1. What is an operating system, and what are its primary functions?

An Operating System (OS) is system software that acts as an intermediary between users and computer hardware.

Primary Functions:

- Process Management – Creates, schedules, and terminates processes.
- Memory Management – Allocates and manages memory efficiently.
- File System Management – Manages files and directories.
- I/O Device Management – Controls peripheral devices.
- Security & Access Control – Protects against unauthorized access.

---

2. Explain the difference between a process and a thread.

- Process: Independent execution unit with its own memory space.
- Thread: A lightweight unit within a process that shares memory and resources.

Comparison:

Feature	Process	Thread
Memory	Has its own memory space	Shares memory within a process
Context Switching	Slow	Fast
Communication	Uses IPC (Inter-Process Communication)	Direct memory access
Creation	Expensive	Cheaper

---

3. What is virtual memory, and how does it work?

Virtual memory is a technique that provides the illusion of more memory than physically available by using disk space (swap space).

- Paging – Divides memory into fixed-size pages and swaps them between RAM and disk.
- Segmentation – Divides memory logically into variable-sized segments.

---

4. Describe the difference between multiprogramming, multitasking, and multiprocessing.

Feature	Multiprogramming	Multitasking	Multiprocessing
Definition	Multiple programs in memory	Multiple tasks running	Multiple CPUs
Execution	Only one process at a time	Rapid switching of tasks	True parallel execution
Goal	Maximize CPU utilization	Increase responsiveness	Improve performance

---

5. What is a file system, and what are its components?

A file system organizes and manages data storage.

Components:

- Files – Data storage units.
- Directories – Organizational structures.
- File Allocation Table (FAT) / Inode Table – Stores metadata.
- File Access Permissions – Controls access.

Types of File Systems: FAT32, NTFS, EXT4, HFS+

---

6. What is a deadlock, and how can it be prevented?

A deadlock occurs when processes hold resources and wait indefinitely for each other.

Deadlock Prevention Strategies:

- Avoid Hold and Wait – Request all resources at once.
- Circular Wait Prevention – Impose an ordering rule.
- Deadlock Detection & Recovery – Detect cycles and terminate processes.

---

7. Explain the difference between a kernel and a shell.

- Kernel: The core of the OS that interacts with hardware.
- Shell: The interface that users interact with to execute commands.

Types of Shells:

- Command-Line Shell (e.g., Bash, PowerShell)
- Graphical Shell (e.g., Windows Explorer)

---

8. What is CPU scheduling, and why is it important?

CPU Scheduling determines which process gets CPU time to optimize performance.

Scheduling Algorithms:

- FCFS (First-Come-First-Serve)
- SJF (Shortest Job First)
- Round Robin
- Priority Scheduling

---

9. How does a system call work?

A system call allows user programs to request services from the OS kernel.

Examples:

- open(), read(), write() – File operations
- fork(), exec() – Process creation
- kill() – Process termination

---

10. What is the purpose of device drivers in an operating system?

Device drivers allow the OS to communicate with hardware (printers, keyboards, etc.).

- Kernel-mode drivers – Run at a low level, directly interacting with hardware.
- User-mode drivers – Operate in user space for safer execution.

---

11. Explain the role of the page table in virtual memory management.

A page table maps virtual addresses to physical memory locations.

- Page Table Entry (PTE): Stores frame number, valid bit, access rights.
- Translation Lookaside Buffer (TLB): Caches page table entries for faster access.

---

12. What is thrashing, and how can it be avoided?

Thrashing occurs when excessive paging degrades system performance.

Solutions:

- Increase RAM.
- Implement Working Set Model (only keep frequently used pages in memory).
- Use Page Fault Frequency control.

---

13. Describe the concept of a semaphore and its use in synchronization.

A semaphore is a signaling mechanism used to manage process synchronization.

Types:

- Binary Semaphore (Mutex): Values 0 or 1, used for mutual exclusion.
- Counting Semaphore: Allows multiple processes to access resources.

---

14. How does an operating system handle process synchronization?

Using synchronization mechanisms like:

- Semaphores – Prevent race conditions.
- Monitors – Encapsulate shared resources.
- Locks & Mutexes – Ensure atomicity.

---

15. What is the purpose of an interrupt in operating systems?

Interrupts temporarily halt CPU execution to handle urgent tasks like I/O operations.

Types:

- Hardware Interrupts – Triggered by external devices.
- Software Interrupts – Generated by programs (e.g., system calls).

---

16. Explain the concept of a file descriptor.

A file descriptor is an integer that represents an open file in Unix-like systems.

File Descriptor	Standard Stream
0	Standard Input (stdin)
1	Standard Output (stdout)
2	Standard Error (stderr)

---

17. How does a system recover from a system crash?

- File System Journaling – Logs changes before applying them.
- Checkpointing – Saves system state periodically.
- Process Rollback – Restores the last stable state.

---

18. Describe the difference between a monolithic kernel and a microkernel.

Feature	Monolithic Kernel	Microkernel
Structure	Single large program	Minimal core, user-space modules
Performance	Fast	Slower due to IPC overhead
Stability	Less stable	More stable

---

19. What is demand paging, and how does it improve memory management efficiency?

Demand paging loads only the required memory pages, reducing memory usage.

Advantages:

- Saves RAM.
- Reduces I/O overhead.
- Faster process loading.

---

20. Explain the concept of a race condition and how it can be prevented.

A race condition occurs when multiple processes access shared data simultaneously, leading to inconsistent results.

Prevention Methods:

- Locks & Mutexes – Ensure exclusive access.
- Atomic Operations – Prevent interruptions.
- Critical Sections – Restrict concurrent access.

21. Explain the difference between preemptive and non-preemptive scheduling.

- Preemptive Scheduling: The OS can interrupt and switch a process before it completes (e.g., Round Robin, Priority Scheduling).
- Non-Preemptive Scheduling: Once a process starts execution, it cannot be interrupted until it finishes (e.g., FCFS, SJF).

---

22. What is round-robin scheduling, and how does it work?

Round-Robin (RR) is a preemptive scheduling algorithm where each process gets a fixed time quantum for execution.

- If the process is not completed within the time quantum, it is moved to the end of the queue.
- It ensures fair CPU time allocation and is used in time-sharing systems.

---

23. Describe the priority scheduling algorithm. How is priority assigned to processes?

- Each process is assigned a priority value; the scheduler selects the highest priority process.
- Two Types:
    - Preemptive Priority Scheduling: A higher-priority process interrupts a lower-priority one.
    - Non-Preemptive Priority Scheduling: A higher-priority process waits until the current process finishes.

Priority Assignment Methods:

- Static: Assigned at process creation.
- Dynamic: Adjusted based on execution time or resource needs.

---

24. What is the shortest job next (SJN) scheduling algorithm, and when is it used?

- SJN (or SJF – Shortest Job First) selects the process with the shortest CPU burst time.
- Used in: Batch systems where process execution times are known in advance.
- Problem: Can lead to starvation for longer jobs.

---

25. Explain the concept of multilevel queue scheduling.

- Divides processes into separate queues based on priority/type (e.g., system processes, interactive, batch).
- Each queue can have its own scheduling algorithm (e.g., RR for interactive, FCFS for batch).
- Ensures better process categorization and execution control.

---

26. What is a process control block (PCB), and what information does it contain?

A PCB stores process-related information required by the OS.

Contains:

- Process ID (PID)
- Process state (Ready, Running, Waiting)
- Program counter (Next instruction)
- CPU registers & scheduling information
- Memory management info

---

27. Describe the process state diagram and transitions between process states.

State	Transition	Description
New	Created → Ready	Process is created.
Ready	CPU assigned → Running	Waiting for CPU time.
Running	Time expired → Ready OR I/O needed → Waiting	Currently executing.
Waiting	I/O complete → Ready	Waiting for I/O or resource.
Terminated	Process completed	Process exits.

---

28. How does a process communicate with another process in an operating system?

Processes communicate using Inter-Process Communication (IPC) mechanisms:

- Pipes – Data transfer between related processes.
- Shared Memory – Common memory space for direct communication.
- Message Queues – Message-passing between processes.
- Sockets – Communication between processes across networks.

---

29. What is process synchronization, and why is it important?

- Ensures that multiple processes execute correctly when accessing shared resources.
- Avoids issues like race conditions, deadlocks, and inconsistency.
- Uses semaphores, mutexes, monitors, and locks.

---

30. Explain the concept of a zombie process and how it is created.

A zombie process occurs when:

- A child process terminates but its parent does not call wait() to collect its exit status.
- The child remains in the process table as a defunct (zombie) process.

Prevention: Use wait() or waitpid() in the parent process.

---

31. Describe the difference between internal fragmentation and external fragmentation.

- Internal Fragmentation: Wasted space inside allocated memory blocks (e.g., a 100-byte block used for 80 bytes).
- External Fragmentation: Free memory exists but is scattered in non-contiguous blocks.

Solution: Compaction or paging.

---

32. What is demand paging, and how does it improve memory management efficiency?

Demand paging loads pages into memory only when required, reducing memory usage.

- Uses Page Fault Handling to fetch missing pages.
- Advantages: Reduces memory wastage, speeds up execution.

---

33. Explain the role of the page table in virtual memory management.

A page table maps virtual addresses to physical memory addresses.

Types:

- Single-level paging – Simple but slow.
- Multi-level paging – Reduces page table size.
- Inverted page table – Uses a hash table for efficient lookups.

---

34. How does a memory management unit (MMU) work?

The MMU:

- Converts virtual addresses to physical addresses.
- Uses TLB (Translation Lookaside Buffer) for fast lookups.
- Supports paging and segmentation for memory protection.

---

35. What is thrashing, and how can it be avoided in virtual memory systems?

- Thrashing occurs when too many page faults degrade performance.
- Prevented by:
    - Increasing RAM.
    - Adjusting the working set size.
    - Using page replacement algorithms like LRU or FIFO.

---

36. What is a system call, and how does it facilitate communication between user programs and the OS?

A system call is a request to the OS for privileged operations like:

- File management (open(), read(), write()).
- Process control (fork(), exec(), exit()).
- Memory management (mmap(), brk()).

---

37. How does an operating system handle I/O operations?

- Uses buffering, caching, and spooling.
- Employs interrupt-driven and DMA (Direct Memory Access) techniques.

---

38. Explain the concept of a race condition and how it can be prevented.

- A race condition occurs when multiple threads/processes access shared data simultaneously.
- Prevented using synchronization mechanisms (semaphores, mutexes, locks).

---

39. Describe the role of device drivers in an operating system.

- Acts as a translator between the OS and hardware.
- Handles I/O operations, interrupts, and device communication.

---

40. What is an orphan process? How does an OS handle orphan processes?

- A child process becomes orphaned if its parent terminates before it.
- Solution: The OS reassigns it to init (PID 1) to prevent resource leaks.

---

41. How does the fork() system call work in creating a new process in Unix-like operating systems?

- fork() creates a new child process that is an exact copy of the parent.
- The child receives a unique Process ID (PID).
- Both parent and child continue execution.

---

42. How does process termination occur in Unix-like operating systems?

- Normal exit (exit()).
- Signal-based termination (kill(pid, signal)).
- Parent termination – If the parent dies, the child becomes orphaned.

---

43. What is the role of the long-term scheduler in process scheduling?

- Controls the degree of multiprogramming by selecting processes from disk to be loaded into memory.

---

44. How does the short-term scheduler differ from the long-term scheduler?

Feature	Long-Term Scheduler	Short-Term Scheduler
Purpose	Decides which processes to load	Selects which process to run next
Execution Frequency	Infrequent	Frequent
Affects	Multiprogramming	CPU scheduling

45. Describe how a parent process can wait for a child process to finish execution.

A parent process can use the wait() or waitpid() system calls to wait for its child process to complete.

- wait(): Blocks the parent until any child process terminates.
- waitpid(pid, &status, options): Allows waiting for a specific child process.
- Prevents zombie processes by releasing resources used by the child.

---

46. What is the significance of the exit status of a child process in the wait() system call?

- The exit status informs the parent process why the child terminated.
- It is stored in an integer variable passed to wait() or waitpid().
- Example:
    - exit(0): Normal termination.
    - exit(1): Error occurred.
    - Termination by signal (e.g., segmentation fault).

---

47. How can a parent process terminate a child process in Unix-like operating systems?

- Using the kill(pid, signal) system call:
    - kill(child_pid, SIGTERM): Graceful termination.
    - kill(child_pid, SIGKILL): Forceful termination.
- The parent can also terminate all child processes when it exits.

---

48. Explain the difference between a process group and a session in Unix-like operating systems.

- Process Group: A collection of related processes that can receive the same signals together.
- Session: A set of process groups managed by a controlling terminal.
- Example:
    - A shell runs a process (session leader).
    - A session may contain multiple process groups (foreground and background jobs).

---

49. Describe how the exec() family of functions is used to replace the current process image with a new one.

- The exec() functions replace the current process image with a new program.
- Common variants:
    - execl(), execp(), execv(), etc.
- Example Usage:

c

CopyEdit

execl("/bin/ls", "ls", "-l", NULL);

    - This replaces the running process with /bin/ls.

---

50. What is the purpose of the waitpid() system call in process management? How does it differ from wait()?

- waitpid() allows a parent to wait for a specific child process, while wait() waits for any child.
- Syntax:

c

CopyEdit

pid_t waitpid(pid, &status, options);

- Differences:
    - waitpid(pid, NULL, 0): Waits for the specific process.
    - waitpid(-1, NULL, 0): Acts like wait(), waits for any child.

---

51. How does process termination occur in Unix-like operating systems?

A process terminates when:

1. Normal exit (exit()).
2. Signal-based termination (kill(pid, SIGTERM)).
3. Fatal error (e.g., segmentation fault).
4. Parent process termination (orphaned processes get adopted by init).

---

52. What is the role of the long-term scheduler in the process scheduling hierarchy? How does it influence the degree of multiprogramming in an operating system?

- Long-Term Scheduler (Job Scheduler) controls the number of processes in memory.
- Determines the degree of multiprogramming (how many processes are in RAM).
- Helps balance CPU-bound and I/O-bound processes for efficient execution.

---

53. How does the short-term scheduler differ from the long-term and medium-term schedulers in terms of frequency of execution and the scope of its decisions?

Feature	Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
Purpose	Loads processes into memory	Schedules processes for CPU execution	Swaps processes in/out of memory
Execution Frequency	Infrequent	Very Frequent	Occasional
Affects	Degree of multiprogramming	CPU scheduling	Memory management

---

54. Describe a scenario where the medium-term scheduler would be invoked and explain how it helps manage system resources more efficiently.

- The medium-term scheduler temporarily removes processes from RAM (swapping).
- Scenario:
    - If too many processes cause thrashing, the scheduler swaps out some processes to free memory.
    - Once resources are available, swapped-out processes are brought back into memory.
- Goal: Improve system performance by optimizing CPU and memory usage.