OPERATING SYSTEM

SYLLABUS DBATU

BTCOC402: Operating Systems

[Unit 1] **[7 Hours]**

Introduction and Operating system structures: Definition, Types of Operating system, Real-Time operating system, System Components: System Services, Systems Calls, System Programs, System structure, Virtual Machines, System Design and Implementation, System Generations.

[Unit 2] **[7 Hours]**

Processes and CPU Scheduling: Process Concept, Process Scheduling, Operation on process, Inter-process Communication, Cooperating processes, Threads, Multithreading model, Scheduling criteria, Scheduling Algorithms, Thread Scheduling, Multiple-Processor Scheduling, Scheduling Algorithms evaluation.

[Unit 3] **[7 Hours]**

Process Synchronization: The critical-section problem, Critical regions, Peterson's Solution, Synchronization Hardware, Semaphores, Classical Problems of synchronization, and Monitors Deadlocks: Systems Model, Deadlock characterization, Methods for handling Deadlocks, Deadlock Prevention, Deadlock Avoidance, Deadlock Detection, Recovery from Deadlock, Combined approach to deadlock Handling.

[Unit 4] **[7 Hours]**

Memory Management: Basic concept, Logical and Physical address map, Memory allocation: Continuous Memory Allocation, Fixed and variable partition, Internal and external fragmentation and compaction, Paging: Principle of operation, Page allocation – Hardware support for paging, Protection and sharing, Disadvantages of paging; Segmentation. Virtual Memory: Basics of Virtual Memory – Hardware and control structures – Locality of reference, Page fault, Working Set, Dirty page / Dirty bit – Demand paging, Page Replacement algorithms: Optimal, First in First Out (FIFO), Second Chance (SC), Not recently used (NRU) and Least Recently used(LRU).

[Unit 5] **[7 Hours]**

File Management: File Concept, Access methods, File types, File operation, Directory and disk structure, File System Structure, File System Implementation, Allocation methods (contiguous, linked, indexed), Free-space management (bit vector, linked list, grouping), directory implementation (linear list, hash table), efficiency and performance. Mass-Storage Structure: Disk Structure, Disk attachment, Disk scheduling, Disk management, Swap Space Management.

Text Book:

1. Abraham Silberschatz, Peter B. Galvin and Greg Gagne, Operating System Concepts, Wiley Publication, 8th Edition, 2008.

OPERATING SYSTEM

NOTES

Chapter 1: Introduction and Operating System Structures -

This chapter lays the foundation for understanding Operating Systems. Focus on clear definitions, types, and the fundamental components and services.

---

1. Operating System Services

- **Question Type:** "Explain operating services with respect to programs and users." or "List out different services of Operating Systems and Describe each service."
- **For 6 Marks, focus on:** Clearly listing and briefly explaining 5-6 services for users and programs.

Notes: An Operating System (OS) provides services to users and to programs to ensure efficient operation of the system. These services act as a bridge between the user/applications and the hardware.

- **Services for Users:**
    - **User Interface (UI):** Provides a way for users to interact with the system (e.g., Command-Line Interface (CLI), Graphical User Interface (GUI), Batch Interface).
    - **Program Execution:** The OS loads a program into memory and runs it. It handles both its normal and abnormal execution.
    - **I/O Operations:** Manages input/output for running programs, abstracting complexities of hardware devices (e.g., reading from keyboard, writing to printer).
    - **File-System Manipulation:** Programs need to create, delete, read, write, and manage files and directories. The OS provides functions for these operations.
    - **Communications:** Allows processes to exchange information, either on the same computer (Inter-process Communication - IPC) or across a network.
    - **Error Detection:** The OS constantly monitors the system for hardware errors (e.g., memory errors, power failure) and software errors (e.g., arithmetic overflow) to maintain correct and consistent computing.
- **Services for Efficient System Operation (for the OS's own benefit):**
    - **Resource Allocation:** Manages and distributes system resources (CPU cycles, memory, files, I/O devices) among multiple competing processes.
    - **Accounting:** Keeps track of which users use how much and what kind of computer resources.
    - **Protection and Security:** Ensures that all access to system resources is controlled and protects the system from unauthorized access.

2. Types of Operating Systems

- **Question Type:** "Explain Real time operating system with its types, advantages and examples." or "Explain microkernel type operating system structure." or "Explain Virtual Machine (VM) based structure of operating system."
- **For 6 Marks, focus on:** Choosing one type and explaining it comprehensively, or briefly describing 2-3 types. If asked specifically for one, detail it.

Notes (Example for Real-Time OS):

Real-Time Operating System (RTOS): An RTOS is designed to process data with strict time constraints, usually for critical applications where tasks must be completed within specific deadlines. They are often used in embedded systems.

- **Types:**
    - **Hard Real-Time Systems:** Guarantee that critical tasks complete within a specified deadline. Failure to meet a deadline can lead to catastrophic results (e.g., flight control systems, medical life-support systems). They have minimal jitter and strict timing requirements.
    - **Soft Real-Time Systems:** Prioritize critical tasks over others but do not guarantee completion within a deadline. Missing a deadline is undesirable but not catastrophic (e.g., multimedia systems, online transaction processing).
- **Advantages:**
    - High responsiveness and predictability.
    - Efficient resource utilization for time-critical tasks.
    - Minimized latency.
- **Examples:** QNX, RTLinux, VxWorks.

Notes (Example for Microkernel Structure):

Microkernel Structure: In a microkernel architecture, only the absolutely essential services are placed in the kernel (e.g., inter-process communication, basic memory management, fundamental CPU scheduling). Other OS services (like file systems, device drivers, network protocols) are implemented as user-level processes.

- **Diagram (Conceptual):**
- +-----------------------------------+
- | User Space |
- | +-----------+ +-----------+ |
- | | File | | Device | |
- | | System | | Drivers | |
- | +-----------+ +-----------+ |

- | ...other services |
- +----------------------------------+
- | Kernel Space |
- | +----------------------------+ |
- | | Microkernel | |
- | | (IPC, Basic Scheduling, Memory) | |
- | +----------------------------+ |
- +----------------------------------+
- | Hardware |
- +----------------------------------+

- **Advantages:**
    - **Extensibility:** Easy to add new services or features without modifying the kernel.
    - **Robustness/Reliability:** A bug in a user-level service won't crash the entire kernel. Services run in protected user mode.
    - **Portability:** Easier to port the OS to new hardware architectures.
    - **Security:** Services run in user space, reducing potential for kernel exploits.
- **Disadvantages:**
    - **Performance Overhead:** Increased number of context switches and inter-process communication (IPC) calls for service requests, leading to slower execution compared to monolithic kernels.

---

3. Virtual Machines (VM) Based Structure

- **Question Type:** "Define a virtual machine with neat diagram. Describe the concept and working of JVM. Explain what are the benefits of a VM?"
- **For 6 Marks, focus on:** Definition, a conceptual diagram, key components (VMM/Hypervisor), and benefits.

Notes: A **Virtual Machine (VM)** is a software implementation of a computer that executes programs like a physical machine. It provides an environment that is identical to the underlying hardware.

- **Concept:** A **Virtual Machine Monitor (VMM)**, also known as a **hypervisor**, is the software layer that creates and manages virtual machines. It presents a virtual hardware interface to each guest OS, making each VM believe it has direct access to the hardware. The VMM intercepts and translates privileged instructions and I/O requests from the guest OS to the actual hardware.
- **Diagram (Conceptual):**

- +----------------------------------+
- | Applications |
- +----------------------------------+
- | Guest OS 1 (e.g., Linux) |
- +----------------------------------+
- | Applications |
- +----------------------------------+
- | Guest OS 2 (e.g., Windows) |
- +----------------------------------+
- | Virtual Machine Monitor |
- | (Hypervisor) |
- +----------------------------------+
- | Hardware |
- | (CPU, Memory, Disk, Network) |
- +----------------------------------+
- **Benefits:**
    - **Resource Isolation:** Each VM is isolated from others, preventing crashes in one from affecting others.
    - **Portability:** A VM image can be moved and run on different physical machines, provided they have a compatible VMM.
    - **Development & Testing:** Provides isolated environments for developing and testing software on different operating systems or configurations without needing multiple physical machines.
    - **Consolidation:** Multiple virtual servers can run on a single physical server, reducing hardware costs and power consumption.
    - **Security:** Provides a sandboxed environment for running potentially malicious software.

---

4. Command Interpreter & Kernel

- **Question Type:** "What is the purpose of command interpreter? Why is it usually separate from the Kernel?"
- **For 6 Marks, focus on:** Defining command interpreter, its role, and the reasons for its separation from the kernel.

Notes:

- **Command Interpreter (Shell):**
    - **Purpose:** It's the primary way a user interacts with the operating system. It reads commands entered by the user (or from a script) and executes them. It does this by fetching the command, parsing it, and then often invoking a system call to execute the requested program or operation.
    - **Examples:** bash, cmd.exe, PowerShell.
- **Kernel:**
    - **Purpose:** The core of the operating system. It manages the system's most fundamental resources (CPU, memory, I/O devices). It provides essential services like process scheduling, memory management, and inter-process communication. The kernel runs in a privileged mode and has direct access to hardware.
- **Why Separation from Kernel?**

1. **Flexibility and Customization:** By keeping the command interpreter separate, users can choose different shells or even write their own without modifying the core OS. This allows for diverse user experiences and scripting capabilities.

2. **Modularity and Maintenance:** Separating user-facing components from the kernel makes the OS more modular. Changes or updates to the command interpreter do not require changes or reboots of the kernel, simplifying maintenance and development.

3. **Stability and Security:** The kernel handles critical system operations. If the command interpreter were part of the kernel, a bug or crash in the shell could potentially destabilize the entire system. Keeping it separate enhances system stability and security by isolating potential failures.

4. **Resource Efficiency:** Some systems might not need a command interpreter (e.g., embedded systems). Separating it allows for a smaller, more efficient kernel in such scenarios.

---

5. Major Activities of an Operating System (Overview)

- **Question Type:** "Describe major activities of an operating system in regard to: 1) Process management 2) File management 3) Main Memory management 4) Secondary storage management."
- **For 6 Marks, focus on:** Briefly describing the OS's role in each of these four core areas.

Notes: The operating system performs several crucial activities to ensure the efficient and correct operation of the computer system. These can be broadly categorized into:

1. **Process Management:**
    - **Activities:** Creating and deleting user/system processes, suspending and resuming processes, providing mechanisms for process synchronization and communication, handling deadlocks.
    - **Goal:** Efficiently manage CPU usage and allow multiple programs to run concurrently.
2. **File Management:**

- **Activities:** Creating and deleting files/directories, primitive operations on files (read, write, reposition), mapping files onto secondary storage, backing up files onto stable storage.
- **Goal:** Provide a uniform logical view of information storage and retrieval.

3. **Main Memory Management:**
   - **Activities:** Keeping track of which parts of memory are being used by whom, deciding which processes/data to move into/out of memory, allocating and de-allocating memory space.
   - **Goal:** Maximize CPU utilization and allow multiple programs to reside in memory simultaneously.
4. **Secondary Storage Management:**
   - **Activities:** Free-space management, storage allocation, disk scheduling. Manages the hard disks, optical drives, etc.
   - **Goal:** Efficiently store data that does not fit in main memory and provide for long-term storage.

Chapter 2: Processes and CPU Scheduling - Efficient Notes (for 6 Marks)

This chapter is fundamental to understanding how the OS manages running programs and allocates CPU time. Pay special attention to the numerical problems for CPU scheduling, as they are highly repetitive.

---

1. Process Concept and Process Control Block (PCB)

- **Question Type:** "What is a process? Describe the contents of Process Control Block (PCB)." or "Describe Process Control Block with suitable Example."
- **For 6 Marks, focus on:** Defining a process, its states, and detailing the key information stored in a PCB.

Notes:

A **process** is a program in execution. It is an active entity, unlike a program, which is a passive entity (a file stored on disk). A process needs resources (CPU, memory, I/O, files) to accomplish its task.

- **Process States:** A process typically goes through various states during its lifetime:
    - **New:** The process is being created.
    - **Ready:** The process is waiting to be assigned to a processor.
    - **Running:** Instructions are being executed by the CPU.
    - **Waiting (Blocked):** The process is waiting for some event to occur (e.g., I/O completion, receiving a signal).
    - **Terminated:** The process has finished execution.
- **Process Control Block (PCB):**
    - A PCB is a data structure maintained by the operating system for each process. It contains all the information needed to manage and control a process. It acts as a repository for the process's current state.
    - **Contents of a PCB:**
        1. **Process State:** Current state of the process (New, Ready, Running, Waiting, Terminated).
        2. **Program Counter (PC):** The address of the next instruction to be executed for this process.
        3. **CPU Registers:** Contents of all CPU registers that might be needed to restart the process (e.g., accumulators, index registers, stack pointers, general-purpose registers).
        4. **CPU Scheduling Information:** Process priority, pointers to scheduling queues (ready queue, wait queues), and other scheduling parameters.

5. **Memory-Management Information:** Value of base and limit registers, page tables, segment tables, etc., depending on the memory system used by the OS.
6. **Accounting Information:** CPU time used, real time elapsed, time limits, account numbers, process numbers, etc.
7. **I/O Status Information:** List of I/O devices allocated to the process, list of open files, etc.

---

2. Threads: User-Level vs. Kernel-Level

- **Question Type:** "What is thread? Differentiates between user level thread and kernel level thread." or "Describe the actions taken by a kernel to context switch between kernel level threads."
- **For 6 Marks, focus on:** Defining a thread and a clear comparison table/points between ULTs and KLTs, or detailing kernel-level context switching.

Notes:

A **thread** (or lightweight process) is the basic unit of CPU utilization. It comprises a thread ID, a program counter, a register set, and a stack. Threads within the same process share code section, data section, and other OS resources (e.g., open files, signals).

Comparison: User-Level Threads (ULTs) vs. Kernel-Level Threads (KLTs)

Feature	User-Level Threads (ULTs)	Kernel-Level Threads (KLTs)
Management	Managed by user-level thread library.	Managed by the operating system kernel.
Kernel Involvement	Kernel is unaware of threads.	Kernel is aware of and directly manages threads.
Context Switching	Faster; no kernel mode switch needed.	Slower; requires kernel mode switch.
Blocking Call	If one thread makes a blocking system call, the *entire process* blocks.	If one thread makes a blocking call, the kernel can schedule *another thread* from the same process.
Multiprocessing	Cannot take advantage of multiple processors (as kernel sees only one process).	Can run on multiple processors simultaneously.
Implementation	Easier to implement; managed by a library.	Harder to implement; integrated into the OS kernel.
Examples (Libraries)	POSIX Pthreads, Java Threads.	Windows threads, Solaris threads, Linux threads.

Export to Sheets

Context Switching (for Kernel-Level Threads): When the kernel switches the CPU from one kernel-level thread to another (possibly within the same process or different processes), it performs the following actions:

1. **Save State:** The CPU saves the context (CPU register values, program counter, stack pointer) of the currently running thread.
2. **Load State:** The CPU loads the saved context of the next thread to be run.
3. **Update PCB/TCB:** The process's (or thread's) state in its PCB (or Thread Control Block - TCB) is updated from "running" to "ready" or "waiting," and the new thread's state is set to "running."
4. **Memory Management Information:** If switching to a thread of a different process, the memory-management information (e.g., page tables) must also be changed.

---

3. Inter-process Communication (IPC)

- **Question Type:** "What are co-operating processes? Describe the mechanism of inter process communication using shared memory and massage passing." or "What is inter-process communication in operating System? Explain its types."
- **For 6 Marks, focus on:** Defining cooperating processes, and explaining Shared Memory and Message Passing mechanisms, including their pros and cons.

Notes:

- **Cooperating Processes:** Processes that can affect or be affected by other processes executing in the system. They need mechanisms to communicate and synchronize their actions, often because they share data or work towards a common goal.
- **Inter-process Communication (IPC):** Mechanisms provided by the OS that allow independent processes to communicate and synchronize their actions without sharing the same address space.
- **Mechanisms of IPC:**
    1. **Shared Memory:**
        - **Mechanism:** A region of memory is established that is shared by cooperating processes. Processes can then read from and write to this shared region directly. The OS typically only helps in establishing the shared region; once set up, communication happens without further OS intervention.
        - **Advantages:**
            - **Speed:** Very fast once established, as processes access memory directly, avoiding kernel overhead for each communication.
        - **Disadvantages:**

        - **Synchronization:** Processes must implement their own synchronization mechanisms (e.g., semaphores) to ensure data consistency in the shared region.
        - **Setup:** Initial setup (creating the shared memory region) requires kernel involvement.
    - **Diagram (Conceptual):**
    - +------------+   +------------+
    - | Process A | | Process B |
    - | | | |
    - | reads/ | | reads/ |
    - | writes | | writes |
    - +-----|------+   +------|-----+
    - | |
    - v   v
    - +-------------------------------+
    - | Shared Memory |
    - | (Data exchange) |
    - +-------------------------------+

2. **Message Passing:**
    - **Mechanism:** Processes communicate by exchanging messages. The OS provides send() and receive() primitives. Processes don't share memory but exchange data through these primitives, which involve the kernel.
    - **Types:**
        - **Direct Communication:** Processes explicitly name each other for sending/receiving (send(P, message), receive(Q, message)).
        - **Indirect Communication:** Messages are sent to and received from mailboxes (ports) (send(A, message), receive(A, message)).
    - **Advantages:**
        - **Simplicity:** Easier to implement for small amounts of data.
        - **No Shared Memory:** Useful in distributed systems where processes are on different machines.
        - **Synchronization:** OS often handles some level of synchronization.
    - **Disadvantages:**

        - **Speed:** Slower than shared memory due to repeated system calls (kernel involvement) for every message transfer.
        - **Overhead:** Overhead for message formatting and copying.
    - **Diagram (Conceptual):**
    - +------------+   +------------+
    - | Process A | | Process B |
    - | | | |
    - | send(msg) | <-> | receive(msg)|
    - +-----|------+   +------|-----+
    - | |
    - v   v
    - +--------------------------------+
    - | Kernel |
    - | (Message Queues/Mailboxes) |
    - +--------------------------------+

---

4. CPU Scheduling Algorithms (Introduction and Criteria)

- **Question Type:** "What are the criteria for CPU scheduling? Explain different types of schedulers."
- **For 6 Marks, focus on:** Listing and briefly explaining scheduling criteria and the three main types of schedulers. *(Note: Actual algorithm calculations are a separate, higher-priority, often 6-mark or more question type).*

Notes:

- **CPU Scheduling:** The process of deciding which process gets the CPU next from the ready queue. The goal is to maximize CPU utilization and provide a good user experience.
- **Scheduling Criteria (Performance Metrics):**
    1. **CPU Utilization:** Keep the CPU as busy as possible (range from 0 to 100%).
    2. **Throughput:** Number of processes completed per unit of time.
    3. **Turnaround Time:** Total time taken from submission of a process until its completion (includes waiting, ready, execution, I/O time).
    4. **Waiting Time:** Total time a process spends waiting in the ready queue.
    5. **Response Time:** Time from submission of a request until the first response is produced (for interactive systems).

6. **Fairness:** Ensure each process gets a fair share of the CPU.

- **Types of Schedulers:**
    1. **Long-Term Scheduler (Job Scheduler):**
        - **Role:** Selects processes from the job pool (disk) and loads them into main memory (ready queue) for execution.
        - **Frequency:** Runs infrequently.
        - **Impact:** Controls the **degree of multiprogramming** (number of processes in memory).
    2. **Short-Term Scheduler (CPU Scheduler / Dispatcher):**
        - **Role:** Selects one of the processes that are in the ready queue and allocates the CPU to it.
        - **Frequency:** Runs very frequently (milliseconds).
        - **Impact:** Directly affects CPU utilization and process turnaround time.
    3. **Mid-Term Scheduler:**
        - **Role:** Involved in **swapping**. It removes processes from main memory (swaps them out) to reduce the degree of multiprogramming or to free up memory, and can later bring them back (swap them in).
        - **Frequency:** Runs less frequently than short-term scheduler, more frequently than long-term.
        - **Impact:** Used for memory management and managing the degree of multiprogramming dynamically.

Chapter 3: Process Synchronization - Efficient Notes (for 6 Marks)

This chapter focuses on managing concurrent processes to prevent inconsistencies and deal with deadlocks. Critical section problems, semaphores, and especially the Banker's Algorithm are frequently tested.

---

1. Critical Section Problem and its Requirements

- **Question Type:** "What is critical section problem and what are the requirements that need to be satisfied by any solution to critical section problem?" or "Describe the three requirements to satisfy as a solution to critical-section problem."
- **For 6 Marks, focus on:** Defining the critical section and detailing the three essential requirements.

Notes:

- **Critical Section Problem:**
    - In a multi-programming environment, multiple processes may concurrently access and update shared data. This concurrent access to shared data can lead to data inconsistency.
    - The **critical section** is a segment of code where a process accesses shared resources (like shared variables, files, or hardware devices). It's crucial that only one process executes its critical section at any given time to maintain data consistency.
    - The Critical Section Problem is to design a protocol that the processes can use to cooperate such that their data remains consistent.
- **Requirements for a Solution to the Critical Section Problem:**

1. **Mutual Exclusion:**
    - **Definition:** If one process is executing in its critical section, then no other process is allowed to execute in its critical section.
    - **Importance:** This is the fundamental requirement to prevent data corruption due to concurrent access.
2. **Progress:**
    - **Definition:** If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder sections can participate in deciding which will enter its critical section next. This selection cannot be postponed indefinitely.
    - **Importance:** Ensures that if the critical section is free, a process waiting to enter it can eventually do so, preventing indefinite blocking.
3. **Bounded Waiting:**

- **Definition:** There must be a limit on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.
- **Importance:** Prevents starvation, where a process might wait indefinitely to enter its critical section while other processes repeatedly enter theirs.

---

2. Peterson's Solution for Critical Section

- **Question Type:** "Illustrate Peterson's Solution for critical section problem." or "Discuss the Peterson's solution for the critical-section problem."
- **For 6 Marks, focus on:** Explaining the solution for two processes using flag and turn variables, and showing how it satisfies the requirements. Pseudocode is beneficial.

Notes:

Peterson's Solution: A classic software-based solution to the critical-section problem for **two processes (P0 and P1)**. It uses two shared variables:

- boolean flag[2]: flag[i] = true indicates that process Pi wants to enter its critical section.
- int turn: Indicates whose turn it is to enter the critical section. If turn == i, then process Pi is allowed to enter.
- **Algorithm for Process Pi:**
- do {
- flag[i] = true; // Pi expresses interest
- turn = j; // Gives turn to Pj
- while (flag[j] && turn == j); // Busy wait while Pj wants & it's Pj's turn
-
- // CRITICAL SECTION
- // ... access shared resources ...
- // END CRITICAL SECTION
-
- flag[i] = false; // Pi exits critical section
- // REMAINDER SECTION
- } while (true);
- **How it Satisfies Requirements:**
    1. **Mutual Exclusion:** If both P0 and P1 try to enter simultaneously, one will set turn to 0 and the other to 1. The last one to execute turn = j will have its turn value

overwritten. The while loop condition (flag[j] && turn == j) ensures that only one process can proceed if both are interested.

2. **Progress:** If P0 is in its remainder section (flag[0]=false) and P1 wants to enter, P1 can immediately enter as flag[0] is false, satisfying the condition. If P0 was waiting and P1 leaves, P0 can enter.
3. **Bounded Waiting:** If P0 sets flag[0] = true and turn = 1, and P1 is repeatedly entering and leaving, P0 will eventually get its turn. When P1 exits, it sets flag[1] = false. This will allow P0 to enter the critical section, ensuring that P0 doesn't wait indefinitely.

---

3. Deadlock Characterization

- **Question Type:** "Describe necessary conditions for a deadlock situation to arise." or "Explain the use of Resource Allocation Graph (RAG) in deadlock detection."
- **For 6 Marks, focus on:** Defining deadlock and explaining the four necessary conditions. Optionally, touch upon RAG.

Notes:

- **Deadlock:** A situation in a multi-programming system where two or more processes are indefinitely blocked, waiting for each other to release resources that they need.
- **Necessary Conditions for Deadlock:** For a deadlock to occur, all four of these conditions must hold simultaneously:
    1. **Mutual Exclusion:**
        - **Definition:** At least one resource must be held in a non-sharable mode; 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.
        - **Example:** A printer cannot be simultaneously used by multiple processes.
    2. **Hold and Wait:**
        - **Definition:** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
        - **Example:** Process P1 holds resource R1 and waits for R2, which is held by P2.
    3. **No Preemption:**
        - **Definition:** Resources cannot be forcibly taken away from a process once they have been allocated to it. A resource can only be released voluntarily by the process holding it after it has completed its task.
        - **Example:** A CPU scheduler cannot forcibly take away a memory block from a process.
    4. **Circular Wait:**

        - **Definition:** A set of processes {P0,P1,...,Pn} 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.
        - **Example:** P1 waits for R2 (held by P2), P2 waits for R1 (held by P1). This forms a cycle.

- **Resource Allocation Graph (RAG):**
    - A directed graph used to depict the current state of resource allocation and requests in a system.
    - **Nodes:** Processes (circles) and Resources (squares).
    - **Edges:**
        - **Request Edge (P -> R):** Process P requests resource R.
        - **Assignment Edge (R -> P):** Resource R is assigned to process P.
    - **Deadlock Detection:** If a RAG contains a cycle, a deadlock *may* exist. If each resource type has only one instance, then a cycle *implies* a deadlock. If resource types have multiple instances, a cycle indicates a *possibility* of deadlock.

---

4. Banker's Algorithm (Conceptual & Overview)

- **Question Type:** (Often a numerical problem, but conceptual explanation can be asked for 6 marks). "Examine banker's algorithm..." or "Explain the concept of Banker's Algorithm."
- **For 6 Marks, focus on:** Its purpose (deadlock avoidance), key data structures (Allocation, Max, Available, Need), and the high-level idea of the safety algorithm. *Do not attempt a full numerical solution here unless specifically asked for numerical problem.*

Notes:

- **Banker's Algorithm:**
    - A **deadlock avoidance algorithm** that ensures the system is always in a **safe state** to prevent deadlocks. It gets its name from the banking system, where a bank ensures it doesn't lend money in a way that it can never satisfy the needs of all its customers.
    - It requires processes to declare their **maximum possible resource needs** beforehand.
- **Key Data Structures:**

1. **Available (Vector):** Indicates the number of available instances of each resource type.
2. **Max (Matrix):** Defines the maximum demand of each process for each resource type.
3. **Allocation (Matrix):** Indicates the number of resources of each type currently allocated to each process.
4. **Need (Matrix):** Indicates the remaining resources each process still needs (Need[i][j] = Max[i][j] - Allocation[i][j]).

- **Safe State and Safe Sequence:**
    - **Safe State:** A state where the system can allocate resources to each process (up to its maximum available) in some order and still avoid a deadlock.
    - **Safe Sequence:** The sequence of processes <P1, P2, ..., Pn> such that for each Pi, the resources that Pi still needs can be satisfied by the currently Available resources plus any resources held by all Pj where j < i. If a safe sequence exists, the system is in a safe state.
- **Algorithm's Logic (High-Level):**

1. When a process requests resources, the OS first checks if granting the request would leave the system in a safe state.
2. If it remains safe, the request is granted.
3. If granting the request would lead to an unsafe state, the request is denied, and the process waits.

- **Advantage:** Prevents deadlocks.
- **Disadvantage:** Requires prior knowledge of maximum resource needs, processes may be delayed.

Chapter 4: Memory Management - Efficient Notes (for 6 Marks)

This chapter is critical for understanding how the OS manages the computer's memory. Paging, Virtual Memory, and Page Replacement Algorithms are core concepts, with numerical problems on page faults being highly common.

---

1. Logical vs. Physical Address & Memory Allocation Basics

- **Question Type:** "Consider a logical address space of X pages of Y words each, mapped on to a physical memory of Z frames. How many bits are there in the logical address? How many bits are there in the physical address?" or "Explain contiguous memory allocation. Describe internal and external fragmentation."
- **For 6 Marks, focus on:** Defining logical/physical addresses, bit calculations, and explaining fragmentation.

Notes:

- **Logical Address vs. Physical Address:**
    - **Logical Address (Virtual Address):** An address generated by the CPU. It refers to a location in the logical (or virtual) address space, which is what the program "sees."
    - **Physical Address:** An address seen by the memory unit (Memory Management Unit - MMU) and actually placed on the memory bus. It refers to a real location in the physical memory (RAM).
    - **Mapping:** The MMU translates logical addresses into physical addresses.
- **Bit Calculation Example:**
    - **Scenario:** Logical address space of 64 pages (2^6 pages) of 1024 words each (2^10 words/page), mapped onto a physical memory of 32 frames (2^5 frames).
    - **Logical Address Bits:**
        - Bits for Page Number: log2(Number of Pages) = log2(64) = 6 bits.
        - Bits for Offset (within a page): log2(Page Size) = log2(1024) = 10 bits.
        - **Total Logical Address Bits:** Page Bits + Offset Bits = 6 + 10 = **16 bits**.
    - **Physical Address Bits:**
        - Bits for Frame Number: log2(Number of Frames) = log2(32) = 5 bits.
        - Bits for Offset (within a frame): log2(Frame Size) = log2(1024) = 10 bits (Page Size = Frame Size).
        - **Total Physical Address Bits:** Frame Bits + Offset Bits = 5 + 10 = **15 bits**.
- **Memory Allocation & Fragmentation:**
    - **Contiguous Memory Allocation:** Each process is allocated a single, contiguous block of memory.

- **Fragmentation:** Unused memory space that cannot be effectively utilized.
    - **Internal Fragmentation:** Occurs when a process is allocated more memory than it actually needs, and the unused portion is *within* the allocated block (e.g., fixed-size partitions, or paging where a page is not fully filled).
    - **External Fragmentation:** Occurs when there is enough total free memory space to satisfy a request, but the available space is scattered in small, non-contiguous chunks (holes), so no single hole is large enough.

---

2. Paging and Page Fault Handling

- **Question Type:** "What is demand paging? Explain the steps in handling page fault using appropriate diagram." or "Describe the action taken by the operating system when a page fault occurs with neat diagram."
- **For 6 Marks, focus on:** Defining demand paging, page fault, and detailing the steps involved with a clear conceptual diagram.

Notes:

- **Paging:** A memory management scheme that allows a process's physical address space to be non-contiguous. Logical memory is divided into fixed-size blocks called **pages**, and physical memory is divided into fixed-size blocks called **frames** (page size = frame size).
- **Demand Paging:** A virtual memory technique where pages are loaded into main memory only when they are needed (demanded) during program execution. This means a process can run even if only part of it is in physical memory.
- **Page Fault:** An event that occurs when a program tries to access a page that is not currently loaded into any physical memory frame. This is a type of trap to the OS.
- **Steps in Handling a Page Fault (with Diagram):**
    1. **Reference to Invalid Page:** The CPU generates a logical address, and the Memory Management Unit (MMU) attempts to translate it. If the page-table entry for the requested page indicates that the page is not in memory (e.g., a "valid-invalid" bit is '0'), a **page-fault trap** occurs.
    2. **OS Takes Control:** The OS gains control and examines the internal page table to determine if the reference was invalid (e.g., illegal address) or just a page not currently in memory.
    3. **Find Free Frame:** If it's a valid reference but the page is absent, the OS finds a free physical memory frame.
    4. **Page Replacement (if no free frame):** If no free frame is available, a page replacement algorithm (like FIFO, LRU, Optimal) is invoked to select a "victim" page to be swapped out. If the victim page has been modified (dirty bit is set), it's written back to disk.
    5. **Load Page:** The desired page is read from secondary storage (disk) into the allocated frame.

6. **Update Page Table:** Once the page is loaded, the page table for the process is updated: the valid-invalid bit is set to '1', and the frame number is entered.
7. **Restart Instruction:** The instruction that caused the page fault is restarted from the beginning, and the program can now access the required page.

- +------------------+
- | CPU |
- | (Logical Address)|
- +--------|---------+
- |
- V
- +------------------+ Page-Table Entry (Valid/Invalid Bit)
- | MMU |----------> +------+------+
- | (Address Trans.) | | Page | Frame|
- +--------|---------+ | Table| Num |
- | Page Fault Trap +------+------+
- V
- +------------------+
- | Operating System | <---------- Page NOT in memory (valid-invalid bit = 0)
- | | (1. Trap)
- | (Find frame, swap)
- +--------|---------+
- | (2. Find Free Frame)
- V
- +------------------+
- | Secondary Storage| <-------- (4. Read Page)
- | (Disk) |
- +--------|---------+
- |
- V
- +------------------+
- | Physical Memory | <-------- (5. Load Page)
- | (Frames) |

- +------------------+
- ^
- | (3. Update Page Table)
- |
- +------------------+
- | OS |
- | (Restart instr.) |
- +------------------+

---

3. TLB (Translation Lookaside Buffer)

- **Question Type:** "Explain with the help of supporting diagram how TLB improves the performance of a demand paging system."
- **For 6 Marks, focus on:** Explaining TLB, its role as a cache, and how it speeds up address translation.

Notes:

- **Problem with Paging:** In a basic paging system, each memory access requires *two* memory accesses: one to fetch the page table entry, and another to fetch the actual data. This can slow down memory access significantly.
- **Translation Lookaside Buffer (TLB):**
    - A **TLB** is a small, fast, hardware cache that stores recent translations of logical page numbers to physical frame numbers. It's typically part of the MMU.
    - It's an associative memory, meaning all entries are searched in parallel.
- **How TLB Improves Performance:**

1. When the CPU generates a logical address, the MMU first checks the TLB.

2. **TLB Hit:** If the page number is found in the TLB (a "hit"), the corresponding frame number is immediately available. The physical address is formed, and the data is accessed from physical memory. This avoids the need to access the main memory-resident page table.

3. **TLB Miss:** If the page number is *not* found in the TLB (a "miss"), the MMU must then consult the page table in main memory to get the frame number.

4. **Update TLB:** Once the frame number is obtained from the page table, the page-number/frame-number pair is added to the TLB so that future references to that page can be faster. If the TLB is full, a replacement policy (e.g., LRU) is used.

- **Benefit:** The TLB dramatically reduces the effective memory access time, as most references result in a TLB hit, converting a two-memory-access process into nearly a one-memory-access process.

- **Diagram (Conceptual):**
- +-------------------------+
- | CPU |
- | (Logical Address: P, D) |
- +-----------|-------------+
- |
- V
- +-------------------------+
- | MMU |
- | +----------+ |
- | | TLB |-- (Hit) --> Physical Address (F, D)
- | +----------+ |
- | | (Miss) |
- +-----|--------------------+
- |
- V
- +-------------------------+
- | Page Table (in Main Mem) | -- (Frame Number) --> Physical Address (F, D)
- +-------------------------+
- |
- V
- +-------------------------+
- | Main Memory |
- +-------------------------+

---

4. Page Replacement Algorithms (Conceptual)

- **Question Type:** (Numerical problems are much more common, but a conceptual explanation might be asked for 6 marks). "Explain the need for page replacement. Briefly describe any two page replacement algorithms."
- **For 6 Marks, focus on:** The need for page replacement and a brief description of 2-3 algorithms, perhaps with a simple example of their rule. (Avoid full numerical trace unless explicitly asked).

Notes:

- **Need for Page Replacement:**
    - When a process demands a page that is not in main memory (a page fault occurs), the OS needs to load that page from disk into a free frame.
    - If all main memory frames are currently occupied, the OS must choose one of the existing pages in memory to **swap out** to disk to free up a frame for the incoming page. This decision is made by page replacement algorithms.
    - The goal is to select a victim page that will minimize the number of future page faults.
- **Common Page Replacement Algorithms:**

1. **First-In, First-Out (FIFO):**
    - **Rule:** Replaces the page that has been in memory for the longest time. It treats memory frames as a circular queue.
    - **Disadvantage:** Can suffer from Belady's Anomaly (more frames lead to more page faults for some reference strings).
2. **Least Recently Used (LRU):**
    - **Rule:** Replaces the page that has not been used for the longest period of time (i.e., the page whose last access time is the furthest in the past). This is based on the assumption that pages used recently are likely to be used again soon.
    - **Advantage:** Generally performs well.
    - **Disadvantage:** Requires hardware support (e.g., counters, stack) to track the age of pages, making it complex and expensive to implement accurately.
3. **Optimal Page Replacement (OPT / MIN):**
    - **Rule:** Replaces the page that will not be used for the longest period of time in the *future*.
    - **Advantage:** Guarantees the lowest possible number of page faults for a given page reference string.
    - **Disadvantage:** Impractical to implement in a real OS because it requires knowing the future page references. It serves as a benchmark for comparing other algorithms.

Remember to practice the numerical problems related to page faults from your past papers, as they are a strong focus for exams.

Chapter 5: File Management - Efficient Notes (for 6 Marks)

This chapter covers how the operating system manages files and organizes them on storage devices. File allocation methods and disk scheduling algorithms are key topics, with numerical problems for disk scheduling being very common.

---

1. File Concept and File Operations

- **Question Type:** "Explain the concept of file. State various file operations." or "Enlist and Explain in details the various operations performed on the file."
- **For 6 Marks, focus on:** Defining a file, its attributes, and describing common file operations.

Notes:

- **File Concept:**
    - A **file** is a named collection of related information that is recorded on secondary storage (e.g., hard disk, SSD). From a user's perspective, a file is the smallest logical unit of data that can be manipulated (read, written, deleted).
    - Files store programs (source code, executable code) and data (numeric, alphabetic, alphanumeric, binary).
    - **File Attributes:** Information associated with each file, managed by the OS. Common attributes include:
        - **Name:** The symbolic name for the file.
        - **Identifier:** Unique tag for the file within the file system.
        - **Type:** Identifies the file type (e.g., text, executable, image).
        - **Location:** Pointer to a device and location on that device.
        - **Size:** Current size of the file.
        - **Protection:** Controls who can read, write, or execute the file.
        - **Time, Date, and User Identification:** Data for creation, last modification, and last use.
- **File Operations:** The OS provides a set of system calls to perform operations on files.

1. **Create:** A new file is created. Requires finding space on the file system and making an entry in the directory.

2. **Write:** Data is written to a file. Requires specifying the file name and the data to be written. A write pointer tracks the current write location.

3. **Read:** Data is read from a file. Requires specifying the file name and where in memory to put the data. A read pointer tracks the current read location.

4. **Reposition (Seek):** Moves the current file pointer to a given position within the file. This allows non-sequential access to the file's data.

5. **Delete:** Removes a file from the file system. Frees up file space and removes its directory entry.

6. **Truncate:** Erases the contents of a file but keeps its attributes. The file is reset to a length of zero.

7. **Open:** Before a file can be used, it must be opened. The OS brings the file's metadata (attributes, location) into memory and returns a file handle (descriptor) to the process.

8. **Close:** Releases the resources (memory buffers, file table entries) associated with an opened file. Any modified data is written back to disk.

---

2. File Allocation Methods

- **Question Type:** "What are the three methods for allocating disk space? Explain with help each method suitable diagram, merits and demerits." or "Discuss linked and index disk space allocation methods with neat sketch."
- **For 6 Marks, focus on:** Explaining Contiguous, Linked, and Indexed allocation with their basic diagrams, advantages, and disadvantages.

Notes:

File Allocation Methods: Strategies used by the OS to store file data blocks on secondary storage.

1. **Contiguous Allocation:**
    - **Mechanism:** Each file occupies a set of contiguous (adjacent) blocks on the disk.
    - **Diagram:**
    - Disk Blocks: [F1_B0][F1_B1][F1_B2]...[F2_B0][F2_B1]...
    - Directory Entry: (File Name, Start Block, Length)
    - **Merits:**
        - Simple to implement: Only need starting block address and length.
        - Excellent performance for sequential access.
        - Good for random access (direct calculation of block address).
    - **Demerits:**
        - **External Fragmentation:** As files are deleted/created, disk space fragments into many small holes, making it hard to find large contiguous blocks for new files.
        - **Fixed Size:** File size needs to be declared at creation, making it difficult for files to grow. Compaction is needed to reclaim space.
2. **Linked Allocation:**
    - **Mechanism:** Each file is a linked list of disk blocks. Each block contains a pointer to the next block in the file, and the directory entry only stores the first block.

- **Diagram:**
- Directory Entry: (File Name, Start Block)
- Block 1: [Data | Pointer to Block 2]
- Block 2: [Data | Pointer to Block 3]
- Block 3: [Data | NULL]
- **Merits:**
  - No external fragmentation: Any free block can be used.
  - Files can grow dynamically as needed.
  - Simple to implement.
- **Demerits:**
  - Slow random access: To access a specific block, you must traverse the list from the beginning.
  - Space overhead: Each block stores a pointer, reducing actual data storage.
  - Reliability: A single pointer failure can break the entire chain.

3. **Indexed Allocation:**
   - **Mechanism:** All the pointers to a file's data blocks are collected together in one special block called the **index block**. The directory entry points to this index block.
   - **Diagram:**
   - Directory Entry: (File Name, Index Block Address)
   - Index Block:
   - [Pointer to Data Block 0]
   - [Pointer to Data Block 1]
   - [Pointer to Data Block 2]
   - ...
   - [Pointer to Data Block N]
   - **Merits:**
     - Supports direct access (random access): Simply retrieve the index block and then access any data block.
     - No external fragmentation: Any free block can be used.
     - Files can grow dynamically by adding more entries to the index block (or using linked/multi-level indexing for very large files).
   - **Demerits:**

- Space overhead: The index block itself consumes disk space.
- Small files: For very small files, the index block might be larger than the actual data, making it inefficient.
- Large files: May require multiple levels of index blocks (e.g., single, double, triple indirect pointers), increasing complexity and disk accesses for very large files.

---

3. Disk Free Space Management

- **Question Type:** "Describe how free-space management is implemented in file system. Also explain bit map with the help of an example" or "Write a note on free space management."
- **For 6 Marks, focus on:** Explaining the need for free space management and describing the Bitmap method.

Notes:

- **Need for Free Space Management:**
  - When files are created, the operating system needs to find available (free) disk blocks to store the file's data.
  - When files are deleted, their blocks become free and need to be marked as such so they can be reused later.
  - Free space management techniques keep track of all the free disk blocks to efficiently allocate and deallocate space.
- **Methods of Free Space Management:**

1. **Bit Map (Bit Vector):**
   - **Mechanism:** The free space is represented as a **bitmap** (or bit vector), which is a string of bits. Each bit corresponds to a disk block.
   - If a bit is 0, the corresponding block is free.
   - If a bit is 1, the corresponding block is occupied.
   - **Example:** For a disk with 16 blocks: 0011010000101100 means blocks 0, 1, 4, 6, 7, 8, 9, 10, 12 are free.
   - **Advantages:**
     - Simple to understand and implement.
     - Finding a contiguous block of N free blocks is easy: just search the bitmap for N consecutive zeros.
   - **Disadvantages:**
     - If the disk is very large, the bitmap itself can become very large and might need to be stored on disk, leading to disk I/O overhead.

- Requires efficient bit manipulation operations.

2. **Linked List:**

   - **Mechanism:** All free disk blocks are linked together into a free-list. Each free block contains a pointer to the next free block. The head of the list is stored in a special location on disk or in memory.
   - **Advantages:** No space overhead for a separate bitmap (the pointers are within the free blocks themselves).
   - **Disadvantages:** Slow for finding a contiguous block. A single pointer error can lose the entire list of free blocks.

3. **Grouping:** Stores the address of N free blocks in the first free block. The last of these N blocks contains the address of another N free blocks, and so on.

---

4. Disk Scheduling Algorithms

- **Question Type:** (Numerical problems are extremely common). "Suppose that a disk drive has X cylinders... Starting from the current position, what is the total distance (in cylinders) that the disk arm moves... for each of the following algorithms i)FCFS ii) SSFT iii) SCAN iv)LOOK v) C-SCAN vi) C-LOOK." or "Explain following disk scheduling techniques with its advantages i. Shortest Seek Time First ii. SCAN."
- **For 6 Marks, focus on:** Explaining the purpose of disk scheduling and describing 2-3 algorithms conceptually. For numerical problems, show all calculations and the Gantt chart.

Notes:

- **Disk Scheduling:**
  - The process of deciding the order in which disk I/O requests are serviced to minimize the total seek time (time taken to move the disk arm to the correct cylinder) and rotational latency.
  - **Goal:** Optimize disk access time and throughput.
- **Common Disk Scheduling Algorithms:**

1. **First-Come, First-Served (FCFS):**

   - **Rule:** Processes requests in the order they arrive in the queue.
   - **Advantages:** Simple, fair.
   - **Disadvantages:** Can lead to very long seek times if requests are scattered across the disk, resulting in the "arm swing" problem.

2. **Shortest-Seek-Time First (SSTF):**

   - **Rule:** Services the request that has the least seek time from the current head position.
   - **Advantages:** Higher throughput, better performance than FCFS.

- **Disadvantages:** Can lead to **starvation** for requests far from the current head position if new, closer requests keep arriving.

3. **SCAN (Elevator Algorithm):**
    - **Rule:** The disk arm starts at one end of the disk and moves towards the other end, servicing all requests in its path. When it reaches the other end, it reverses direction and continues servicing requests.
    - **Advantages:** Prevents starvation, good for systems with heavy load.
    - **Disadvantages:** Bias against requests at the "other end" of the disk (where the arm just came from).
4. **C-SCAN (Circular SCAN):**
    - **Rule:** The disk arm moves in one direction only (e.g., from inner to outer cylinders), servicing requests. When it reaches the end, it immediately jumps back to the beginning of the disk without servicing any requests on the return trip, and then starts servicing again in the same direction.
    - **Advantages:** More uniform wait time compared to SCAN, as it avoids the bias.
    - **Disadvantages:** Arm movement for return trip is wasted (no requests serviced).
    - **(LOOK and C-LOOK are variations of SCAN and C-SCAN respectively, where the arm only goes as far as the furthest request in the current direction, instead of going all the way to the end of the disk.)**

Remember to practice numerical problems for disk scheduling to master these algorithms.
