---

1. Process Management

Concept: A **Process** is a program in execution. A **Thread** is a lightweight unit of a process.

- **Deep Dive:**
    - **Process Control Block (PCB):** Data structure storing process info (ID, State, Registers).
    - **Context Switching:** Saving the state of the current process and loading the next one. *Cost:* Pure overhead (CPU does no useful work during this).
    - **Process States:** New $\to$ Ready $\to$ Running $\to$ Terminated (or Blocked/Wait).

The `fork()` System Call (Crucial for C-CAT)

- **Explanation:** Creates a duplicate process. The child gets a copy of the parent's data/code.
- **Return Values:**
    - Returns `0` to the **Child**.
    - Returns `Child_PID` (>0) to the **Parent**.
    - Returns `<0` if it fails.

> 💡 **TRICK:** Counting Processes
>
> - **Total Processes:** 2^n (where n is number of `fork()` calls).
> - **Total Child Processes:** $2^n - 1$.
> - Example:
> c fork(); // 2 processes (1 parent, 1 child) fork(); // 4 processes (2 parents, 2 children) fork(); // 8 processes (4 parents, 4 children)

---

2. CPU Scheduling

Concept: Deciding which process runs on the CPU next.

Metrics:

- **Arrival Time (AT):** When process enters system.
- **Burst Time (BT):** Time required to execute.
- **Turnaround Time (TAT):** Completion Time - Arrival Time.
- **Waiting Time (WT):** TAT - Burst Time.

Important Algorithms:

1. **FCFS (First Come First Serve):**
    - *Type:* Non-preemptive.
    - *Issue:* **Convoy Effect** (Short process stuck behind a long one).

2. **SJF (Shortest Job First):**
    - *Type:* Can be Preemptive (SRTF) or Non-preemptive.
    - *Advantage:* Gives **Minimum Average Waiting Time** (Optimal).
    - *Issue:* **Starvation** (Long jobs never run).
3. **Round Robin (RR):**
    - *Type:* Preemptive (uses **Time Quantum**).
    - *Best for:* Time-sharing systems (Response time).
    - *Trick:* If Time Quantum $\to \infty$, RR becomes FCFS.

💡 TRICK: Gantt Chart Shortcut

Don't calculate mentally. Draw a horizontal bar (Gantt Chart) immediately.

- Write time on the bottom axis ($0, 2, 5...$).
- Cross out processes as they finish.

---

3. Deadlock

Concept: A situation where a set of processes are blocked because each is holding a resource and waiting for another resource held by someone else.

The 4 Necessary Conditions (Coffman Conditions):

(All 4 must happen for Deadlock)

1. **Mutual Exclusion:** Only one process can use a resource at a time.
2. **Hold and Wait:** A process holding at least one resource is waiting for others.
3. **No Preemption:** Resources cannot be forcibly taken away.
4. **Circular Wait:** P1 waits for P2, P2 waits for P3... Pn waits for P1.

Handling Deadlock:

- **Prevention:** Break one of the 4 conditions.
- **Avoidance: Banker's Algorithm**. (Check if system is in a "Safe State").
- **Detection:** Wait for deadlock, then kill processes.

💡 **TRICK:** Banker's Algorithm

- **Need Matrix = Max - Allocation**.
- If `Need <= Available`, the process can run.
- Once it finishes, `Available = Available + Allocation` (It returns resources).

---

4. Synchronization

Concept: Coordinating processes to avoid data inconsistency.

- **Race Condition:** When output depends on the order of execution (bad!).
- **Critical Section (CS):** Part of code where shared resources are accessed.
- **Semaphores:**
    - **Wait (P):** Decrements value. If <0, block.
    - **Signal (V):** Increments value. Wakes up a process.
    - **Binary Semaphore (Mutex):** Value is 0 or 1. Used for locks.
    - **Counting Semaphore:** Value can be >1. Used for managing multiple instances of a resource (e.g., 3 printers).

---

5. Memory Management

Concept: Managing RAM efficiently.

Key Definitions:

- **Fragmentation:** Wasted space.
    - *Internal:* Wasted space *inside* a block (Fixed partitioning).
    - *External:* Wasted space scattered *outside* (Dynamic partitioning).
- **Paging:** Divides memory into fixed-size "Frames" and process into "Pages".
    - *Solves:* External Fragmentation.
    - *Hardware:* **MMU** (Memory Management Unit) maps Logical Address $\to$ Physical Address.

Virtual Memory:

- Allows running programs **larger than RAM**.
- **Page Fault:** CPU asks for a page $\to$ Not in RAM $\to$ OS fetches from Disk.
- **Thrashing:** High paging activity (CPU is busy swapping pages, not working). *Cause:* Degree of Multi-programming is too high (too many processes).

> 💡 **TRICK:** Address Calculation
>
> - **Logical Address** = Page Number + Offset.
> - **Physical Address** = Frame Number + Offset.
> - *Formula:* `Size of Page Table` = (Total Logical Address Space) / (Page Size).

---

6. Important Linux Commands & Permissions

File Permissions (`chmod`):

- **Read (r) = 4**, **Write (w) = 2**, **Execute (x) = 1**.
- **User Types:** Owner (u), Group (g), Others (o).
- **Example:** `chmod 751 file`

    - **7** (Owner) = $4+2+1$ (rwx).
    - **5** (Group) = $4+0+1$ (r-x).
    - **1** (Others) = $0+0+1$ (--x).

Must-Know Commands:

- `grep pattern file`: Search for text.
- `ls -l`: List files with details (permissions, size).
- `ps`: Display current processes.
- `kill PID`: Terminate a process.
- `top`: Real-time view of system (CPU/RAM usage).

Quick Recap for Exam Day

1. **Starvation** is solved by **Aging**.
2. **Threads** share Heap/Data/Code but have **own Stack/Registers**.
3. **Banker's Algorithm** is for **Avoidance**.
4. **SJF** is optimal for minimum waiting time.
5. **Kernel Mode** allows execution of privileged instructions; **User Mode** does not.

Would you like me to create a **visual comparison table** for "Process vs. Thread" or "Paging vs. Segmentation" to help you memorize the differences?

Final "Exam Day" Checklist for OS

If you can answer these 3 questions in your head right now, you are done with OS.

1. **Math:** Can you calculate the number of processes for `fork(); fork(); fork();`? *(Ans: 8)*
2. **Logic:** Do you know that **SJF** is the best for average waiting time but causes starvation? *(Ans: Yes)*
3. **Command:** Do you know `chmod 777` means everyone has full permission? *(Ans: Yes)*