and a waiting room with a limited number of chairs. Customers arrive at the shop and either wait in the waiting room if there are free chairs or leave if the waiting room is full.

- Challenge: The challenge is to synchronize the interactions between the barber (who alternates between cutting hair and waiting for customers) and the customers (who either wait or leave based on availability of chairs).

- Solution: Solutions involve using synchronization primitives to manage access to the waiting room (e.g., using semaphores or monitors) and signaling mechanisms to notify the barber when a customer arrives or leaves.

These classic synchronization problems demonstrate common scenarios in concurrent programming where careful synchronization and resource management are essential to avoid race conditions, deadlocks, and other concurrency issues. Solutions often involve using synchronization primitives, such as semaphores, mutexes, condition variables, and various synchronization algorithms, to coordinate the activities of concurrent processes and ensure correct and efficient execution.

UNIT-III

Memory Management and Virtual Memory

3.1 LOGICAL & PHYSICAL ADDRESS SPACE:

In operating systems, the concepts of logical and physical address space are fundamental to understanding how memory management works.

Logical Address Space:

Logical Address Space refers to the set of addresses that a process can use to access memory. These addresses are generated by the CPU during program execution and are also known as virtual addresses. The logical address space is a part of the virtual memory concept, where each process has its own separate address space. This separation ensures that one process cannot directly access the memory of another process, providing isolation and security.

Virtual Memory: The logical address space enables the use of virtual memory, which allows the system to use disk storage to extend the apparent size of RAM.

Address Translation: Logical addresses must be translated to physical addresses through a process known as address translation, which is typically handled by the Memory Management Unit (MMU).

Physical Address Space:

Physical Address Space refers to the actual addresses in the computer's main memory (RAM). These addresses are used by the hardware to access data stored in memory. The physical address space is the set of all physical memory addresses available on the system**.**

Memory Management Unit (MMU): The MMU handles the translation from logical (virtual) addresses to physical addresses. This translation is done using a combination of page tables and hardware mechanisms.

Page Tables: These are data structures used by the operating system to manage the mapping between logical and physical addresses.

3.2 SWAPPING:

Swapping is a memory management technique used by operating systems to manage the limited amount of physical memory (RAM) available. It involves moving processes between the main memory and a storage device (typically a hard disk) to ensure that the system can handle multiple processes concurrently, even if the total memory required exceeds the physical memory available.

Key Concepts of Swapping

1. Swapping In:

- This is the process of moving a process from the disk to the main memory so that it can be executed.
- When a process is swapped in, its state is restored, and it can continue execution from where it left off.

2. Swapping Out:

- This is the process of moving a process from the main memory to the disk to free up memory for other processes.
- The state of the process is saved to disk so that it can be restored later when the process is swapped back in.

How Swapping Works

1. Process Execution:

- When a process is executed, it is loaded into the main memory.
- If there isn't enough memory available, the OS may decide to swap out an inactive or lower-priority process to free up space.

2. Context Saving:

Before swapping a process out, the OS saves the process's state (e.g., CPU registers, program counter, memory contents) to a designated area on the disk, often called the swap space or swap file.

3. Loading Swapped Process:

- When the swapped-out process needs to be executed again, it is swapped back into the main memory.
- The OS restores the process's state, allowing it to resume execution as if it had never been interrupted.

Advantages of Swapping

- **Memory Management:** Swapping allows the OS to manage memory more effectively by freeing up physical memory for active processes.
- **Multitasking:** It enables multitasking by ensuring that multiple processes can share the limited physical memory resources.
- **Handling Large Processes:** Processes that require more memory than is physically available can still run, as portions of their memory can be swapped in and out as needed.

Disadvantages of Swapping

- **Performance Overhead:** Swapping can introduce significant performance overhead due to the time required to read from and write to the disk.
- **Disk Wear:** Frequent swapping can lead to increased wear and tear on the disk, particularly on SSDs.
- **Complexity:** The process of managing which processes to swap in and out adds complexity to the OS.

Swapping is a technique used by operating systems to manage memory more effectively by moving processes between the main memory and disk storage. It enables multitasking and efficient memory usage but can introduce performance overhead and complexity. By understanding swapping, one can better appreciate the mechanisms that allow modern operating systems to handle multiple processes efficiently.

3.3 CONTIGUOUS ALLLOCATION :

Contiguous allocation is a memory management technique used by operating systems to allocate a single contiguous block of memory to a process. This method ensures that all memory addresses assigned to a process are consecutive, which simplifies memory management but also has some limitations.

REFER NOTES FOR DIAGRAM & EXPLANATION

3.4 PAGING:

REFER NOTES FOR EXPLANATION

Advantages of Paging

- **Eliminates External Fragmentation:** Since pages and frames are fixed in size, there is no external fragmentation.
- **Efficient Memory Use:** Non-contiguous allocation of pages allows for more efficient use of memory.
- **Simplifies Allocation:** Easier to find free frames in physical memory as any free frame can be used.

Disadvantages of Paging

- **Internal Fragmentation:** Fixed-size pages can lead to some wasted space within pages if the process does not use the entire page.
- **Overhead:** The use of page tables introduces additional overhead in terms of memory and processing.
- **Page Table Management:** Large logical address spaces require large page tables, which can be complex to manage.

3.5 STRUCTURE OF PAGE TABLE:

The page table is a critical data structure used in operating systems to manage virtual memory. It maps virtual addresses (pages) to physical addresses (frames). The structure of the page table can vary depending on the specific implementation and the needs of the operating system. Below are the key components and common structures of a page table.

Key Components of a Page Table Entry (PTE)

Each entry in a page table typically contains the following information:

1. Frame Number:

The frame number indicates which physical frame in memory the page is mapped to.

2. Present/Absent Bit:

This bit indicates whether the page is currently in physical memory. If the bit is 0, the page is not in memory and might be on disk.

3. Protection Bits:

These bits specify the permissions for the page, such as read, write, and execute permissions.

4. Modified (Dirty) Bit:

This bit indicates whether the page has been modified since it was last loaded into memory. If it has been modified, it must be written back to disk before being replaced.

5. Accessed Bit:

This bit indicates whether the page has been accessed recently. It is used for implementing page replacement algorithms.

6. Cache Disable Bit:

This bit can disable caching for the page, which might be necessary for certain types of memory-mapped I/O.

1. Simple Single-Level Page Table:

In a simple paging system, a single-level page table is used, where the page table is a linear array indexed by the virtual page number.

Structure

Index: Virtual Page Number

Value: Physical Frame Number + Control Bits

2. Multilevel (Hierarchical) Page Tables:

For systems with large address spaces, a single-level page table can become excessively large. Multilevel page tables break the page table into multiple levels, reducing the size of each table and making memory management more efficient.

a.) Two-Level Page Table:

- The virtual address is divided into three parts: the first part indexes the outer page table, the second part indexes the inner page table, and the third part is the offset within the page.

Structure:

Outer Page Table: Points to inner page tables.

Inner Page Tables: Map virtual pages to physical frames.

Address Breakdown:

Outer Index: First part of the virtual address.

Inner Index: Second part of the virtual address.

Offset: Third part of the virtual address.

b). Three-Level (or More) Page Table:

For even larger address spaces, additional levels can be added, following the same principle.

Inverted Page Table

An inverted page table uses a different approach by having a single page table for the entire physical memory, with one entry for each frame of physical memory. This structure is more memory efficient for systems with large address spaces.

Structure:

Index: Physical Frame Number

Value: Virtual Page Number + Process Identifier + Control Bits

3, Hashed Page Tables

Hashed page tables use a hash table to handle large, sparse address spaces. The virtual page number is hashed to find the corresponding page table entry.

Structure:

Hash Table: Maps the hash of the virtual page number to the page table entry.

Linked List (optional): Handles collisions in the hash table.

Example of Address Translation with a Two-Level Page Table

Assume a virtual address is 32 bits, with 10 bits for the outer page table, 10 bits for the inner page table, and 12 bits for the offset.

1. Virtual Address Breakdown:

   Outer Index: Bits 31-22

   Inner Index: Bits 21-12

   Offset: Bits 11-0

2. Translation Steps

   - Extract the outer index from the virtual address.
   - Use the outer index to find the inner page table.
   - Extract the inner index from the virtual address.
   - Use the inner index to find the frame number.
   - Combine the frame number with the offset to get the physical address.

The structure of a page table in an operating system is designed to efficiently map virtual addresses to physical addresses. Simple single-level page tables are easy to understand but not suitable for large address spaces. Multilevel page tables, inverted page tables, and hashed page tables offer more scalable and efficient solutions for modern systems with extensive memory requirements. Each approach has its own trade-offs in terms of complexity, memory usage, and performance.

3.6 SEGMENTATION:

Segmentation is a memory management technique in operating systems that divides the memory into variable-sized segments. Each segment is a logical unit, such as a function, an array, or a data structure, which the operating system manages independently. This method provides a way to handle memory in a more flexible and organized manner, as segments represent the logical division of a program.

1. Segment:

A segment is a logical unit of memory, which could be a code segment, data segment, stack segment, etc. Each segment can vary in size.

2. Segment Table:

Each process has a segment table that stores the base address and limit (size) of each segment. The base address indicates where the segment begins in physical memory, and the limit specifies the length of the segment.

3. Logical Address:

A logical address in a segmented system consists of a segment number and an offset within that segment.

Advantages of Segmentation:

Logical Organization: Segmentation reflects the logical structure of a program, making it easier to manage and understand.

Protection: Different segments can have different access rights, enhancing security.

Sharing: Segments can be shared among processes, which is useful for shared libraries and data.

Dynamic Size: Segments can grow or shrink dynamically, providing flexibility.

Disadvantages of Segmentation

External Fragmentation: As segments are of variable sizes, they can lead to external fragmentation, making memory allocation less efficient.

Complexity: Managing and keeping track of segments adds complexity to the memory management system.

Segmentation Faults: Incorrect segment number or offset can result in segmentation faults, causing program crashes.

Segmentation is a memory management technique that divides memory into variable-sized segments based on logical divisions within a program. While it offers benefits in terms of logical organization, protection, and flexibility, it also has drawbacks like external fragmentation and added complexity. Combining segmentation with paging can provide a more efficient and flexible memory management solution, leveraging the strengths of both methods.

3.7 Segmentation with Paging

Segmentation with paging is a hybrid memory management scheme that combines the advantages of both segmentation and paging. This approach helps to mitigate the drawbacks associated with each method when used independently. It is particularly useful in systems that require both logical division of memory (segmentation) and efficient use of physical memory (paging).

How Segmentation with Paging Works

In this scheme, the logical address space is divided into segments, and each segment is further divided into pages. The address translation process involves two levels of mapping: segment to page and page to frame.

Address Structure

A logical address in a segmented paging system is typically divided into three parts:

1. Segment Number (s): Identifies the segment.

2. Page Number (p):Identifies the page within the segment.

3. Offset (d): Identifies the specific location within the page.

1. Segment Table:

- Each entry in the segment table points to a page table for that segment.
- The segment table entry contains the base address of the page table and the limit (size) of the segment.

2. Page Table:

- Each entry in the page table maps a page number to a physical frame number.
- The page table entry contains the frame number and control bits (e.g., present/absent, protection bits).

Address Translation Process

The translation of a logical address to a physical address involves the following steps:

1. Extract the Segment Number:

- Use the segment number (s) to index into the segment table.
- Retrieve the base address of the page table for that segment and the segment limit.

2. Check Segment Limit:

Ensure the page number (p) and offset (d) are within the segment limit. If not, a segmentation fault occurs.

3. Extract the Page Number:

- Use the page number (p) to index into the page table of the segment.
- Retrieve the frame number.

4. Form the Physical Address:

Combine the frame number with the offset (d) to form the complete physical address.

Advantages of Segmentation with Paging

- Logical Organization: Segmentation provides a logical division of programs, making it easier to manage and understand.
- Efficient Memory Use: Paging ensures efficient use of physical memory by eliminating external fragmentation.
- Protection and Sharing: Both segments and pages can have different access rights, enhancing security and allowing for shared memory.

Disadvantages of Segmentation with Paging

- Complexity: The combined scheme is more complex to implement and manage compared to using segmentation or paging alone.
- Overhead: Maintaining both segment tables and page tables introduces additional memory and computational overhead.

Segmentation with paging is a hybrid memory management technique that combines the logical organization benefits of segmentation with the efficient physical memory usage of paging. By breaking down segments into pages, this approach helps to reduce fragmentation and provides a flexible, efficient way to manage memory. The trade-off is increased complexity and overhead, but the advantages often outweigh these drawbacks in modern operating systems.

3.8 Demand Paging

Demand paging is a memory management technique used in operating systems to load pages into memory only when they are needed (i.e., on demand). This method helps to efficiently utilize memory by keeping only the necessary pages in physical memory, thereby allowing more processes to run concurrently.

1. **Page Fault:**

A page fault occurs when a program tries to access a page that is not currently in physical memory. The operating system then handles the fault by loading the required page from secondary storage (disk) into physical memory.

2. **Lazy Loading:**

Instead of loading all pages of a process into memory at the start, only the pages that are immediately required are loaded. Subsequent pages are loaded as and when needed.

3. **Swapping:**

   If physical memory is full and a new page needs to be loaded, the operating system may swap out a page from memory to disk to make space for the new page.

How Demand Paging Works

1. **Process Initialization:**

When a process is created, only a minimal number of pages (such as the pages containing the start code and initial stack) are loaded into memory. The rest of the pages are marked as not present in memory in the page table.

2. **Page Access:**

When the process accesses a page that is not in memory, a page fault occurs. The operating system's page fault handler is invoked.

3. **Page Fault Handling:**

The page fault handler determines whether the access is valid.

If the access is invalid (e.g., accessing an address outside the process's address space), the process is terminated.

If the access is valid, the operating system finds the page on disk and loads it into memory.

4. **Page Replacement:**

If there is no free memory available, the operating system must select a page to swap out using a page replacement algorithm (e.g., Least Recently Used (LRU), First-In-First-Out (FIFO), etc.).

The selected page is written to disk if it has been modified (dirty), and its page table entry is updated to reflect that it is no longer in memory.

5. **Update Page Table:**

The page table is updated to reflect the new page's location in memory, and the instruction that caused the page fault is restarted.

Advantages of Demand Paging

Efficient Memory Use:

Only the necessary pages are loaded into memory, allowing more processes to run concurrently and reducing the overall memory usage.

Faster Process Start-up:

Processes start faster since not all pages need to be loaded initially.

Flexibility:

The operating system can handle larger processes that do not fit entirely in physical memory.

Disadvantages of Demand Paging

Page Fault Overhead:

Handling page faults incurs overhead due to the time required to load pages from disk and update page tables.

Performance Impact:

Frequent page faults (thrashing) can significantly degrade performance, especially if the working set of a process does not fit into physical memory.

Complexity:

The implementation of demand paging and page replacement algorithms adds complexity to the operating system.

3.9 Page Replacement

REFER NOTES

3.10 Page Replacement Algorithms

Several algorithms can be used to determine which page to replace when memory is full:

1. **First-In-First-Out (FIFO):**

Replaces the oldest page in memory, i.e., the one that was loaded first.

The oldest page in memory (the one that was loaded first) is replaced.

Simple but may not be efficient, as the oldest page might still be frequently accessed.

FIFO Queue:

[Page 1, Page 2, Page 3] (oldest to newest)

REFER NOTES

2. **Least Recently Used (LRU):**

Replaces the page that has not been used for the longest time.

Replaces the page that has not been used for the longest time.

Based on the assumption that pages used recently will likely be used again soon.

LRU Order:

[Page 2, Page 3, Page 1] (least to most recently used)

REFER NOTES

3. **Optimal Page Replacement:**

Replaces the page that will not be used for the longest time in the future (requires knowledge of future requests, typically used for theoretical analysis).

Replaces the page that will not be used for the longest period in the future.

Provides the best performance but requires future knowledge of page references, making it impractical for actual implementation.

Reference string: [7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3]

Replace the page that will be used furthest in the future.

REFER NOTES

3.11 Allocation of Frames:

REFER NOTES

UNIT IV

File Concept:

Computers can store information on various storage media such as, magnetic disks, magnetic tapes, optical disks. The physical storage is converted into a logical storage unit by operating system. The logical storage unit is called FILE. A file is a collection of similar records. A record is a collection of related fields that can be treated as a unit by some application program. A field is some basic element of data. Any individual field contains a single value. A data base is collection of related data.

Student	Marks	Marks	Fail/Pas
KUMA	85	86	P
LAKSH	93	92	P

DATA FILE

Student name, Marks in sub1, sub2, Fail/Pass is fields. The collection of fields is called a RECORD.

RECORD:

LAKSH	93	92	P

Collection of these records is called a data file.

FILE ATTRIBUTES :

1. Name : A file is named for the convenience of the user and is referred by its name. A name is usually a string of characters.
2. Identifier : This unique tag, usually a number ,identifies the file within the file system.
3. Type : Files are of so many types. The type depends on the extension of the file.

Example:
.exe Executable file
.obj Object file
.src Source file

4. Location : This information is a pointer to a device and to the location of the file on that device.
5. Size : The current size of the file (in bytes, words,blocks).
6. Protection : Access control information determines who can do reading, writing, executing and so on.
7. Time, Date, User identification : This information may be kept for creation, last modification,last use.

FILE OPERATIONS

1. Creating a file : Two steps are needed to create a file. They are:
   - *Check whether the space is available ornot.*
   - *If the space is available then made an entry for the new file in the directory. The entry includes name of the file, path of the file,etc…*
2. Writing a file: To write a file, we have to know 2 things. One is name of the file and second is the information or data to be written on the file, the system searches the entired given location for the file. If the file is found, the system must keep a write pointer to the location in the file where the next write is to take place.

3. Reading a file: To read a file, first of all we search the directories for the file, if the file is found, the system needs to keep a read pointer to the location in the file where the next read is to take place. Once the read has taken place, the read pointer is updated. 4. Repositioning within a file: The directory is searched for the appropriate entry and the current file position pointer is repositioned to a given value. This operation is also called file seek.

5. Deleting a file: To delete a file, first of all search the directory for named file, then released the file space and erase the directory entry.

6. Truncating a file: To truncate a file, remove the file contents only but, the attributes are as itis.

FILE TYPES:

The name of the file split into 2 parts. One is name and second is Extension. The file type is depending on extension of the file.

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

FILE STRUCTURE

File types also can be used to indicate the internal structure of the file. The operating system requires that an executable file have a specific structure so that it can determine where in memory to load the file and what the location of the first instruction is. If OS supports multiple file structures, the resulting size of OS is large. If the OS defines 5 different file structures, it needs to contain the code to support these file structures. All OS must support at least one structure that of an executable file so that the system is able to load and run programs.

INTERNAL FILE STRUCTURE

In UNIX OS, defines all files to be simply stream of bytes. Each byte is individually addressable by its offset from the beginning or end of the file. In this case, the logical record size is 1 byte. The file system automatically packs and unpacks bytes into physical disk blocks, say 512 bytes per block.

The logical record size, physical block size, packing determines how many logical records are in each physical block. The packing can be done by the user's application program or OS. A file may be considered a sequence of blocks. If each block were 512 bytes, a file of 1949 bytes would be allocated 4 blocks (2048 bytes). The last 99 bytes would be wasted. It is called internal fragmentation all file systems suffer from internal fragmentation, the larger the block size, the greater the internal fragmentation.

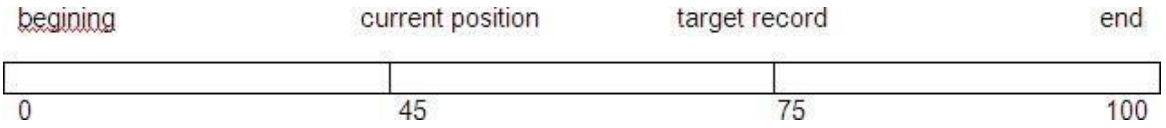
FILE ACCESS METHODS

Files stores information, this information must be accessed and read into computer memory. There are so many ways that the information in the file can be accessed.

1. Sequential file access:

Information in the file is processed in order i.e. one record after the other. Magnetic tapes are supporting this type of file accessing.

Eg : A file consisting of 100 records, the current position of read/write head is 45th record, suppose we want to read the 75th record then, it access sequentially from 45, 46, 47 …….. 74, 75. So the read/write head traverse all the records between 45 to 75.



2. Direct access:

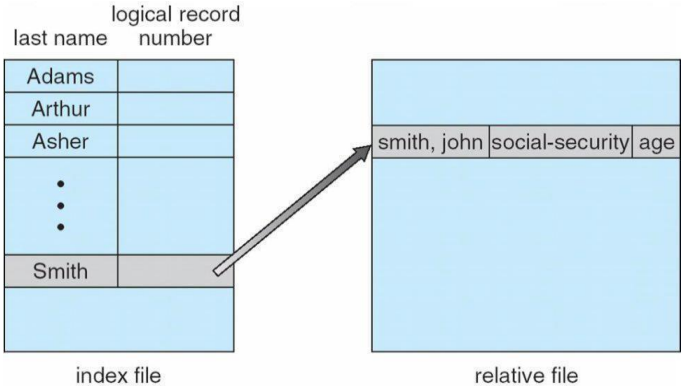
Direct access is also called relative access. Here records can read/write randomly without any order. The direct access method is based on a disk model of a file, because disks allow random access to any file block.

Eg : A disk containing of 256 blocks, the position of read/write head is at 95th block. The block is to be read or write is 250th block. Then we can access the 250th block directly without any restrictions.

Eg : CD consists of 10 songs, at present we are listening song 3, If we want to listen song 10, we can shift to 10.

3. Indexed Sequential File access

The main disadvantage in the sequential file is, it takes more time to access a Record . Records are organized in sequence based on a key field.
Eg :
A file consisting of 60000 records,the master index divide the total records into 6 blocks, each block consisiting of a pointer to secondary index.The secondary index divide the 10,000 records into 10 indexes.Each index consisting of a pointer to its orginal location.Each record in the index file consisting of 2 field, A key field and a pointer field.



DIRECTORY STRUCTURE

Sometimes the file system consisting of millions of files,at that situation it is very hard to manage the files. To manage these files grouped these files and load one group into one partition.

Each partition is called a directory .a directory structure provides a mechanism for organizing many files in the file system.

OPERATION ON THE DIRECTORIES :

1. Search for a file : Search a directory structure for required file.

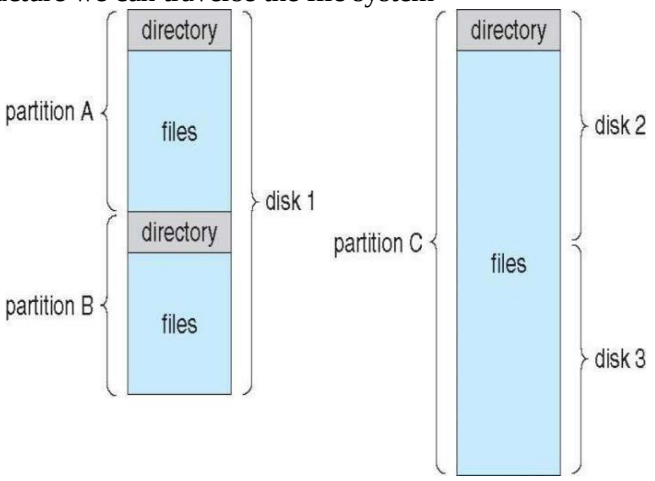
2. createafile : New files need to be created, added to the directory.

3. Deleteafile : When a file is no longer needed, we want to remove it from the directory.

4. List adirectory : We can know the list of files in thedirectory.

5. Renameafile : Whenever we need to change the name of the file, we can change The name.

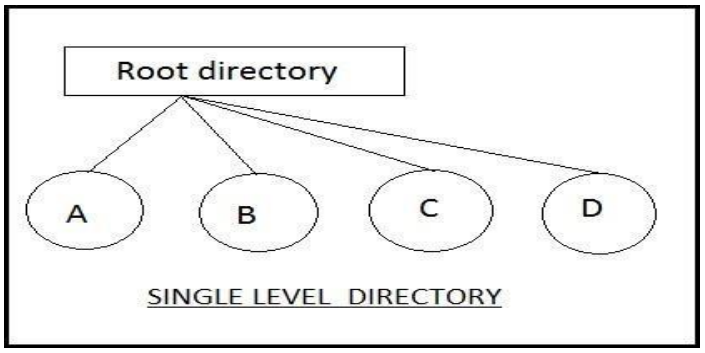
6. Traverse the file system : We need to access every directory and every file with in a directory structure we can traverse the file system



The various directory structures

1. Single level directory:

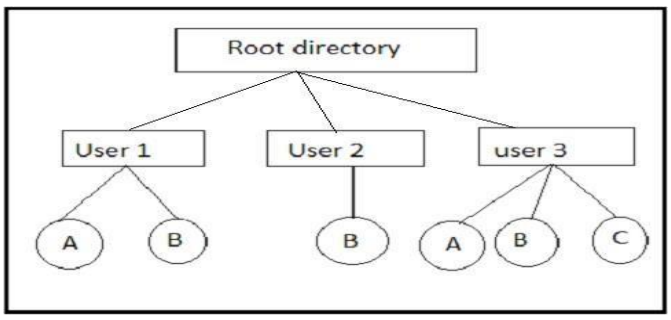
The directory system having only one directory,it consisting of all files some times it is said to be root directory.



E.g :- Here directory containing 4 files (A,B.C,D).the advantage of the scheme is its simplicity and the ability to locate files quickly.The problem is different users may accidentally use the same names for their files.
E.g :- If user 1 creates a files caled sample and then later user 2 to creates a file called sample,then user2's file will overwrite user 1 file.Thats why it is not used in the multi user system.

2. Two level directory:

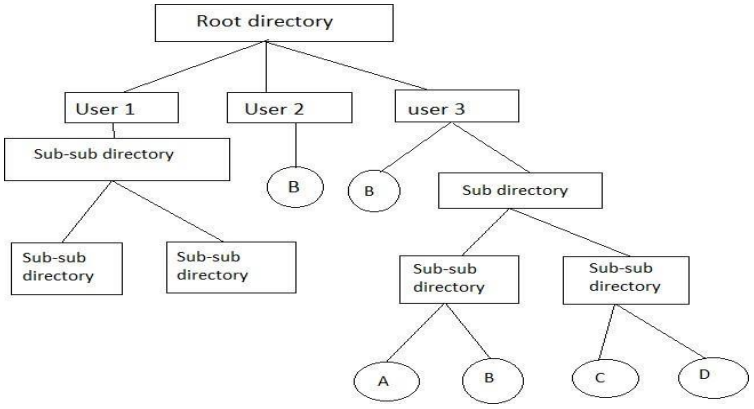
The problem in single level directory is different user may be accidentally use the same name for their files. To avoid this problem each user need a private directory,
Names chosen by one user don't interfere with names chosen by a different user.



Root directory is the first level directory.user 1,user2,user3 are user level of directory A,B,C are files.

3. Tree structured directory:

Two level directory eliminates name conflicts among users but it is not satisfactory for users with a large number of files.To avoid this create the subdirectory and load the same type of files into the sub-directory.so, here each can have as many directories are needed.



There are 2 types of path

1. Absoulte path
2. Relative path

Absoulte path : Begging with root and follows a path down to specified files giving directory, directory name on the path.

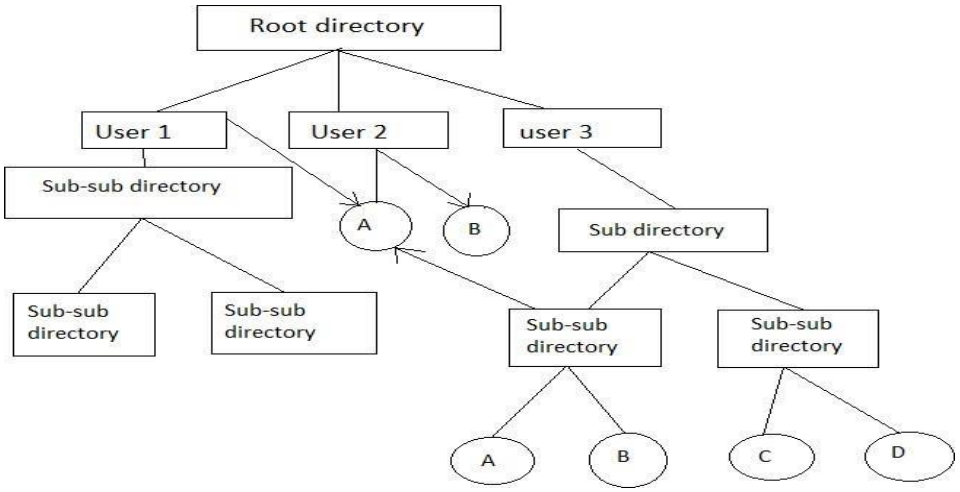
Relative path : A path from current directory.

4. Acyclic graphdirectory

Multiple users are working on a project, the project files can be stored in a comman sub-directory of the multiple users. This type of directory is called acyclic graph directory .The common directory will be declared a shared directory. The graph contain no cycles with shared files, changes made by one user are made visible to other users.A file may now have multiple absolute paths. when shared directory/file is deleted, all pointers to the directory/ files also to be removed.

5. General graph directory:

When we add links to an existing tree structured directory, the tree structure is destroyed, resulting is a simple graph structure.

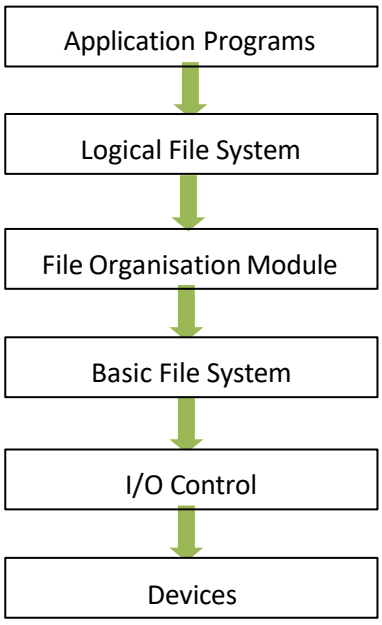


Advantages :- Traversing is easy. Easy sharing is possible.

File system structure:

Disk provides the bulk of secondary storage on which a file system is maintained. They have 2 characteristics that make them a convenient medium for storing multiple files.

1. A disk can be rewritten in place. It is possible to read a block from the disk, modify the block, and write it back into same place.
2. A disk can access directly any block of information it contains.



I/O Control: consists of device drivers and interrupt handlers to transfer information between the main memory and the disk system. The device driver writes specific bit patterns to special locations in the I/O controller's memory to tell the controller which device location to act on and what actions to take.

The Basic File System needs only to issue commands to the appropriate device driver to read and write physical blocks on the disk. Each physical block is identified by its numeric disk address (Eg. Drive 1, cylinder 73, track2, sector 10).

The File Organization Module knows about files and their logical blocks and physical blocks. By knowing the type of file allocation used and the location of the file, file organization module can translate logical block address to physical addresses for the basic file system to transfer. Each file's logical blocks are numbered from 0 to n. so, physical blocks containing the data usually do not match the logical numbers. A translation is needed to locate each block.

The Logical File System manages all file system structure except the actual data (contents of file). It maintains file structure via file control blocks. A file control block (inode in Unix file systems) contains information about the file, ownership, permissions, location of the file contents.

File-System Mounting

• The basic idea behind mounting file systems is to combine multiple file systems into one large tree structure.

• The mount command is given a file system to mount and a mount point (directory) on which to attach it.

• Once a file system is mounted onto a mount point, any further references to that directory actually refer to the root of the mounted file system.

• Any files (or sub-directories) that had been stored in the mount point directory prior to mounting the new file system are now hidden by the mounted file system, and are no longer available. For this reason some systems only allow mounting onto empty directories.

• File systems can only be mounted by root, unless root has previously configured certain file systems to be mountable onto certain pre-determined mount points. (E.g. root may allow users to mount floppy file systems to /mnt or something like it.) Anyone can run the mount command to see what file systems are currently mounted.

• File systems may be mounted read-only, or have other restrictions imposed.

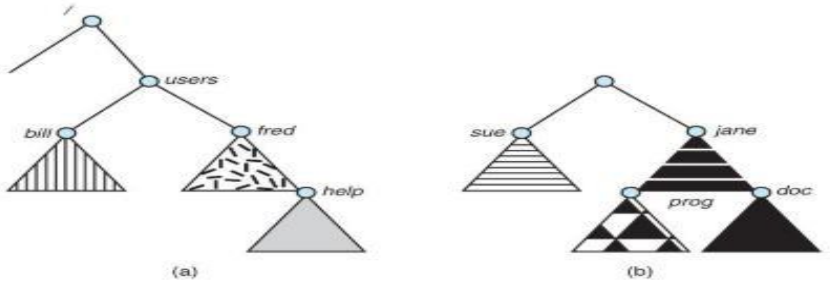


Figure 11.14 - File system. (a) Existing system. (b) Unmounted volume.

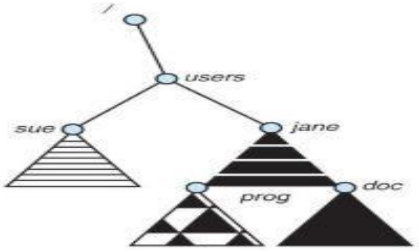


Figure 11.15 - Mount point.

The traditional Windows OS runs an extended two-tier directory structure, where the first tier of the structure separates volumes by drive letters, and a tree structure is implemented below that level.

• Macintosh runs a similar system, where each new volume that is found is automatically mounted and added to the desktop when it is found.

• More recent Windows systems allow file systems to be mounted to any directory in the file system, much like UNIX.

File Sharing

Multiple Users

• On a multi-user system, more information needs to be stored for each file:
o The owner (user) who owns the file, and who can control its access.
o The group of other user IDs that may have some special access to the file.
o What access rights are afforded to the owner (User), the Group, and to the rest of the world (the universe, a.k.a. Others.)
o Some systems have more complicated access control, allowing or denying specific accesses to specifically named users or groups.

Remote File Systems

• The advent of the Internet introduces issues for accessing files stored on remote computers
o The original method was FTP (File Transfer Protocol), allowing individual files to be transported across systems as needed. FTP can be either account or password controlled, or anonymous, not requiring any user name or password.
o Various forms of distributed file systems allow remote file systems to be mounted onto a local directory structure, and accessed using normal file access commands.
o The WWW has made it easy once again to access files on remote systems without mounting their file systems, generally using (anonymous) ftp as the underlying file transport mechanism.

The Client-Server Model

• When one computer system remotely mounts a filesystem that is physically located on another system, the system which physically owns the files acts as a server, and the system which mounts them is the client.
• User IDs and group IDs must be consistent across both systems for the system to work properly
• The same computer can be both a client and a server. (E.g. cross-linked file systems.)
• There are a number of security concerns involved in this model:
o Servers commonly restrict mount permission to certain trusted systems only. Spoofing (a computer pretending to be a different computer) is a potential security risk.
o Servers may restrict remote access to read-only.
o Servers restrict which file systems may be remotely mounted. Generally the information within those subsystems is limited, relatively public, and protected by frequent backups.
• The NFS (Network File System) is a classic example of such a system.

Protection

• Files must be kept safe for reliability (against accidental damage), and protection (against deliberate malicious access.) The former is usually managed with backup copies.
• One simple protection scheme is to remove all access to a file. However this makes the file unusable, so some sort of controlled access must be arranged.

Access Control

In access-control list (ACL) specifying user names and the types of access allowed for each user. When a user requests access to a particular file, the operating system checks the access list associated with that file. If that user is listed for the requested access, the access is allowed. Otherwise, a protection violation occurs, and the user job is denied access to the file.
This technique has two undesirable consequences:

• Constructing such a list may be a tedious and unrewarding task, especially if we do not know in advance the list of users in the system.
• The directory entry, previously of fixed size, now must be of variable size, resulting in more complicated space management.
These problems can be resolved by use of a condensed version of the access list. To condense the length of the access-control list, many systems recognize three classifications of users in connection with each file:
• Owner. The user who created the file is the owner.
• Group. A set of users who are sharing the file and need similar access is a group, or work group.
• Universe. All other users in the system constitute the universe.

To illustrate, consider a person, Sara, who is writing a new book. She has hired three graduate students (Jim, Dawn, and Jill) to help with the project. The text of the book is kept in a file named book.tex. The protection associated with this file is as follows:
• Sara should be able to invoke all operations on the file.
• Jim, Dawn, and Jill should be able only to read and write the file; they should not be allowed to delete the file.
• All other users should be able to read, but not write, the file. (Sara is interested in letting as many people as possible read the text so that she can obtain feedback.)

Types of Access

• The following low-level operations are often controlled:
o Read - View the contents of the file
o Write - Change the contents of the file.
o Execute - Load the file onto the CPU and follow the instructions contained there in.
o Append - Add to the end of an existing file.
o Delete - Remove a file from the system.
o List -View the name and other attributes of files on the system.
• Higher-level operations, such as copy, can generally be performed through combinations of the above.
• UNIX uses a set of 9 access control bits, in three groups of three. These correspond to R, W, and X permissions for each of the Owner, Group, and Others. (See "man chmod" for full details.) The RWX bits control the following privileges for ordinary files and directories:

bit	Files	Directories
R	Read (view) file contents.	Read directory contents. Required to get a listing of the directory.
W	Write (change) file contents.	Change directory contents. Required to create or delete files.
X	Execute file contents as a program.	Access detailed directory information. Required to get a long listing, or to access any specific file in the directory. Note that if a user has X but not R permissions on a directory, they can still access specific files, but only if they already know the name of the file they are trying to access.

• In addition there are some special bits that can also be applied:

o The set user ID (SUID) bit and/or the set group ID (SGID) bits applied to executable files temporarily change the identity of whoever runs the program to match that of the owner / group of the executable program. This allows users running specific programs to have access to files (while running that program) to which they would normally be unable to access. Setting of these

two bits is usually restricted to root, and must be done with caution, as it introduces a potential security leak.

o The sticky bit on a directory modifies write permission, allowing users to only delete files for which they are the owner. This allows everyone to create files in /tmp, for example, but to only delete files which they have created, and not anyone else's.

o The SUID, SGID, and sticky bits are indicated with an S, S, and T in the positions for execute permission for the user, group, and others, respectively. If the letter is lower case, (s, s, t), then the corresponding execute permission is not also given. If it is upper case, (S, S, T), then the corresponding execute permission is given.

o The numeric form of chmod is needed to set these advanced bits.

-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5 pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 jwg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2012	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2012	program
drwx--x--x	4 tag	faculty	512	Jul 31 10:31	lib/
drwx------	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

Sample permissions in a UNIX system.

• Windows adjusts files access through a simple GUI:

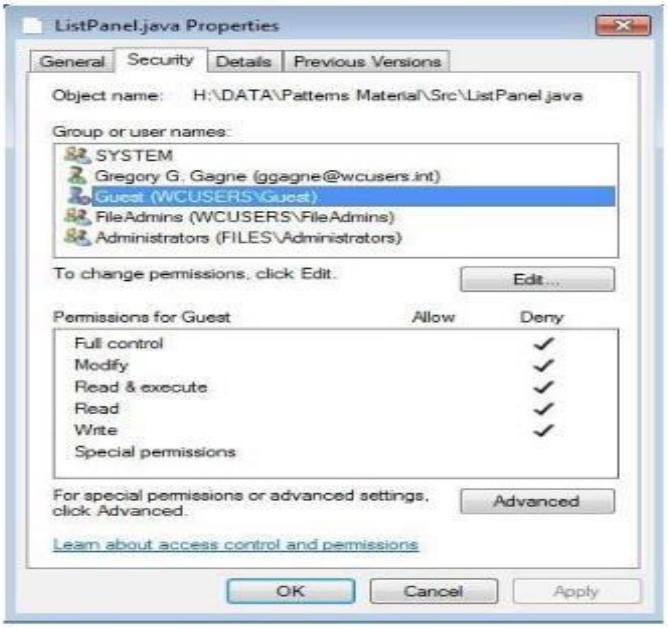


Figure - Windows 7 access-control list management.

Overview of mass storage structure

Magnetic disks provide the bulk of secondary storage for modern computer systems. Conceptually, disks are relatively simple (Figure 10.1). Each disk platter has a flat circular shape, like a CD. Common platter

diameters range from 1.8 to 3.5 inches. The two surfaces of a platter are covered with a magnetic material. We store information by recording it magnetically on the platters

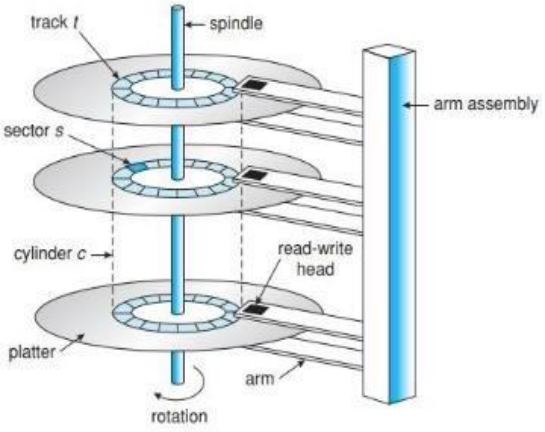


Figure 10.1 Moving-head disk mechanism.

A read–write head "flies" just above each surface of every platter. The heads are attached to a disk arm that moves all the heads as a unit. The surface of a platter is logically divided into circular tracks, which are subdivided into sectors. The set of tracks that are at one arm position makes up a cylinder. There may be thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors. The storage capacity of common disk drives is measured in gigabytes.

When the disk is in use, a drive motor spins it at high speed. Most drives rotate 60 to 250 times per second, specified in terms of rotations per minute (RPM). Common drives spin at 5,400, 7,200, 10,000, and 15,000 RPM. Disk speed has two parts. The transfer rate is the rate at which data flow between the drive and the computer. The positioning time, or random-access time, consists of two parts: the time necessary to move the disk arm to the desired cylinder, called the seek time, and the time necessary for the desired sector to rotate to the disk head, called the rotational latency.

Typical disks can transfer several megabytes of data per second, and they have seek times and rotational latencies of several milliseconds Other forms of removable disks include CDs, DVDs, and Blu-ray discs as well as removable flash-memory devices known as flash drives (which are a type of solid-state drive).

A disk drive is attached to a computer by a set of wires called an I/O bus. Several kinds of buses are available, including advanced technology attachment (ATA), serial ATA (SATA), eSATA, universal serial bus (USB), and fibre channel (FC). The data transfers on a bus are carried out by special electronic processors called controllers. The host controller is the controller at the computer end of the bus. A disk controller is built into each disk drive. To perform a disk I/O operation, the computer places a command into the host controller, typically using memory-mapped I/O ports.

Disk Structure

Modern magnetic disk drives are addressed as large one-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfer. The size of a logical block is usually 512 bytes, although some disks can be low-level formatted to have a different logical block
size, such as 1,024 bytes.

The one-dimensional array of logical blocks is mapped onto the sectors of the disk sequentially. Sector 0 is the first sector of the first track on the outermost cylinder. The mapping proceeds in order through that track, then through the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

Tracks in the outermost zone typically hold 40 percent more sectors than do tracks in the innermost zone. The drive increases its rotation speed as the head moves from the outer to the inner tracks to keep the same rate of data moving under the head. This method is used in CD-ROM 10.3 Disk Attachment 471 and DVD-ROM drives. Alternatively, the disk rotation speed can stay constant; in this case, the density of bits decreases from inner tracks to outer tracks to keep the data rate constant. This method is used in hard disks and is known as constant angular velocity (CAV).

The number of sectors per track has been increasing as disk technology improves, and the outer zone of a disk usually has several hundred sectors per track. Similarly, the number of cylinders per disk has been increasing; large disks have tens of thousands of cylinders.

Disk Attachment

Computers access disk storage in two ways. One way is via I/O ports (or host-attached storage); this is common on small systems. The other way is via a remote host in a distributed file system; this is referred to as network-attached storage.

Host-Attached Storage

- Host-attached storage is storage accessed through local I/O ports. These ports use several technologies. The typical desktop PC uses an I/O bus architecture called IDE or ATA. This architecture supports a maximum of two drives per I/O bus. A newer, similar protocol that has simplified cabling is SATA.
- High-end workstations and servers generally use more sophisticated I/O architectures such as fibre channel (FC), a high-speed serial architecture that can operate over optical fiber or over a four-conductor copper cable. It has two variants. One is a large switched fabric having a 24-bit address space. This variant is expected to dominate in the future and is the basis of storage-area networks (SANs), because of the large address space and the switched nature of the communication, multiple hosts and storage devices can attach to the fabric, allowing great flexibility in I/O communication.
- The other FC variant is an arbitrated loop (FC-AL) that can address 126 devices (drives and controllers). A wide variety of storage devices are suitable for use as host-attached storage. Among these are hard disk drives, RAID arrays, and CD, DVD, and tape drives.

Network-Attached Storage

➢ Network attached storage connects storage devices to computers using a remote procedure call, RPC, interface, typically with something like NFS filesystem mounts. This is convenient for allowing several computers in a group common access and naming conventions for shared storage.

➢ NAS can be implemented using SCSI cabling, or ISCSI uses Internet protocols and standard network connections, allowing long-distance remote access to shared files.

➢ NAS allows computers to easily share data storage, but tends to be less efficient than standard host-attached storage.

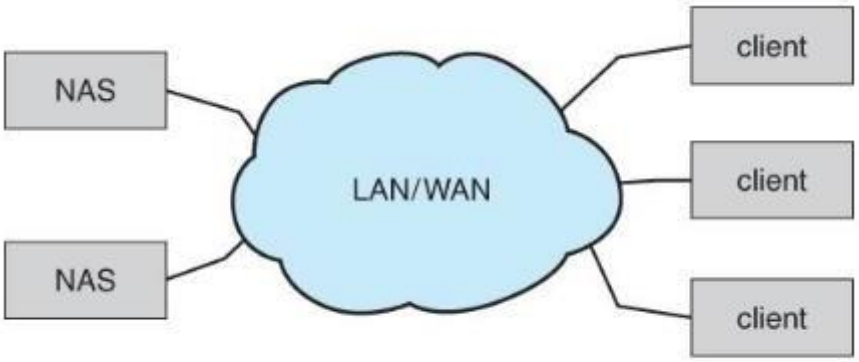


Figure - Network-attached storage.

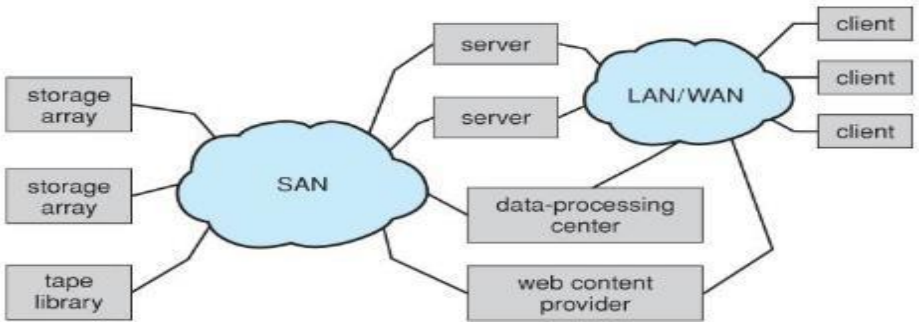
Storage-Area Network

➢ A Storage-Area Network, SAN, connects computers and storage devices in a network, using storage protocols instead of network protocols.

➢ One advantage of this is that storage access does not tie up regular networking bandwidth.

➢ SAN is very flexible and dynamic, allowing hosts and devices to attach and detach on the fly.

➢ SAN is also controllable, allowing restricted access to certain hosts and devices.



Disk Scheduling Algorithms

As we know, a process needs two type of time, CPU time and IO time. For I/O, it requests the Operating system to access the disk. However, the operating system must be fare enough to satisfy each request and at the same time, operating system must maintain the efficiency and speed of process execution. The technique that operating system uses to determine the request which is to be satisfied next is called disk scheduling.

Let's discuss some important terms related to disk scheduling.

Seek Time

Seek time is the time taken in locating the disk arm to a specified track where the read/write request will be satisfied.

Rotational Latency

It is the time taken by the desired sector to rotate itself to the position from where it can access the R/W heads.

Transfer Time

It is the time taken to transfer the data.

Disk Access Time

Disk access time is given as,

Disk Access Time = Rotational Latency + Seek Time + Transfer Time

Disk Response Time

It is the average of time spent by each request waiting for the IO operation.

Purpose of Disk Scheduling

The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed.

Goal of Disk Scheduling Algorithm

- o Fairness
- o High throughout
- o Minimal traveling head time

Disk Scheduling Algorithms

The list of various disks scheduling algorithm is given below. Each algorithm is carrying some advantages and disadvantages. The limitation of each algorithm leads to the evolution of a new algorithm.

- o FCFS scheduling algorithm
- o SSTF (shortest seek time first) algorithm
- o SCAN scheduling
- o C-SCAN scheduling
- o LOOK Scheduling
- o C-LOOK scheduling

FCFS Scheduling Algorithm

It is the simplest Disk Scheduling algorithm. It services the IO requests in the order in which they arrive. There is no starvation in this algorithm, every request is serviced.

Disadvantages

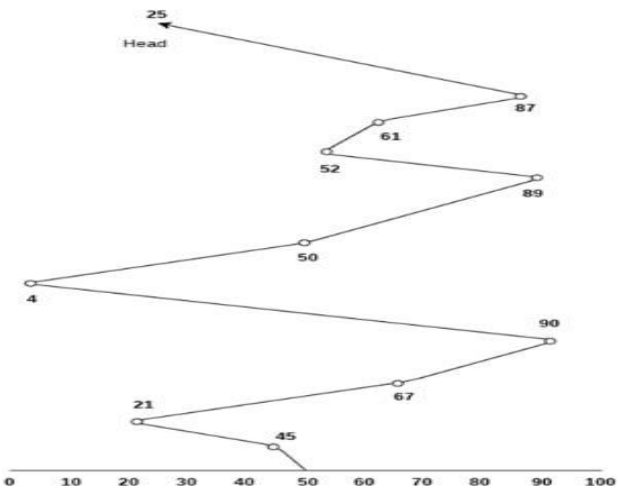
- o The scheme does not optimize the seek time.
- o The request may come from different processes therefore there is the possibility of inappropriate movement of the head.

Example

Consider the following disk request sequence for a disk with 100 tracks 45, 21, 67, 90, 4, 50, 89, 52, 61, 87, 25

Head pointer starting at 50 and moving in left direction. Find the number of head movements in cylinders using FCFS scheduling.

Solution



Number of cylinders moved by the head

= (50-45)+(45-21)+(67-21)+(90-67)+(90-4)+(50-4)+(89-50)+(61-52)+(87-61)+(87-25)

= 5 + 24 + 46 + 23 + 86 + 46 + 49 + 9 + 26 + 62

= 376

SSTF Scheduling Algorithm

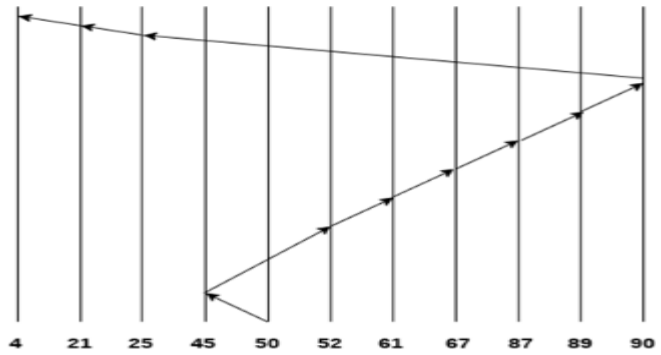
Shortest seek time first (SSTF) algorithm selects the disk I/O request which requires the least disk arm movement from its current position regardless of the direction. It reduces the total seek time as compared to FCFS. It allows the head to move to the closest track in the service queue.

Disadvantages

- o It may cause starvation for some requests.
- o Switching direction on the frequent basis slows the working of algorithm.
- o It is not the most optimal algorithm.

Example

Consider the following disk request sequence for a disk with 100 tracks 45, 21, 67, 90, 4, 89, 52, 61, 87, 25
Head pointer starting at 50. Find the number of head movements in cylinders using SSTF scheduling.



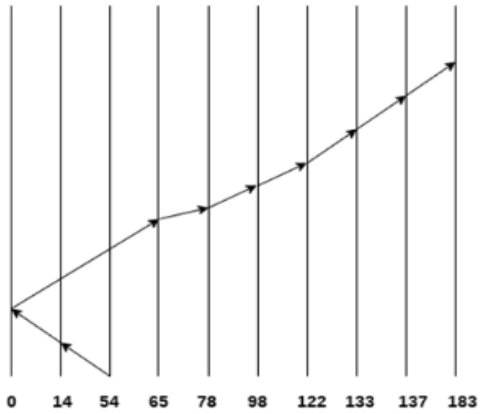
Number of cylinders = 5 + 7 + 9 + 6 + 20 + 2 + 1 + 65 + 4 + 17 = 136

Scan Algorithm

- ✓ It is also called as Elevator Algorithm. In this algorithm, the disk arm moves into a particular direction till the end, satisfying all the requests coming in its path, and then it turns back and moves in the reverse direction satisfying requests coming in its path.
- ✓ It works in the way an elevator works, elevator moves in a direction completely till the last floor of that direction and then turns back.

Example

Consider the following disk request sequence for a disk with 100 tracks 98, 137, 122, 183, 14, 133, 65, 78
Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using SCAN scheduling.



Number of Cylinders = 40 + 14 + 65 + 13 + 20 + 24 + 11 + 4 + 46 = 237

C-SCAN algorithm

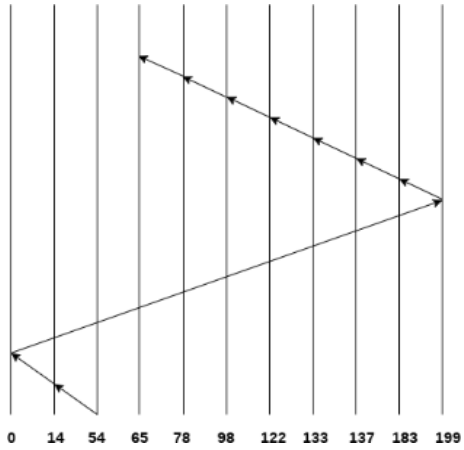
In C-SCAN algorithm, the arm of the disk moves in a particular direction servicing requests until it reaches the last cylinder, then it jumps to the last cylinder of the opposite direction without servicing any request then it turns back and start moving in that direction servicing the remaining requests.

Example

Consider the following disk request sequence for a disk with 100 tracks

98, 137, 122, 183, 14, 133, 65, 78

Head pointer starting at 54 and moving in left direction. Find the number of head movements in cylinders using C-SCAN scheduling.



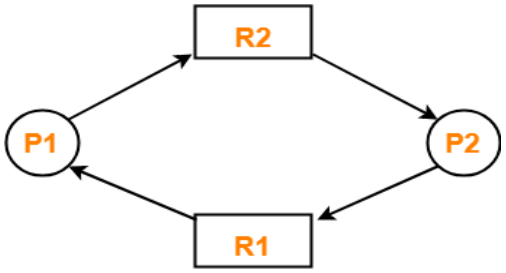
No. of cylinders crossed = 40 + 14 + 199 + 16 + 46 + 4 + 11 + 24 + 20 + 13 = 387

UNIT 5

What is a Deadlock?

In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, awaiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock.

Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process.



Example of a deadlock

System Model

System consists of a finite number of resources to be distributed among a number of competing processes. The resources may be partitioned into several types (or classes), each consisting of some number of identical instances. CPU cycles, files, and I/O devices (such as printers and DVD drives) are examples of resource types. If a system has two CPUs, then the resource type CPU has two instances. Similarly, the resource type printer may have five instances.

A process must request a resource before using it and must release there source after using it. A process may request as many resources as it requires to carry out its designated task. Obviously, the number of resources requested may not exceed the total number of resources available in the system. In other words, a process cannot request three printers if the system has only two. Under the normal mode of operation, a process may utilize a resource in only the following sequence:

1. Request. The process requests the resource. If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire there source.

2. Use. The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).

3. Release. The process releases the resource.

The request and release of resources may be system calls like request() and release() device, open() and close() file, and allocate() and free() memory system calls.

Deadlocks may also involve different resource types. For example, consider a system with one printer and one DVD drive. Suppose that process Pi is holding the DVD and process Pj is holding the printer. If Pi requests the printer and Pj requests the DVD drive, a deadlock occurs.

Deadlock Characterization

In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs from starting.

Necessary Conditions

UNIT 5

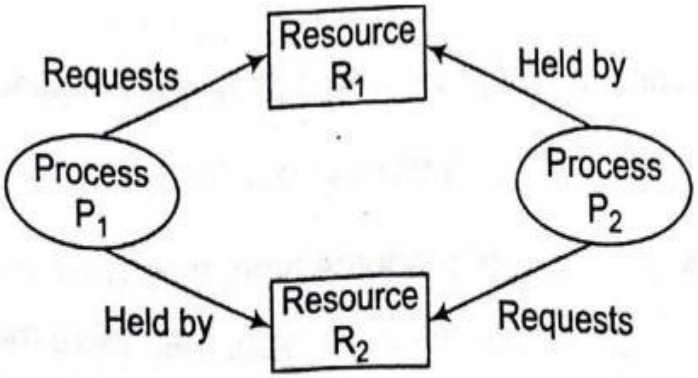
A deadlock situation can arise if the following four conditions hold simultaneously in a system:

1. Mutual exclusion. At least one resource must be held in a non-sharable mode; that is, only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.

2. Hold and wait. A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

3. No preemption. Resources cannot be preempted; that is, a resource can be released only voluntarily by the process holding it, after that process has completed its task.

4. Circular wait. A set {P0, P1, ..., Pn} of waiting processes must exist such that P0 is waiting for a resource held by P1, P1 is waiting for a resource held by P2, ..., Pn−1 is waiting for a resource held by Pn, and Pn is waiting for a resource held by P0.



State diagram of circular wait condition

We emphasize that all four conditions must hold for a deadlock to occur. The circular-wait condition implies the hold-and-wait condition, so the four conditions are not completely independent.

Resource-Allocation Graph

Deadlocks can be described more precisely in terms of a directed graph called a system resource- allocation graph. This graph consists of a set of vertices V and a set of edges E. The set of vertices V is partitioned into two different types of nodes:

P = {P1, P2, ..., Pn}, the set consisting of all the active processes in the system, and R = {R1, R2, ..., Rm}, the set consisting of all resource types in the system.

A directed edge from process Pi to resource type Rj is denoted by Pi → Rj ;it signifies that process Pi has requested an instance of resource type Rj and is currently waiting for that resource.

A directed edge from resource type Rj to process Pi is denoted by Rj → Pi ; it signifies that an instance of resource type Rj has been allocated to process Pi .

A directed edge Pi → Rj is called a request edge; a directed edge Rj → Pi is called an assignment edge.

Pictorially, we represent each process Pi as a circle and each resource type Rj as a rectangle. Since resource type Rj may have more than one instance, were present each such instance as a dot within the rectangle. Note that a request edge points to only the rectangle Rj , whereas an assignment edge must also designate one of the dots in the rectangle. When process Pi requests an instance of resource type Rj, a request edge is inserted in the resourceallocation graph. When this request can be fulfilled, the request edge is instantaneously transformed to an assignment edge. When the process no longer needs access to the resource, it releases the resource. As a result, the assignment edge is deleted.

The resource-allocation graph shown in Figure 7.1 depicts the following situation.

The sets P, R, and E:

- P = {P1, P2, P3}

R = {R1, R2, R3, R4}

◦E = {P1 → R1, P2 → R3, R1 → P2, R2 → P2, R2 → P1, R3 → P3}

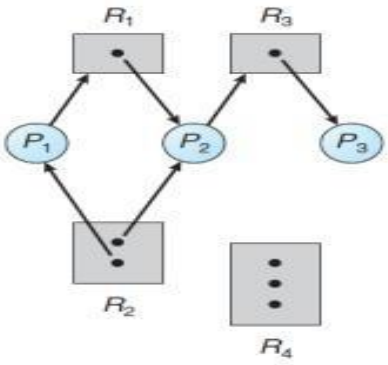


Figure 7.1 Resource-allocation graph.

• Resource instances:

◦One instance of resource type R1
◦Two instances of resource type R2
◦One instance of resource type R3
◦Three instances of resource type R4

• Process states:

◦Process P1 is holding an instance of resource type R2 and is waiting for an instance of resource type P1
◦Process P2 is holding an instance of R1 and an instance of R2 and is waiting for an instance of R3
◦Process P3 is holding an instance of R3.

If the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle, then a deadlock may exist. Suppose that process P3 requests an instance of resource type R2. Since no resource instance is currently available, a request edge P3 ->R2 is added to the graph. 2 cycles:
P1 → R1 → P2 → R3 → P3 → R2 → P1
P2 → R3 → P3 → R2 → P2

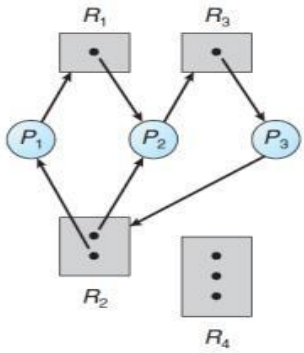


Figure 7.2 Resource-allocation graph with a deadlock.

Processes P1, P2, and P3 are deadlocked. Process P2 is waiting for the resourceR3, which is held by process P3. Process P3 is waiting for either process P1 or process P2 to release resource R2. In addition, process P1 is waiting for processP2 to release resource R1.

UNIT 5

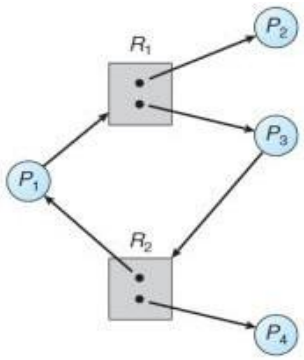


Figure 7.3 Resource-allocation graph with a cycle but no deadlock.

we also have a cycle:
P1 → R1 → P3 → R2 → P1
However, there is no deadlock. Observe that process P4 may release its instance of resource type R2. That resource can then be allocated to P3, breaking the cycle.

Methods for Handling Deadlocks

Generally speaking, we can deal with the deadlock problem in one of three ways:
• We can use a protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlocked state.
• We can allow the system to enter a deadlocked state, detect it, and recover.
• We can ignore the problem altogether and pretend that deadlocks never occur in the system.

The third solution is the one used by most operating systems, including Linux and Windows. It is then up to the application developer to write programs that handle deadlocks.

To ensure that deadlocks never occur, the system can use either a deadlock prevention or a deadlock-avoidance scheme. Deadlock prevention provides a set of methods to ensure that at least one of the necessary conditions cannot hold. These methods prevent deadlocks by constraining how requests for resources can be made.

Deadlock avoidance requires that the operating system be given additional information in advance concerning which resources a process will request and use during its lifetime. With this additional knowledge, the operating system can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process.

If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may arise. In this environment, the system can provide an algorithm that examines the state of the system to determine whether a deadlock has occurred and an algorithm to recover from the deadlock.

DEADLOCK PREVENTION

For a deadlock to occur, each of the 4 necessary conditions must held. By ensuring that at least one of these conditions cannot hold, we can prevent the occurrence of a deadlock. **Mutual Exclusion –** not required for sharable resources; must hold for non sharable resources

Hold and Wait – must guarantee that whenever a process requests a resource, it does not hold any other resources

o Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none

o Low resource utilization; starvation possible

No Preemption –

o If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released

o Preempted resources are added to the list of resources for which the process is waiting

o Process will be restarted only when it can regain its old resources, as well as the new ones that it isrequesting

Circular Wait – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration

Deadlock Avoidance

Requires that the system has some additional a priori information available

- Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need
- The deadlock-avoidance algorithm dynamically examines the resource allocation state to ensure that there can never be a circular-wait condition
- Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes .

Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state System is in safe state if there exists a sequence of ALL the processes in the systems such that for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by all the Pj, with j j <I

That is:

o If Pi resource needs are not immediately available, then Pi can wait until all Pj have finished

o When Pj is finished, Pi can obtain needed resources, execute, return allocated resources, and terminate

o When Pi terminates, Pi +1 can obtain its needed resources, and so on If a system is in safe state no deadlocks

If a system is in unsafe state possibility of deadlock

Avoidance ensure that a system will never enter an unsafe state

Avoidance algorithms

Single instance of a resource type

o Use a resource-allocation graph Multiple instances of a resource type

o Use the banker's algorithm

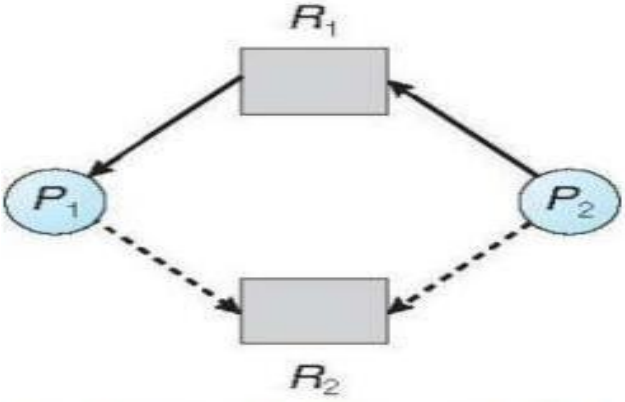
Resource-Allocation Graph Scheme

Claim edgePiÆRj indicated that process Pj may request resource Rj;

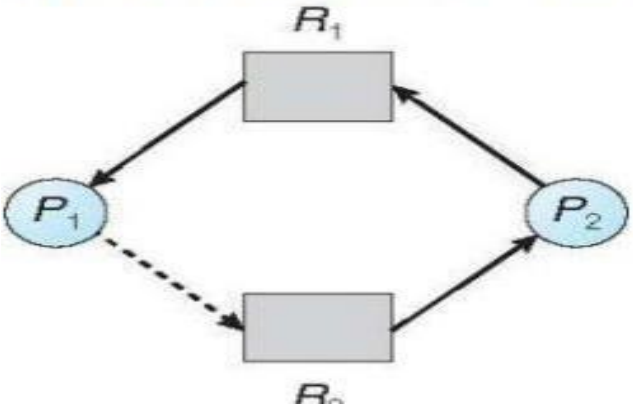
represented by a dashed line

Claim edge converts to request edge when a process requests a resource. Request edge converted to an assignment edge when the resource is allocated to the process When a resource is released by a process, assignment edge reconverts to a claim edge Resources must be claimed a priori in the system

UNIT 5



Unsafe State In Resource-Allocation Graph



Banker's Algorithm

Banker's algorithm is a deadlock avoidance algorithm. It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not. Consider there are n account holders in a bank and the sum of the money in all of their accounts is S. Every time a loan has to be granted by the bank, it subtracts the loan amount from the total money the bank has. Then it checks if that difference is greater than S. It is done because, only then, the bank would have enough money even if all the n account holders draw all their money at once.

Banker's algorithm works in a similar way in computers. Whenever a new process is created, it must specify the maximum instances of each resource type that it needs, exactly.

Let us assume that there are n processes and m resource types. Some data structures that are used to implement the banker's algorithm are:

1. **Available:** Vector of length m. If available [j] = k, there are k instances of resource type Rj available
2. **Max:** n x m matrix. If Max [i,j] = k, then process Pimay request at most k instances of resource type Rj
3. **Allocation:** n x m matrix. If Allocation[i,j] = k then Pi is currently allocated k instances of Rj
4. **Need:** n x m matrix. If Need[i,j] = k, then Pi may need k more instances of Rj to complete its task .Need [i,j] = Max[i,j] – Allocation [i,j]

Safety Algorithm

1. Let Work and Finish be vectors of length m and n, respectively. Initially,
2. Work = Available
3. Finish[i] =false for i = 0, 1, ... , n - 1.

This means, initially, no process has finished and the number of available resources is represented by the Available array.

4. Find an index i such that both
5. Finish[i] ==false
6. Needi <= Work

If there is no such i present, then proceed to step 4.
It means, we need to find an unfinished process whose need can be satisfied by the available resources. If no such process exists, just go to step 4.
7. Perform the following:
8. Work = Work + Allocation;
9. Finish[i] = true;
Go to step 2.
When an unfinished process is found, then the resources are allocated and the process is marked finished. And then, the loop is repeated to check the same for all other processes.
10. If Finish[i] == true for all i, then the system is in a safe state.
That means if all processes are finished, then the system is in safe state.
Example:
Considering a system with five processes P0 through P4 and three resources types A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances.
Suppose at time t0 following snapshot of the system has been taken:

Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Σ The content of the matrix Need is defined to be Max
– Allocation Need
A B C
The system is in a safe state since the sequence <P1, P3, P4, P2, P0>satisfies safety criteria
P1 Request (1,0,2)
Check that Request £ Available (that is, (1,0,2) £ (3,3,2) true

Process	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

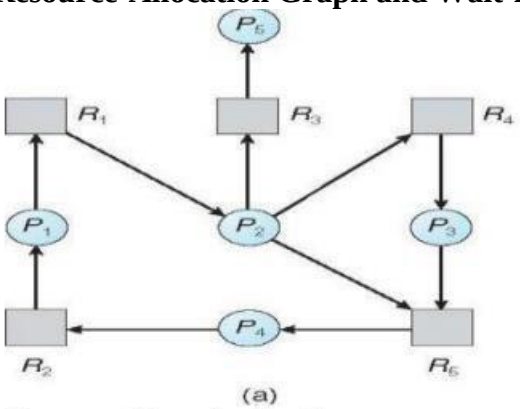
Deadlock Detection

If deadlock prevention and avoidance are not done properly, as deadlock may occur and only things left to do is to detect the recover from the deadlock.

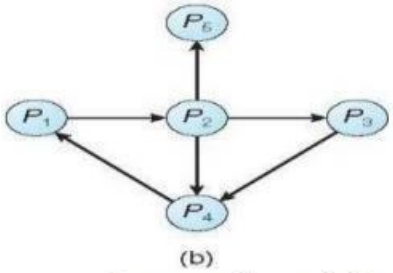
If all resource types has only single instance, then we can use a graph called wait-forgraph, which is a variant of resource allocation graph. Here, vertices represent processes and a directed edge from P1 to P2 indicate that P1 is waiting for a resource held by P2. Like in the case of resource allocation graph, a cycle in a wait-for-graph indicate a deadlock. So the system can maintain a wait-for-graph and check for cycles periodically to detect any deadlocks.

UNIT 5

Resource-Allocation Graph and Wait-for Graph



Resource-Allocation Graph



Corresponding wait-for graph

Several Instances of a Resource Type

Available: A vector of length m indicates the number of available resources of each type. Allocation: An n x m matrix defines the number of resources of each type currently allocated to each process.

Request: An n x m matrix indicates the current request of each process.

If Request [i][j] = k, then process Pi is requesting k more instances of resource type.Rj.

Detection Algorithm

Let Work and Finish be vectors of length m and n, respectively Initialize:

(a) Work = Available

(b) For i = 1,2, …, n, if Allocationiπ 0, then

Finish[i] = false; otherwise, Finish[i] = true

2. Find an index isuch that both:

(a) Finish[i] == false

(b) Requesti£Work

If no such i exists, go to step 4

3. Work = Work + Allocationi

Finish[i] = true

go to step 2

4. If Finish[i] == false, for some i, 1 £i£n, then the system is in deadlock state. Moreover, if Finish[i] == false, then Pi is deadlocked

Recovery from Deadlock:

When a detection algorithm determines that a deadlock exists, several alter-natives are available. One possibility is to inform the operator that a deadlock has occurred and to let the operator deal with the deadlock manually. Another possibility is to let the system recover from the deadlock automatically. There are two options for breaking a deadlock. One is simply to abort one or more processes to break the circular wait. The other is to preempt some resources from one or more of the deadlocked processes.

Process Termination

Abort all deadlocked processes

Abort one process at a time until the deadlock cycle is eliminated In which order should we choose to abort?

o Priority of the process

o How long process has computed, and how much longer to completion

o Resources the process has used

o Resources process needs to complete

o How many processes will need to be terminated

o Is process interactive or batch?

Resource Preemption

Selecting a victim – minimize cost

UNIT 5

Rollback – return to some safe state, restart process for that state
Starvation – same process may always be picked as victim, include number of rollback in cost factor of cylinders.