OS MIDS-2 IMP

SAQ:

1. How to avoid critical section?

To avoid the critical section problem in concurrent programming, synchronization mechanisms like semaphores, mutexes, or monitors can be used. By ensuring that only one process or thread can enter the critical section at a time, we prevent race conditions and maintain data integrity.

2. Give the necessary & sufficient conditions for deadlocks?

- Mutual Exclusion: At least one resource must be held in a non-sharable mode.

- Hold and Wait: Processes must be holding resources while waiting for additional resources.

- No Preemption: Resources cannot be forcibly taken away from a process; they must be released voluntarily.

- Circular Wait: A circular chain of two or more processes, each waiting for a resource held by the next one in the chain.

3. What do you mean by Internal Fragmentation & External Fragmentation?

- Internal Fragmentation: Occurs when a process or file is allocated more memory or disk space than it actually requires. The unused memory within the allocated block is wasted.

- External Fragmentation: The phenomenon where free memory or disk space becomes scattered or fragmented into small non-contiguous blocks over time, making it challenging to allocate large contiguous chunks of memory to new processes or files.

4. List out file system structures.

- Superblock: Contains metadata about the file system, such as the total number of blocks, block size, and free block information.

- Inode: A data structure that stores information about individual files, including file permissions, size, timestamps, and disk block pointers.

- Directory: A special type of file that maintains records of file names and corresponding inode numbers.

- Data Blocks: The actual blocks on the disk where the file data is stored.

5. Differentiate between Monitors and Semaphores.

- Monitors: Higher-level synchronization construct providing mutual exclusion and synchronization through procedures or methods.

- Semaphores: Lower-level synchronization primitives used to control access to resources in a concurrent environment.

6. State the importance of Disk Scheduling.

Disk Scheduling is essential to optimize the access time to data stored on disk. Efficient disk scheduling algorithms reduce seek time and rotational latency, improving overall system performance and ensuring fair and timely access to data for different processes.

7. Why Virtual Memory is required?

Virtual Memory is required to provide the illusion of a vast address space to processes, even when the physical memory (RAM) is limited. It allows the execution of processes that are larger than the available physical memory and enables efficient memory management through the use of page or segment-based memory mapping techniques.

8. Enlist various page replacement algorithms.

- FIFO (First-In-First-Out)
- LRU (Least Recently Used)
- Optimal Page Replacement
- LFU (Least Frequently Used)
- NRU (Not Recently Used)
- Second Chance (Clock)

9. What are the Classical synchronous problems?

- Producer-Consumer Problem
- Dining Philosophers Problem
- Readers-Writers Problem

10. Define the file system and its access methods?

A file system is a method used by an operating system to organize and manage files on a storage device, such as a hard disk or SSD. Access methods define how data can be read from or written to files and include sequential access, direct access (random access), and indexed access.

LAQ:

11. Solve the following page reference string using the Optimal page replacement algorithm with three initially empty frames. Page reference string: 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6. How many page faults would occur?

Using the Optimal page replacement algorithm, we can calculate the number of page faults as follows:

Page Reference String: 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

Number of Frames: 3

Initial Frames: [, ,]

<u>Page Faults:</u>

1. 1 - Page fault (Frames: [1, ,])
2. 2 - Page fault (Frames: [1, 2,])
3. 3 - Page fault (Frames: [1, 2, 3])
4. 4 - Page fault (Frames: [4, 2, 3])
5. 2 - Page fault (Frames: [4, 2, 3]) - Replacing 1
6. 1 - Page fault (Frames: [4, 2, 1]) - Replacing 3
7. 5 - Page fault (Frames: [4, 2, 5]) - Replacing 1
8. 6 - Page fault (Frames: [4, 2, 6]) - Replacing 5
9. 2 - Page fault (Frames: [4, 2, 6]) - Replacing 4
10. 1 - Page fault (Frames: [1, 2, 6]) - Replacing 2
11. 2 - No page fault (Frames: [1, 2, 6])
12. 3 - No page fault (Frames: [1, 2, 6])
13. 7 - Page fault (Frames: [1, 7, 6]) - Replacing 2
14. 6 - Page fault (Frames: [1, 7, 6]) - Replacing 1
15. 3 - Page fault (Frames: [3, 7, 6]) - Replacing 6
16. 2 - Page fault (Frames: [3, 2, 6]) - Replacing 7
17. 1 - Page fault (Frames: [3, 2, 1]) - Replacing 6
18. 2 - No page fault (Frames: [3, 2, 1])
19. 3 - No page fault (Frames: [3, 2, 1])
20. 6 - No page fault (Frames: [3, 2, 1])

<u>Total Page Faults: 9</u>

12. Discuss Disk Scheduling algorithms.

Disk Scheduling algorithms are used to manage the order in which read/write requests are serviced by a disk. These algorithms aim to optimize disk access and reduce seek time, which is the time taken by the disk arm to position the read/write head over the requested track. Some commonly used disk scheduling algorithms are:

a) FCFS (First-Come-First-Serve): In FCFS, the disk services the requests in the order they arrive in the queue. It is simple to implement but may lead to poor performance due to high seek times when requests are scattered.

b) SSTF (Shortest Seek Time First): SSTF selects the request with the shortest seek time from the current head position. It reduces seek time and improves performance compared to FCFS.

c) SCAN: The SCAN algorithm moves the disk arm in one direction while servicing the requests along the way. When it reaches the end, it reverses direction and continues servicing requests in the opposite direction. This helps to reduce the average seek time.

d) C-SCAN (Circular SCAN): Similar to SCAN, but instead of reversing direction, it jumps to the other end and starts servicing requests from there. This prevents unnecessary head movement back to the starting position.

e) LOOK: LOOK is an improved version of SCAN where the disk arm only goes up to the last request in each direction and reverses without reaching the end of the disk.

f) C-LOOK (Circular LOOK): C-LOOK is an improved version of C-SCAN, where the disk arm jumps to the other end and starts servicing requests without reaching the end of the disk.

13. Explain deadlock conditions and the Banker's algorithm in detail.

Deadlock Conditions:

Deadlock is a situation in a multi-process system where each process is waiting for a resource that is held by another process, leading to a standstill where no process can proceed. Deadlock arises due to the following four necessary conditions:

1. Mutual Exclusion: Each resource can be accessed by only one process at a time. Once a process acquires a resource, other processes are denied access until the resource is released.

2. Hold and Wait: A process holding at least one resource requests additional resources that are held by other processes. The process will be in a waiting state until all the requested resources are acquired.

3. No Preemption: Resources cannot be forcibly taken away from a process. Only the process that holds a resource can release it voluntarily.

4. Circular Wait: A circular chain of processes exists, where each process holds the resource that the next process in the chain is waiting for. The last process in the chain is waiting for a resource held by the first process, completing the circular wait.

Banker's Algorithm:

The Banker's algorithm is a deadlock avoidance algorithm used by the operating system to prevent deadlocks from occurring. It is based on the idea of resource allocation to processes in a safe manner, ensuring that the system will not enter into a deadlock state.

The Banker's algorithm works as follows:

- It requires information about the maximum demand of each process, the current allocation of resources, and the available resources in the system.

- When a process requests resources, the algorithm checks if granting those resources will result in a safe state or not.

- A state is considered safe if there exists a sequence of resource allocations to processes that allows them to complete their tasks without encountering deadlock.

- If the requested resources can be safely allocated, the allocation is done; otherwise, the process must wait until the resources are available.

- The Banker's algorithm dynamically evaluates resource requests and resource releases to ensure that the system remains in a safe state and avoids deadlock.

14. Explain RAID architecture.

RAID (Redundant Array of Independent Disks) is a data storage architecture that involves combining multiple physical disk drives into a single logical unit. The primary purpose of RAID is to improve data reliability, availability, and performance. It achieves these objectives through different RAID levels, each with its own data distribution and redundancy techniques.

RAID levels include RAID 0, RAID 1, RAID 5, RAID 6, RAID 10, and more. RAID 0 uses striping to enhance read/write performance but offers no redundancy. RAID 1 uses mirroring to duplicate data across disks, providing high fault tolerance. RAID 5 utilizes distributed parity to combine striping and redundancy for improved performance and fault tolerance. RAID 6 enhances fault tolerance further by using double distributed parity. RAID 10 combines mirroring and striping for both performance and redundancy.

RAID architecture is commonly used in servers and storage systems to meet diverse data storage needs. The choice of RAID level depends on specific requirements, such as performance, data protection, and capacity utilization. By leveraging RAID technology, organizations can enhance data integrity and optimize storage system performance effectively.

15. Explain Page replacement algorithm? (LRU)

The Page Replacement Algorithm LRU (Least Recently Used) is a crucial technique used in operating systems to manage virtual memory efficiently. In scenarios where physical memory (RAM) is limited and cannot accommodate all active processes, the LRU algorithm helps decide which page to evict from memory when a new page needs to be brought in.

LRU operates on the principle that the page that has not been accessed for the longest time is the least likely to be used again in the near future. It maintains a record of the order in which pages are accessed and uses this information to make eviction decisions.

Here's how the LRU algorithm works:

1. Page Access Tracking: The operating system maintains a data structure, such as a linked list or a queue, to track the order in which pages are accessed. When a process reads or writes to a page, the LRU algorithm updates this data structure to reflect the page's most recent access.

2. Page Replacement: When a page fault occurs (i.e., the requested page is not present in RAM), the operating system uses the LRU data structure to identify the least recently used page. The page that was accessed the furthest back in time is selected for replacement.

3. Updating Page Access History: After a page is selected for replacement, the new page is brought into memory, and the LRU data structure is updated to reflect the most recent access, ensuring accurate tracking for future evictions.

The LRU algorithm's advantage lies in its ability to minimize the number of page faults, as it typically evicts pages that are less likely to be used soon, thus promoting better memory utilization and system performance.

However, implementing the LRU algorithm requires additional overhead to maintain the page access history, and its performance can be impacted if the access history data structure becomes too large. Various techniques, such as approximation algorithms or hardware support, are used to optimize the LRU algorithm's implementation and achieve efficient page replacement in modern operating systems.

16. Describe Classical synchronization problem.

The Classical Synchronization Problem is a fundamental issue in concurrent programming, where multiple processes or threads share common resources and need to coordinate their access to avoid conflicts and ensure data consistency. The problem arises due to the non-deterministic nature of process scheduling, which can lead to race conditions and unexpected behaviour when multiple processes attempt to access shared resources simultaneously.

Several classical synchronization problems highlight these challenges:

1. The Producer-Consumer Problem: In this problem, there are two types of processes, producers, and consumers. Producers generate data and put it into a shared buffer, while consumers retrieve and process the data from the buffer. The challenge is to ensure that the buffer is accessed safely, so a consumer does not read from an empty buffer, and a producer does not overwrite data in the buffer before it is consumed.

2. The Readers-Writers Problem: This problem involves two types of processes, readers, and writers. Multiple readers can access a shared resource simultaneously, but only one writer can access it exclusively. The goal is to prevent a writer from accessing the resource while any reader is currently reading to maintain data integrity.

3. The Dining Philosophers Problem: In this scenario, a group of philosophers sits at a circular table with bowls of rice and chopsticks. To eat, a philosopher must hold both chopsticks adjacent to their seat. However, there are limited chopsticks, leading to potential deadlock situations if all philosophers try to pick up the chopsticks simultaneously.

Addressing the Classical Synchronization Problem involves implementing synchronization mechanisms to control the access to shared resources and maintain mutual exclusion, avoiding race conditions and data inconsistency. Techniques like semaphores, mutexes, and condition variables are employed to enforce critical sections where only one process can access the shared resource at a time, ensuring orderly and safe resource utilization. Solving these synchronization problems is essential for building reliable and robust concurrent software systems.

17. Write about Deadlocks? State four necessary conditions for a Deadlock situation to arise.

Deadlocks are a critical issue in operating systems that occur when two or more processes are unable to proceed because each process is waiting for a resource that is held by another process. Deadlocks can lead to a system standstill and are crucial to understand for effective resource management in concurrent systems.

Four necessary conditions for a deadlock to arise are:

1. Mutual Exclusion: At least one resource must be non-shareable, meaning only one process can access it at a time. When a process holds a non-shareable resource, no other process can access it until the resource is released.

2. Hold and Wait: A process must be holding at least one resource while waiting to acquire additional resources that are held by other processes. The process does not release any of its current resources while waiting for others, leading to a potential resource deadlock.

3. No Preemption: Resources cannot be forcibly taken away from a process; they must be released voluntarily by the process holding them. In a deadlock scenario, no process will voluntarily release its resources, preventing other processes from proceeding.

4. Circular Wait: A circular chain of two or more processes must exist, where each process is waiting for a resource held by the next process in the chain. The last process in the chain is waiting for a resource held by the first process, creating a circular dependency that leads to a deadlock.

To prevent or resolve deadlocks, various techniques can be employed, such as deadlock detection and recovery algorithms, resource allocation strategies, and using synchronization mechanisms like semaphores and mutexes to avoid circular waits and enforce proper resource handling. Understanding the necessary conditions for deadlocks is crucial for designing efficient and deadlock-free systems.

18. Solve the following: Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The current head position is at cylinder 143. The queue of pending requests is: 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130. What is the total distance that the disk arm moves to satisfy all the pending requests for each of the following Disk scheduling algorithms?

a) SSTF

b) FCFS

Given disk queue: 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

<u>a) SSTF (Shortest Seek Time First) Algorithm:</u>

1. Move from 143 to 130 (seek distance = 13)
2. Move from 130 to 86 (seek distance = 44)
3. Move from 86 to 913 (seek distance = 827)
4. Move from 913 to 948 (seek distance = 35)
5. Move from 948 to 1022 (seek distance = 74)
6. Move from 1022 to 1509 (seek distance = 487)
7. Move from 1509 to 1750 (seek distance = 241)
8. Move from 1750 to 1774 (seek distance = 24)
9. Move from 1774 to 1470 (seek distance = 304)

Total Seek Distance (SSTF): 13 + 44 + 827 + 35 + 74 + 487 + 241 + 24 + 304 = 3069 cylinders

<u>b) FCFS (First-Come-First-Serve) Algorithm:</u>

1. Move from 143 to 86 (seek distance = 57)
2. Move from 86 to 1470 (seek distance = 1384)
3. Move from 1470 to 913 (seek distance = 557)
4. Move from 913 to 1774 (seek distance = 861)
5. Move from 1774 to 948 (seek distance = 826)
6. Move from 948 to 1509 (seek distance = 561)
7. Move from 1509 to 1022 (seek distance = 487)
8. Move from 1022 to 1750 (seek distance = 728)
9. Move from 1750 to 130 (seek distance = 1620)

Total Seek Distance (FCFS): 57 + 1384 + 557 + 861 + 826 + 561 + 487 + 728 + 1620 = 7184 cylinders