What is Deadlock?

Definition: Deadlock is a situation in which two or more processes are unable to proceed because each is waiting for one of the others to do something.

Characterization of Deadlock:

1. Mutual Exclusion: At least one resource must be held in a non-shareable mode. This means only one process can use the resource at any given time.
2. Hold and Wait: A process holds allocated resources while waiting for additional resources that are currently being held by other processes.
3. No Preemption: Resources cannot be forcibly taken from a process holding them; they must be released voluntarily.
4. Circular Wait: A set of processes {P0, P1, ..., Pn} must exist such that P0 is waiting for a resource that is held by P1, P1 is waiting for a resource held by P2, ..., and Pn is waiting for a resource held by P0, creating a circular chain of waiting.

Prevention of Deadlock:

1. Mutual Exclusion: To prevent deadlock, we can eliminate the mutual exclusion condition by allowing resources to be shareable.
2. Hold and Wait: Processes can request all the resources they will need upfront, or release currently held resources before requesting new ones.
3. No Preemption: If a process holding some resources requests another resource that cannot be immediately allocated, then all resources currently being held are released, preventing deadlock.
4. Circular Wait: Impose a total ordering of all resource types and require that each process requests resources in an increasing order of enumeration.

Avoidance of Deadlock:

1. Resource Allocation Graph (RAG): A directed graph that depicts the resource allocation and requests among processes. Use algorithms like Banker's Algorithm to ensure that deadlock never occurs by carefully allocating resources.

Detection of Deadlock:

1. Resource Allocation Graph: Detect cycles in the RAG to identify potential deadlocks. If a cycle exists, deadlock is possible.

Recovery from Deadlock:

1. Process Termination: Identify deadlock, abort all deadlocked processes, and release their resources.
2. Resource Preemption: Preempt some resources from some processes to break the deadlock. However, this approach is complex and might lead to starvation.
3. Process Rollback: Roll back the state of some or all processes to a safe state and restart them, avoiding the conditions that led to deadlock.

Example:

Let's consider a scenario in an academic setting where students need access to shared resources like seminar rooms and projectors. Suppose there are two seminar rooms and two projectors. If student A is holding seminar room 1 and waiting for projector 1, and student B is holding projector 1 and waiting for seminar room 2, and student C is holding seminar room 2 and waiting for projector 2, while student D is holding projector 2 and waiting for seminar room 1, then a deadlock occurs. None of the students can proceed because they are waiting for resources held by others in a circular manner.

In this scenario, prevention methods like breaking the hold and wait condition by requesting all necessary resources upfront or avoiding circular wait by imposing a total ordering of resource types can help prevent deadlock. Additionally, detection methods like checking for cycles in the resource allocation graph can help identify potential deadlocks early on, allowing for recovery strategies to be employed.

File Concept:

Definition: A file is a named collection of related data that is stored on secondary storage devices like hard disks or SSDs. Each file has a unique name and is managed by the operating system.

Attributes of a File:

- Name: The unique identifier of the file.
- Type: Indicates the format or content type of the file (e.g., text file, image file, executable file).

- Location: Physical location on the storage device where the file is stored.
- Size: The amount of data the file occupies.
- Protection: Permissions specifying who can access and modify the file.
- Time and Date: Metadata indicating when the file was created, last accessed, and last modified.

Access Methods:

Sequential Access:

- Data is accessed in a sequential manner, starting from the beginning of the file and proceeding sequentially through the file until the desired data is found.
- Suitable for tasks like reading log files or processing large datasets sequentially.

Direct Access:

- Data can be accessed randomly without reading through preceding records.
- Requires a method to calculate the physical address of the desired data based on its logical address.
- Commonly used in database systems and file systems supporting random access.

Directory Structure:

Single-Level Directory:

- All files are stored in a single directory.
- Simplest directory structure but can become cluttered and difficult to manage as the number of files grows.

Two-Level Directory:

- Organizes files into hierarchical structures, with each user having their directory.
- Enhances organization and allows users to manage their files independently.

Tree-Structured Directory:

- Allows for the creation of subdirectories, forming a tree-like structure.
- Provides a more flexible organization and easier navigation of files and directories.

Acyclic-Graph Directory:

- Allows directories to have multiple parent directories, enabling files to be shared among different projects or categories.
- Offers increased flexibility but requires careful management to avoid cyclic references.

General Graph Directory:

- Allows directories to form arbitrary graphs, enabling complex relationships between files and directories.
- Offers maximum flexibility but can be challenging to implement and manage.

Protection:

Access Control Lists (ACL):

- Detailed lists associated with each file that specify which users or system processes are granted access rights to the file and what operations are allowed.
- Provides fine-grained control over file access and permissions.

File Permissions:

- Assigns read, write, and execute permissions to files for the owner, group, and others.
- Uses permission bits to represent different access levels (e.g., read = 4, write = 2, execute = 1).

Encryption:

- Encrypts file contents to protect sensitive data from unauthorized access.
- Requires encryption keys to decrypt and access the file.

File-System Structure:

File-System Layout:

- Defines how files are organized and managed on the storage medium.
- Includes metadata structures like file allocation tables (FAT), inode tables, and master file tables (MFT) for tracking file attributes and locations.

Metadata:

- Information about files and directories stored by the file system, including file names, sizes, creation dates, and permissions.
- Managed by the operating system to facilitate file management and access control.

Journaling:

- Technique used in file systems to maintain a consistent state in the event of system crashes or power failures.
- Logs changes to the file system in a journal before committing them to disk, allowing for recovery and consistency checks during system startup.

Directory Implementation:

Linear List:

- Simplest directory implementation where directories are represented as linear lists of file names and pointers to the actual files.
- Provides straightforward access but can be inefficient for large directories.

Hash Table:

- Uses hash tables for efficient lookup of file names, reducing search time to O(1) on average.
- Requires a well-designed hash function to minimize collisions and ensure even distribution of file names.

Binary Search Tree:

- Organizes directories as binary search trees, allowing for efficient searching and insertion of file names.
- Provides logarithmic search time but may become unbalanced in the presence of frequent insertions or deletions.

B-Trees:

- Balanced tree data structure optimized for disk-based storage systems.
- Allows for efficient insertion, deletion, and search operations, making it suitable for large directories and file systems.

Allocation Methods:

Contiguous Allocation:

- Allocates files into contiguous blocks of disk space, reducing fragmentation and improving sequential access performance.
- Requires advance knowledge of file sizes and suffers from external fragmentation.

Linked Allocation:

- Allocates each file as a linked list of disk blocks, where each block contains a pointer to the next block.
- Flexibility in file size but suffers from increased overhead due to pointers and inefficient random access.

Indexed Allocation:

- Allocates each file an index block containing pointers to the actual data blocks.
- Enables direct access to file blocks without traversing pointers but requires additional overhead for index maintenance.

Multilevel Indexing:

- Extends indexed allocation by using multiple levels of indirection to accommodate large files.
- Reduces the size of index blocks and improves performance for large files.

Free-Space Management:

Bitmaps:

- Maintains a bitmap where each bit represents the allocation status of a disk block (allocated or free).
- Allows for efficient checking and allocation of free space but requires additional space overhead.

Linked Lists:

- Maintains a linked list of free disk blocks, where each block contains a pointer to the next free block.
- Simple and efficient for small file systems but suffers from traversal overhead for large file systems.

Grouping:

- Divides the disk into groups of contiguous blocks and maintains a bitmap for each group.
- Reduces the size of the free space bitmap and improves performance for large disks.

Example:

In an academic institution, students store their assignments and research papers on a shared network drive. Each department has its directory structure, with subdirectories for courses and individual student directories. File permissions ensure that only authorized users can access and modify files, while encryption protects sensitive research data. The file system uses indexed allocation to manage file storage efficiently, with bitmap-based free-space management for tracking available disk space. Journaling ensures the integrity of the file system, allowing for recovery in the event of system crashes.