1. What is Deadlock? What are the necessary conditions for Deadlock?   
  
ANS:- Deadlock is a situation in computing where a group of processes are all waiting for each other to release resources, and as a result, none of the processes can proceed.  
  
**Necessary Conditions for Deadlock :-**  
There are **four conditions**, all of which must be true simultaneously for a deadlock to occur:  
  
**Mutual Exclusion :-**  
At least one resource must be held in a non-shareable mode. Only one process can use the resource at any given time.  
  
**Hold and Wait :-**  
A process is holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.  
  
**No Preemption :-**  
A resource cannot be forcibly taken away from a process holding it; it must be released voluntarily by the process.  
  
**Circular Wait :-**  
A set of processes exists where each process is waiting for a resource that the next process in the chain holds, forming a circular chain.  
  
If any one of these conditions is broken, deadlock can be prevented.

2. State Banker's Algorithm.  
  
ANS:- The Banker's Algorithm is a resource allocation and deadlock avoidance algorithm developed by Edsger Dijkstra. It's used in operating systems to allocate resources to multiple processes in a way that avoids deadlock.  
  
**Algorithm Steps:**  
**1. Resource Request:**  
When a process requests resources, the algorithm temporarily allocates those resources to the process.  
**2. Safety Check:**  
It then checks if the system remains in a safe state after this temporary allocation.  
**3. Safe State Check:**  
This involves finding a sequence of processes that can complete, even with the new allocation.  
**4. Decision:**  
If the system remains safe, the resource request is granted.  
If granting the request leads to an unsafe state, the request is denied, and the process must wait.

3. Describe the topics in short of below:-  
i) Fragmentation  
ii) Semaphore  
iii) Readers-Writers Problem  
iv) RAID  
  
ANS:-

**i) Fragmentation** :-  
Fragmentation is a condition in memory management where memory space is used inefficiently, reducing the system's performance or limiting its capacity. It occurs when memory is allocated and deallocated dynamically, especially in systems with variable-sized memory blocks.  
  
**Types of Fragmentation:**  
  
**Internal Fragmentation:**  
  
• Happens when fixed-sized memory blocks are allocated to processes.  
  
• The process may not use the entire block, leading to unused space within the block.  
  
• Example: Allocating 4 KB blocks to a process that needs only 3 KB wastes 1 KB per allocation.  
  
**External Fragmentation:**  
  
• Happens in systems using variable-sized memory blocks.  
  
• When memory is allocated and freed repeatedly, free spaces are scattered across memory.  
  
• Even if enough total memory is available, large processes may not get a contiguous block.  
  
**Solutions:**  
  
• Compaction: Moving memory contents to combine free memory into one large block.  
  
• Paging and Segmentation: Techniques to manage memory in non-contiguous chunks, reducing external fragmentation.  
  
**ii) Semaphore**  
A semaphore is a variable or abstract data type synchronization tool used in computer science, particularly in operating systems, to manage access to shared resources among multiple processes or threads.like multitasking operating systems.  
  
**Types of Semaphores:**  
  
**Binary Semaphore (Mutex):**  
  
• Only two values: 0 and 1.  
  
• Used for mutual exclusion to allow only one process in the critical section at a time.  
  
**Counting Semaphore:**  
  
• Integer value can range more than 1.  
  
• Useful for controlling access to a resource pool with multiple instances (e.g., printers, servers).  
  
**Use Cases:**  
  
• Preventing race conditions.  
  
• Synchronizing processes.  
  
• Ensuring mutual exclusion in critical sections.  
  
**iii) Readers-Writers Problem**  
The Readers-Writers problem is a concurrency issue in computer science where multiple processes need to access a shared resource.  
  
**Goal:**  
  
• Multiple readers can read at the same time.  
  
• Writers must have exclusive access (no reading or writing during a write).  
  
**Challenges:**  
  
• Avoid race conditions and maintain data consistency.  
  
• Prevent starvation of either readers or writers.  
  
**Solutions:**  
  
Use semaphores and counters to manage the number of readers and writers.  
  
**Apply different strategies:**  
  
• Reader Preference: Readers get access first, may starve writers.  
  
• Writer Preference: Writers get priority, may starve readers.  
  
• Fair (No Starvation): Ensures all processes get a fair turn.  
  
**iv) RAID (Redundant Array of Independent Disks)** :-  
RAID is a data storage technology that combines multiple physical hard drives into one logical unit to improve performance, reliability, or both.  
  
**Key Benefits:**  
  
• Redundancy: Prevents data loss due to disk failure.  
  
• Improved Performance: Multiple disks allow faster read/write operations.  
  
**Common RAID Levels:**  
  
**RAID 0 (Striping):**  
  
• Splits data across multiple disks.  
  
• Fast performance, no redundancy.  
  
• If one disk fails, all data is lost.  
  
**RAID 1 (Mirroring):**  
  
• Duplicates data on two disks.  
  
• High redundancy, slower write speed.  
  
• If one disk fails, the other has all the data.  
  
**RAID 5 (Striping with Parity):**  
  
• Data and parity (error-check info) are distributed across all disks.  
  
• Can tolerate one disk failure.  
  
• Good balance between performance and redundancy.  
  
**RAID 10 (1+0):**  
  
• Combines mirroring and striping.  
  
• High performance and redundancy.  
  
• Requires at least 4 disks.  
  
**Use Cases:**  
  
RAID 0: For speed (e.g., gaming).  
  
RAID 1: For data security (e.g., servers).  
  
RAID 5: For business servers needing speed + reliability.  
  
RAID 10: For high-end enterprise systems.

4. What are the different methods for handling deadlock ?   
  
ANS:- There are four methods for handling deadlocks:  
  
**Deadlock Handling Methods:**  
**1. Deadlock Prevention:** Preventing deadlocks by ensuring that at least one of the necessary conditions for deadlock cannot occur.  
**2. Deadlock Avoidance**: Avoiding deadlocks by using algorithms that predict and prevent deadlocks.  
**3. Deadlock Detection:** Detecting deadlocks and recovering from them.  
**4. Deadlock Ignorance:** Ignoring deadlocks and dealing with them only if they occur.  
  
Each method has its own trade-offs and is suited for different systems and applications.