Operating Systems Interview Preparation Guide

Process Management

- **Process**: Program in execution with PCB (Process Control Block)
- **Thread**: Lightweight process sharing same resources; faster context switching
- **States**: New → Ready → Running → Waiting → Terminated
- **Context Switch**: Saving current process state and loading another (expensive due to cache invalidation)
- **Scheduling Algorithms**:
  - FCFS: First come, first served (simple, non-preemptive)
  - SJF: Shortest job first (optimal but requires prediction)
  - Round Robin: Time slices for each process (fair but overhead)
  - Priority: Higher priority first (can cause starvation)

Memory Management

- **Virtual Memory**: Maps virtual addresses to physical addresses, enabling larger memory space
- **Paging**: Fixed-size blocks (pages); eliminates external fragmentation
- **Segmentation**: Variable-sized logical units; can cause fragmentation
- **Page Replacement**:
  - FIFO: Replace oldest page (simple but suboptimal)
  - LRU: Replace least recently used (optimal but tracking intensive)
  - Clock/Second-chance: Approximates LRU with lower overhead

Concurrency Control

- **Critical Section**: Code accessing shared resources that must execute atomically
- **Mutual Exclusion**: Ensuring only one process accesses shared resources
- **Synchronization Primitives**:
  - Mutex: Binary lock for exclusive access
  - Semaphore: Counter for controlling resource access
  - Monitor: High-level construct combining mutex and condition variables
- **Deadlock**: Circular waiting where processes hold resources others need
  - Prevention: Deny one of four necessary conditions
  - Avoidance: Banker's algorithm ensures safe states
  - Detection & Recovery: Periodically check and break deadlocks

File Systems

- **File Allocation**:
  - Contiguous: Fast access but fragmentation issues
  - Linked: No fragmentation but slow random access
  - Indexed: Flexible access through index blocks
- **Directory Structure**: Hierarchical organization of files
- **Journaling**: Transaction log preventing corruption during crashes

Disk Management

- **Disk Scheduling**:
  - FCFS: Simple but suboptimal seek times
  - SCAN/Elevator: Serves requests as disk arm moves back and forth
  - C-SCAN: One-directional scan for more uniform waiting times
- **RAID**: Redundant Array of Independent Disks for performance/reliability

Common Interview Questions

Process Management

1. **Process vs Thread?** Threads share memory space and are lightweight; processes are isolated with separate memory.
2. **What happens during context switch?** Save registers, switch page tables, flush TLB, load new process state.
3. **Why is Round Robin better for interactive systems?** Gives fair CPU time to all processes, ensuring responsiveness.

Memory Management

1. **Why virtual memory?** Enables larger address spaces, isolation, and memory protection.
2. **What is thrashing?** Excessive paging where the system spends more time swapping than executing.
3. **How to handle fragmentation?** Internal: fixed block sizes; External: compaction or non-contiguous allocation.

Concurrency

1. **How to prevent deadlock?** Break one condition: mutual exclusion, hold-and-wait, no preemption, or circular wait.
2. **Producer-Consumer solution?** Buffer with mutex and two semaphores for empty and full slots.
3. **What is priority inversion?** Lower priority process holds resource needed by higher priority process.

File Systems

1. **Why journaling?** Records pending operations for recovery after crashes.
2. **Inode vs FAT?** Inode: UNIX-style with metadata separated; FAT: linked list of clusters.
3. **Why buffer cache?** Improves performance by keeping frequently accessed blocks in memory.

Advanced Topics

1. **Monolithic vs Microkernel?** Monolithic: all OS services in kernel space (faster); Microkernel: minimal kernel with services as processes (more stable).
2. **How system calls work?** User program triggers trap, switches to kernel mode, executes privileged operation, returns to user mode.
3. **Copy-on-write?** Shares memory pages until modification, then creates copy (optimizes fork()).

Performance Optimization

- **CPU Optimization**: Efficient scheduling, process affinity, minimizing context switches
- **Memory Optimization**: Proper page size, working set estimation, reducing fragmentation
- **I/O Optimization**: Efficient disk scheduling, caching, asynchronous I/O operations
- **Cache Optimization**: Locality of reference, appropriate cache sizes, prefetching

OS Security

- **Authentication**: Verifying user identity (passwords, biometrics)
- **Authorization**: Controlling access to resources (ACLs, capabilities)
- **Isolation**: Preventing processes from interfering with each other
- **Protection Rings**: Hierarchical privilege levels (kernel mode vs user mode)