Operating Systems (AI 3002) – 120+ Viva Q&A

Contents

SECTION I: INTRODUCTION TO OPERATING SYSTEM

Unit 1: Introduction to OS (4 Hours)

Q1. What is an Operating System?
Ans: An OS is system software that manages hardware resources and provides a platform for application programs to run. It acts as an intermediary between user and hardware.

Q2. List the primary goals of an OS.
Ans: Convenience (user-friendly interface), efficiency (optimal resource utilization), and ability to evolve (support new functions without interference).

Q3. Name five services provided by an OS.
Ans: Program execution, I/O operations, file system manipulation, communication, error detection.

Q4. What are system calls? Give five examples.
Ans: System calls are kernel-level interfaces for user programs. Examples: `fork()`, `exec()`, `wait()`, `open()`, `read()`.

Q5. Differentiate between system calls and library functions.
Ans: System calls involve kernel mode switch and OS services; library functions run in user space and may or may not use system calls.

Q6. What is a batch OS? Give one drawback.
Ans: Jobs are grouped and executed without user interaction. Drawback: No interactivity; low CPU utilization due to I/O waits.

Q7. Explain multiprogramming.
Ans: Multiple programs reside in memory; CPU switches among them to keep itself busy while one waits for I/O.

Q8. What is time-sharing? How is it different from multiprogramming?
Ans: Time-sharing allows multiple users to interact simultaneously using CPU time slices. Multiprogramming focuses on CPU utilization; time-sharing adds interactivity.

Q9. Define real-time OS. Give two types.
Ans: An OS that guarantees task completion within strict time limits. Types: Hard RTOS (missed deadline = system failure) and Soft RTOS (occasional misses acceptable).

Q10. What is a distributed OS?
Ans: An OS that manages a collection of independent computers appearing as a single system to users. Resources are shared across networked machines.

Q11. What is a parallel OS?
Ans: An OS designed to manage multiprocessor systems where multiple CPUs work simultaneously on a single task.

Q12. Explain the evolution from batch to modern OS.
Ans: Batch (no interaction) → Multiprogramming (better CPU use) → Time-sharing (interactivity) → Personal computing → Networked/distributed → Mobile/cloud OS.

Q13. What is BASH?
Ans: Bourne Again SHell—a command-line interpreter for Unix/Linux that supports scripting and automation.

Q14. Give five basic Linux commands.
Ans: `ls`, `cd`, `pwd`, `mkdir`, `rm`.

Q15. What is the role of a shell in OS?
Ans: A shell is a command interpreter that provides a user interface to access OS services via commands or scripts.

Unit 2: Process Management (6 Hours)

Q16. Define a process.
Ans: A program in execution, including its code, data, stack, heap, and PCB (Process Control Block).

Q17. What is a Process Control Block (PCB)? What does it contain?
Ans: A data structure storing process state: PID, program counter, CPU registers, memory limits, open files, scheduling info, etc.

Q18. Draw and explain the 5-state process model.
Ans: New $\rightarrow$ Ready $\rightarrow$ Running $\rightarrow$ (Blocked/Waiting) $\rightarrow$ Terminated. Blocked occurs on I/O wait; Ready means ready to run but not scheduled.

Q19. What is context switching? Why is it costly?
Ans: Saving the state of one process and loading another. Costly due to CPU overhead (cache flush, TLB invalidation, register save/restore).

Q20. What is a thread? Why use threads?
Ans: A lightweight unit of execution within a process. Advantages: Faster creation, shared memory, improved responsiveness, better utilization of multiprocessors.

Q21. Differentiate user-level and kernel-level threads.
Ans: User threads: managed by user library; kernel unaware $\rightarrow$ faster but blocking one blocks all. Kernel threads: managed by OS $\rightarrow$ true concurrency but slower.

Q22. What is concurrency? What problems arise due to it?
Ans: Multiple computations executing during overlapping time periods. Problems: race conditions, deadlock, starvation, inconsistent data.

Q23. What is a race condition?
Ans: When outcome depends on the sequence/timing of uncontrollable events (e.g., two threads modifying shared variable without sync).

Q24. Define critical section.
Ans: A code segment accessing shared resources that must not be concurrently executed by more than one process/thread.

Q25. State the three requirements for solving critical section problem.
Ans: Mutual exclusion, progress, bounded waiting.

Q26. What is a mutex? How is it implemented?
Ans: A mutual exclusion lock allowing only one thread to enter critical section. Implemented via atomic instructions like Test-and-Set or Compare-and-Swap.

Q27. What is a semaphore? Types?
Ans: A synchronization variable. Binary (0/1) for mutual exclusion; counting (¿1) for resource counting.

Q28. Explain the Producer-Consumer problem.
Ans: Producer adds items to buffer; consumer removes. Use empty/full semaphores and mutex to avoid overflow/underflow.

Q29. Describe the Readers-Writers problem.
Ans: Multiple readers can access data simultaneously, but writers need exclusive access. Solutions prioritize readers or writers.

Q30. Explain the Dining Philosophers problem.
Ans: Five philosophers alternate between thinking and eating; each needs two adjacent forks. Poor coordination leads to deadlock or starvation.

Q31. How to solve Dining Philosophers using semaphores?
Ans: Use an array of 5 mutexes for forks and a global semaphore limiting concurrent eaters to 4, breaking circular wait.

Q32. What is a monitor?
Ans: A high-level synchronization construct that encapsulates shared data and procedures, ensuring only one process executes inside at a time.

Q33. Why are orphan and zombie processes created?
Ans: Zombie: child terminates but parent hasn't read its exit status. Orphan: parent dies before child; child adopted by init.

Q34. How does `fork()` work?
Ans: Creates a child process as a copy of the parent. Returns 0 to child, child PID to parent.

Q35. What is `exec()`?
Ans: Replaces current process image with a new program (e.g., `execvp("ls", args)`).

Q36. What is `wait()`?
Ans: Parent suspends until child terminates, then collects exit status to prevent zombie.

Q37. Explain Pthreads.
Ans: POSIX threads API for multithreading in C. Includes functions like `pthread_create()`, `pthread_join()`, `pthread_mutex_lock()`.

Q38. What is thread safety?
Ans: Code that functions correctly when accessed by multiple threads simultaneously (uses proper synchronization).

Q39. What is deadlock in multithreading?
Ans: Two or more threads wait indefinitely for resources held by each other (e.g., Thread A locks X, waits for Y; Thread B locks Y, waits for X).

Q40. How to avoid race conditions in threads?
Ans: Use mutexes, semaphores, atomic operations, or thread-safe data structures.

SECTION II: CPU SCHEDULING, DEADLOCK, MEMORY, AND I/O

Unit 3: Uniprocessor Scheduling (4 Hours)

Q41. What is CPU scheduling?
Ans: The process of selecting which ready process gets the CPU next.

Q42. List scheduling criteria.
Ans: CPU utilization, throughput, turnaround time, waiting time, response time.

Q43. What is turnaround time?
Ans: Total time from submission to completion: `Completion - Arrival`.

Q44. Define waiting time.
Ans: Total time spent in ready queue.

Q45. Explain FCFS scheduling. Is it preemptive?
Ans: First-Come-First-Served; non-preemptive. Suffers from convoy effect.

Q46. What is SJF? Why is it optimal?
Ans: Shortest Job First minimizes average waiting time if burst times are known.

Q47. Differentiate preemptive and non-preemptive SJF.
Ans: Non-preemptive: once started, runs to completion. Preemptive (SRTF): if new shorter job arrives, current is preempted.

Q48. How does Round Robin work?
Ans: Each process gets a fixed time quantum. After quantum, it's preempted and moved to end of ready queue.

Q49. What happens if time quantum is too small or too large in RR?
Ans: Too small $\rightarrow$ high context switch overhead. Too large $\rightarrow$ behaves like FCFS.

Q50. Explain Priority Scheduling. What is starvation?
Ans: Highest priority process runs first. Starvation: low-priority processes may never run.

Q51. How to prevent starvation in priority scheduling?
Ans: Aging: gradually increase priority of waiting processes.

Q52. Which algorithm is used in modern OS?
Ans: Multilevel Feedback Queue (not in syllabus but conceptually built from these), combining RR and priority with aging.

Q53. Can SJF be implemented in real life? Why/why not?
Ans: Rarely—burst times are unknown. Approximated using exponential averaging.

Q54. What is response time? Why is it important for interactive systems?
Ans: Time from request to first response. Critical for user experience in interactive apps.

Q55. Compare FCFS and RR in terms of fairness.
Ans: FCFS unfair to short jobs; RR fair as all get equal CPU time.

Unit 4: Deadlock (4 Hours)

Q56. Define deadlock.
Ans: A situation where a set of processes are blocked forever, each waiting for a resource held by another.

Q57. State four necessary conditions for deadlock.
Ans: Mutual exclusion, hold and wait, no preemption, circular wait.

Q58. How does mutual exclusion contribute to deadlock?
Ans: Only one process can use a resource at a time; others must wait.

Q59. Explain "hold and wait."
Ans: A process holds at least one resource while waiting for others.

Q60. How to prevent hold-and-wait?
Ans: Require processes to request all resources before execution begins.

Q61. What is deadlock prevention?
Ans: Ensuring at least one of the four necessary conditions never holds.

Q62. What is deadlock avoidance?
Ans: Dynamically examine resource allocation state to ensure it never enters an unsafe state (e.g., Banker's algorithm).

Q63. Explain Banker's algorithm.
Ans: Before allocating, check if system remains in safe state (all processes can finish in some order).

Q64. What is a safe state? Unsafe state?
Ans: Safe: there exists a safe sequence. Unsafe: may lead to deadlock.

Q65. What is deadlock detection?
Ans: Allow deadlock to occur; use algorithms (e.g., resource allocation graph) to detect and recover.

Q66. How does resource allocation graph detect deadlock?
Ans: Cycle in graph with single-instance resources → deadlock.

Q67. What is deadlock recovery? Methods?
Ans: Abort processes or preempt resources. Methods: kill all deadlocked, kill one at a time, rollback to checkpoint.

Q68. Why is Banker's algorithm rarely used in real systems?
Ans: Requires fixed number of processes/resources; assumes known max demand—unrealistic in dynamic environments.

Q69. Can deadlock occur with reusable resources only?
Ans: Yes—e.g., memory, files, printers.

Q70. Can two processes deadlock with one resource?
Ans: No—circular wait requires at least two resources.

Q71. How does Dining Philosophers illustrate deadlock?
Ans: All pick up left fork simultaneously → each waits for right fork → circular wait → deadlock.

Unit 5: Memory Management (5 Hours)

Q72. What are memory management requirements?
Ans: Relocation, protection, sharing, logical organization, physical organization.

Q73. What is static vs dynamic loading?
Ans: Static: load entire program at start. Dynamic: load parts on demand → saves memory.

Q74. Explain fixed and variable partitioning.
Ans: Fixed: memory divided into fixed partitions (internal fragmentation). Variable: partitions created per process (external fragmentation).

Q75. What is internal fragmentation?
Ans: Wasted space within allocated memory block (e.g., in paging).

Q76. What is external fragmentation?
Ans: Free memory exists but not in contiguous blocks (e.g., in segmentation). Solved by compaction or paging.

Q77. What is paging?
Ans: Memory divided into fixed-size frames; program into pages. Enables non-contiguous allocation.

Q78. How is logical address converted to physical in paging?
Ans: Logical = `page# + offset`; use page table to get frame#; physical = `frame# + offset`.

Q79. What is a page table?
Ans: Maps page numbers to frame numbers in physical memory.

Q80. Explain segmentation.
Ans: Memory divided into variable-size segments (code, data, stack). Logical address = `segment# + offset`.

Q81. Compare paging and segmentation.
Ans: Paging: fixed size, no external frag, internal frag. Segmentation: logical units, external frag, no internal frag.

Q82. What is virtual memory?
Ans: Technique to execute processes larger than physical memory using disk as extension.

Q83. How is virtual memory implemented?
Ans: Via demand paging—pages loaded only when referenced.

Q84. What is a page fault?
Ans: Occurs when referenced page is not in RAM; OS loads it from disk.

Q85. Explain FIFO page replacement.
Ans: Replace oldest loaded page. Suffers from Belady's anomaly.

Q86. What is Belady's anomaly?
Ans: Increasing page frames may increase page faults (in FIFO).

Q87. Explain LRU.
Ans: Replace least recently used page. Approximates optimal but costly to implement.

Q88. What is Optimal page replacement?
Ans: Replace page not used for longest future time. Theoretical minimum faults; not implementable (needs future knowledge).

Q89. What is TLB? Why used?
Ans: Translation Lookaside Buffer—cache for recent page table entries to speed up address translation.

Q90. What is thrashing? How to prevent?
Ans: High page fault rate due to insufficient frames. Prevent via working-set model or page fault frequency control.

Q91. Explain placement strategies: First Fit, Best Fit, Worst Fit.
Ans: First Fit: allocate first hole size. Best Fit: smallest sufficient hole. Worst Fit: largest hole. Best Fit → small unusable holes; Worst Fit → less fragmentation.

Q92. Why is paging preferred over segmentation in modern OS?
Ans: Simpler hardware, no external fragmentation, easier swapping.

Unit 6: I/O and File Management (5 Hours)

Q93. Why is I/O management complex?
Ans: Devices vary in speed, data rate, protocol; require buffering, error handling, and scheduling.

Q94. What is I/O buffering? Why used?
Ans: Temporary storage to absorb speed mismatches between CPU and devices.

Q95. Types of I/O buffering?
Ans: Single, double, circular buffering.

Q96. What are disk scheduling algorithms?
Ans: Methods to order disk read/write requests to minimize seek time.

Q97. Explain FCFS disk scheduling.
Ans: Serve requests in arrival order. Simple but inefficient.

Q98. What is SSTF? Drawback?
Ans: Shortest Seek Time First—serve nearest request. Drawback: starvation of distant requests.

Q99. Explain SCAN algorithm.
Ans: Disk arm moves in one direction, serving all requests; reverses at end.

Q100. What is C-SCAN?
Ans: Circular SCAN—arm moves to end, then jumps to start without servicing—uniform wait time.

Q101. Compare SCAN and C-SCAN.
Ans: SCAN has lower average seek time; C-SCAN gives more uniform performance.

Q102. What is a file?
Ans: Logical collection of related data with a name, stored on secondary storage.

Q103. What is a file system?
Ans: Mechanism to store, organize, and retrieve files (e.g., ext4, NTFS).

Q104. What is file organization? Types?
Ans: How records are stored: sequential, indexed, hashed.

Q105. What is sequential access? Direct access?
Ans: Sequential: read in order. Direct: access any record via offset/key.

Q106. What is a directory?
Ans: File containing metadata (names, attributes, locations) of other files.

Q107. Explain single-level, two-level, tree-structured directories.
Ans: Single: all files in one dir. Two-level: user + file. Tree: hierarchical (modern OS).

Q108. What is file sharing? Problems?
Ans: Multiple users access same file. Problems: consistency, access control, concurrency.

Q109. What is record blocking?
Ans: Storing multiple records in one disk block to reduce I/O operations.

Q110. What is secondary storage management?
Ans: OS functions: disk formatting, free-space management (bit vector, linked list), allocation (contiguous, linked, indexed).

Q111. Compare contiguous, linked, and indexed allocation.
Ans: Contiguous: fast access, external frag. Linked: no frag, slow access. Indexed: supports direct access, overhead of index block.

Q112. What is FAT?
Ans: File Allocation Table—linked allocation with table in memory for fast access (used in MS-DOS).

Q113. What is inode?
Ans: Data structure in Unix storing file metadata and block pointers.

Q114. What is spooling?
Ans: Simultaneous Peripheral Operations On-Line—buffering jobs (e.g., print queue) for devices slower than CPU.

Q115. How does OS handle I/O errors?
Ans: Retry, report to user, log error, or terminate process.

Q116. What is device independence?
Ans: Programs access devices via logical names (e.g., "printer")—OS maps to physical device.

PRACTICALS, AI ASSIGNMENTS, AND PROJECTS

Q117. How do you create a zombie process in C?
Ans: Parent calls `fork()`, child exits, parent sleeps without calling `wait()`.

Q118. How to avoid zombie processes?
Ans: Parent must call `wait()` or `waitpid()`. Alternatively, use double fork.

Q119. Write a shell script to list files with .txt extension.
Ans: `ls *.txt`

Q120. How to implement Banker's algorithm in C?
Ans: Input: available, max, allocation → compute need → check safe sequence using work and finish arrays.

Q121. How to simulate FIFO page replacement?
Ans: Use queue—on page fault, if full, dequeue oldest and enqueue new.

Q122. What ML model can predict best CPU scheduler?
Ans: Decision tree or random forest trained on features like burst time, arrival pattern, I/O ratio.

Q123. How can LSTM predict page faults?
Ans: Train on page access sequences; LSTM learns temporal patterns to forecast next access.

Q124. What data is needed for deadlock prediction model?
Ans: Resource allocation state, process requests, hold patterns over time.

Q125. What is the role of system logs in anomaly detection?
Ans: Logs reveal unusual sequences (e.g., repeated failed logins)—autoencoders detect deviations.

Q126. What are key components of your multiprogramming OS project?
Ans: CPU simulator, PCB, scheduler, memory manager (paging), interrupt handler, IPC.

Q127. How is virtual memory implemented in your project?
Ans: Demand paging with page table, page fault handler, and disk swap space simulation.

Q128. What is inter-process communication (IPC)? Methods?
Ans: Mechanisms for processes to exchange data: pipes, message queues, shared memory, sockets.

Created by Anshul