IMPORTANT Topics

1. Priority Inversion

- **Problem:** A high-priority task is waiting for a lower-priority task to release a resource, but a medium-priority task preempts the lower-priority task, leading to a deadlock or significant delays.
- **Solution:** Priority inheritance protocol, where the lower-priority task temporarily inherits the higher priority to prevent preemption.

---

2. Deadlock

- **Problem:** A set of processes are blocked because each process is waiting for a resource that another process holds.
- **Conditions:** Mutual exclusion, hold and wait, no preemption, and circular wait.
- **Solutions:**
    - **Prevention:** Break at least one of the necessary conditions.
    - **Avoidance:** Use algorithms like Banker's Algorithm.
    - **Detection and Recovery:** Detect cycles in resource allocation and preempt resources.

---

3. Starvation (or Aging)

- **Problem:** A low-priority process waits indefinitely because higher-priority processes continue to preempt it.
- **Solution:** Aging, where the priority of waiting processes increases over time.

---

4. Producer-Consumer Problem (Bounded Buffer Problem)

- **Problem:** The producer and consumer processes share a bounded buffer, leading to synchronization issues.
- **Solution:** Use semaphores or mutex locks to synchronize access to the buffer.

---

5. Reader-Writer Problem

- **Problem:** Multiple readers and writers access a shared resource, potentially leading to inconsistent states.
- **Solutions:**

- First Reader-Writers Problem: Ensure readers don't starve writers.
- Second Reader-Writers Problem: Ensure writers don't starve readers.
- Implement using semaphores or reader-writer locks.

---

7. Critical Section Problem

- **Problem:** Multiple processes accessing a shared resource can lead to race conditions.
- **Solution:** Use synchronization primitives like locks, semaphores, or monitors to ensure mutual exclusion.

---

8. Page Fault and Thrashing

- **Problem:** When the required page is not in memory (page fault) or excessive page faults occur due to insufficient memory allocation (thrashing).
- **Solution:**
  - Use efficient page replacement algorithms (e.g., LRU, FIFO).
  - Allocate sufficient memory or adjust the degree of multiprogramming.

---

9. Memory Fragmentation

- **Problem:** Free memory is split into small blocks, leading to inefficient memory usage.
- **Solutions:**
  - **External Fragmentation:** Use compaction or paging.
  - **Internal Fragmentation:** Use smaller allocation units.

---

10. File Allocation Problems

- **Problem:** Issues like fragmentation or file allocation table (FAT) corruption.
- **Solution:** Use efficient allocation methods like contiguous allocation, linked allocation, or indexed allocation.

---

11. Banker's Algorithm Problem

- **Problem:** Simulate deadlock avoidance by determining safe and unsafe states.
- **Solution:** Implement the Banker's Algorithm to check resource allocation safety.

---

12. Process Synchronization Problems

- **Problem:** Processes need to coordinate and share resources without conflicts.
- **Solution:** Use synchronization mechanisms like semaphores, monitors, and condition variables.

---

13. Load Balancing

- **Problem:** Uneven distribution of tasks across processors in a multi-processor system.
- **Solution:** Implement dynamic or static load balancing algorithms.

---

14. CPU Scheduling Problems

- **Problem:** Ensuring fairness, efficiency, and responsiveness in CPU allocation.
- **Solution:** Use scheduling algorithms like FCFS, SJF, SRTF, RR, and Priority Scheduling.

---

15. I/O Bottleneck

- **Problem:** Slow I/O devices limit overall system performance.
- **Solution:** Implement buffering, caching, or asynchronous I/O operations.

---

16. Thrashing in Virtual Memory

- **Problem:** Excessive paging due to insufficient frames for processes.
- **Solution:** Increase physical memory, use working set model, or reduce degree of multiprogramming.
