UNIT 3: CPU SCHEDULING & DEADLOCK - DETAILED EXPLANATION OF ALL BULLET POINTS

3.1 CPU SCHEDULING CRITERIA (DETAILED EXPLANATION)

• CPU Utilization (Keep CPU as busy as possible)

Definition: CPU utilization measures the percentage of time the CPU is actively executing instructions rather than sitting idle.

Detailed Explanation:

- **Goal**: Maximize CPU usage to ensure hardware resources are not wasted
- **Range**: 0% (completely idle) to 100% (fully utilized)
- **Real-world scenarios**:
  - Light load system: 40% utilization is considered good
  - Heavy load system: 90% utilization is acceptable
  - 100% utilization indicates system is overloaded
- **Why important**: CPU is the most expensive component; keeping it busy improves system efficiency
- **Measurement**: (Total CPU busy time / Total time) × 100

• Throughput (Number of processes completed per unit time)

Definition: Throughput measures the number of processes that complete their execution in a given time period.

Detailed Explanation:

- **Units**: Processes per second, minute, or hour
- **Factors affecting throughput**:
  - Process mix (CPU-bound vs I/O-bound processes)
  - Scheduling algorithm efficiency
  - System load
  - Hardware capabilities

- **Example**: If 10 processes complete in 5 seconds, throughput = 2 processes/second
- **Trade-off**: High throughput might increase individual process waiting time
- **Goal**: Maximize number of completed processes

• Turnaround Time (Time from submission to completion)

Definition: Total time taken from when a process is submitted until it completes execution.

Detailed Explanation:

- **Formula**: Turnaround Time = Completion Time - Arrival Time
- **Components included**:
  - Waiting time in ready queue
  - CPU execution time
  - I/O wait time
  - Context switching overhead
- **Example**: Process arrives at 0ms, completes at 25ms → Turnaround time = 25ms
- **User perspective**: This is what users actually experience
- **Goal**: Minimize average turnaround time

• Waiting Time (Time spent in ready queue)

Definition: Total time a process spends waiting in the ready queue before getting CPU access.

Detailed Explanation:

- **Formula**: Waiting Time = Turnaround Time - Burst Time
- **What it measures**: Scheduling algorithm efficiency
- **Does NOT include**:
  - CPU execution time
  - I/O waiting time
  - Time spent in other queues
- **Example**: If turnaround time = 25ms and burst time = 10ms → Waiting time = 15ms
- **Critical metric**: High waiting time indicates poor scheduling

- **Goal**: Minimize average waiting time

• Response Time (Time to start responding after submission)

Definition: Time from when a process is submitted until it produces its first response.

Detailed Explanation:

- **Formula**: Response Time = First Response Time - Arrival Time
- **Importance**: Critical for interactive systems
- **Different from turnaround time**: Measures first response, not completion
- **Example**: Process submitted at 0ms, first response at 3ms → Response time = 3ms
- **User experience**: Users notice response time more than turnaround time
- **Goal**: Minimize response time for interactive systems

• Fairness (Equal chance for all processes)

Definition: Ensuring all processes get equitable access to CPU resources without discrimination.

Detailed Explanation:

- **Starvation prevention**: No process should wait indefinitely
- **Equal opportunity**: All processes should eventually get CPU time
- **Priority considerations**: Balance between fairness and priority requirements
- **Aging mechanism**: Gradually increase priority of waiting processes
- **Trade-offs**: Absolute fairness might reduce system efficiency
- **Goal**: Prevent starvation while maintaining system performance

3.2 DEADLOCK CONDITIONS (COFFMAN'S CONDITIONS) - DETAILED EXPLANATION

1. • Mutual Exclusion (Only one process can use a resource at a time)

Definition: At least one resource must be held in a non-sharable mode; only one process at a time can use the resource.

Detailed Explanation:

- **Inherent property**: Some resources are naturally non-sharable
- **Examples of non-sharable resources**:
  - **Printer**: Only one process can print at a time
  - **Tape drive**: Physical limitation prevents sharing
  - **Write access to files**: Data integrity requires exclusive access
  - **Critical sections**: Code segments requiring atomic execution
- **Sharable resources** (don't cause mutual exclusion):
  - **Read-only files**: Multiple processes can read simultaneously
  - **Memory pages**: Can be shared if read-only
- **Why this condition is necessary**: If all resources were sharable, no deadlock could occur
- **Real-world analogy**: Only one person can use a bathroom at a time

2. • Hold and Wait (A process is holding some resources and waiting for others)

Definition: A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

Detailed Explanation:

- **Scenario**: Process P1 holds Resource A and requests Resource B
- **Problem**: While waiting for B, P1 doesn't release A
- **Examples**:
  - **Database systems**: Transaction holds lock on Table A, requests lock on Table B
  - **File operations**: Process has file1 open, tries to open file2
  - **Memory management**: Process holds memory pages, requests more pages
- **Why problematic**: Creates resource dependencies that can form cycles
- **Real-world analogy**: Person holding elevator button while waiting for someone else
- **Alternative approach**: Request all resources at once (prevents this condition)

3. • No Preemption (Resources cannot be forcibly taken from processes)

Definition: Resources cannot be preempted; a resource can be released only voluntarily by the process holding it, after that process has completed its task.

Detailed Explanation:

- **Voluntary release only**: Process must explicitly release resources
- **Examples of non-preemptive resources**:
  - **Printer**: Cannot take away mid-job (would corrupt output)
  - **Database locks**: Taking away could cause data inconsistency
  - **File handles**: Forcible removal could corrupt files
- **Examples of preemptive resources**:
  - **CPU time**: Can be taken away by scheduler
  - **Memory pages**: Can be swapped to disk
- **Why preemption is difficult**:
  - **State preservation**: Hard to save and restore resource state
  - **Data integrity**: May corrupt ongoing operations
  - **Recovery complexity**: Difficult to resume from interrupted state
- **Real-world analogy**: Cannot take away someone's meal while they're eating

4. • Circular Wait (Closed chain of processes, each waiting for a resource held by the next)

Definition: A set of waiting processes {P0, P1, ..., Pn} exists such that P0 waits for a resource held by P1, P1 waits for a resource held by P2, ..., and Pn waits for a resource held by P0.

Detailed Explanation:

- **Chain formation**: P0 → R1 → P1 → R2 → P2 → R0 → P0
- **Cycle completion**: The chain must form a complete circle
- **Example scenario**:
  - Process A holds Printer, wants Scanner
  - Process B holds Scanner, wants Disk
  - Process C holds Disk, wants Printer
  - **Result**: A→Scanner→B→Disk→C→Printer→A (circular wait)
- **Graph representation**: Can be visualized using Resource Allocation Graph
- **Detection**: Cycle detection algorithms can identify circular waits

- **Real-world analogy**: Traffic jam at roundabout where each car blocks the next

3.3 DEADLOCK HANDLING STRATEGIES - DETAILED EXPLANATION

(a) DEADLOCK PREVENTION (Prevent at least one of the four conditions)

• Mutual Exclusion (Make resources sharable - not always possible)

Detailed Explanation:

- **Approach**: Convert non-sharable resources to sharable ones
- **Techniques**:
  - **Spooling**: Multiple processes can "write" to printer simultaneously by writing to disk
  - **Read-only access**: Make resources read-only when possible
  - **Resource virtualization**: Create virtual instances of resources
- **Limitations**:
  - **Inherently non-sharable resources:** Printers, tape drives cannot be truly shared
  - **Data integrity**: Some operations require exclusive access
  - **Hardware constraints**: Physical devices have sharing limitations
- **Example**: Instead of direct printer access, use print spooler queue

• Hold and Wait (Require processes to request all resources at once)

Detailed Explanation:

- **Method 1 - All at once**: Process must request all needed resources before execution
  - **Advantage**: No holding while waiting
  - **Disadvantage**: May not know all resources needed in advance
- **Method 2 - Release and re-acquire**: Process releases all resources before requesting new ones
  - **Advantage**: No holding while waiting
  - **Disadvantage**: May lose work progress
- **Problems with this approach**:
  - **Low resource utilization**: Resources held but not immediately used

- **Starvation possible**: Process needing many resources may never get all at once
- **Unpredictability**: Hard to predict all future resource needs

- **Example**: Database transaction requests all table locks before starting

• No Preemption (Allow preemption - take away resources)

Detailed Explanation:

- **Method 1 - Voluntary preemption**: If process cannot get needed resource, release all held resources
- **Method 2 - Forced preemption**: OS forcibly takes resources from processes
- **Preemption techniques**:
  - **Save state**: Store current state before taking resource
  - **Resume later**: Restore state when resource becomes available
- **Applicable to**:
  - **CPU**: Already preemptive in most systems
  - **Memory**: Pages can be swapped to disk
- **Not applicable to**:
  - **Printers**: Cannot interrupt print job
  - **Database locks**: May cause inconsistency
- **Implementation challenges**: State saving and restoration complexity

• Circular Wait (Impose ordering of resources - request in increasing order)

Detailed Explanation:

- **Resource numbering**: Assign unique numbers to all resource types
- **Ordering rule**: Processes must request resources in increasing order of numbers
- **Example numbering**:
  - Tape drive = 1
  - Disk = 5
  - Printer = 12
- **How it prevents circular wait**:

- If P1 has resource i and wants resource j, then i < j
- If P2 has resource j and wants resource k, then j < k
- Cannot form cycle because numbers always increase

- **Disadvantages**:
  - **Inconvenient programming**: May not match natural resource usage order
  - **Resource utilization**: May hold resources longer than needed
- **Real implementation**: Many systems use this approach for kernel resources

(b) DEADLOCK AVOIDANCE

• Requires advance knowledge of resource requirements

Detailed Explanation:

- **Advance declaration**: Processes must declare maximum resource needs before execution
- **Safe state maintenance**: System only grants resources if it can guarantee completion
- **Resource reservation**: System reserves enough resources to complete all processes
- **Limitations**:
  - **Static analysis**: Hard to predict dynamic resource needs
  - **Conservative approach**: May deny safe requests to maintain safety
  - **Overhead**: Requires complex algorithms and bookkeeping

• Banker's Algorithm is commonly used

Detailed Explanation:

- **Banking analogy**: Like bank ensuring it can satisfy all customers' credit needs
- **Key data structures**:
  - **Available**: Currently available resources
  - **Max**: Maximum demand of each process
  - **Allocation**: Currently allocated resources
  - **Need**: Remaining resource requirements (Max - Allocation)
- **Safety algorithm**:

    - Find process whose needs can be satisfied with available resources
    - Assume it completes and releases all resources
    - Repeat until all processes complete or deadlock detected
- **Request algorithm**:
    - Check if request ≤ need
    - Check if request ≤ available
    - Temporarily grant request
    - Run safety algorithm
    - If safe, grant permanently; otherwise, deny
- **Advantages**: Guarantees deadlock-free execution
- **Disadvantages**: Conservative, requires advance knowledge, high overhead

(c) DEADLOCK DETECTION AND RECOVERY

• Allow deadlock to occur, then detect using resource allocation graphs or algorithms

Detection Methods:

Resource Allocation Graph (Single instance per resource type):

- **Nodes**: Processes (circles) and Resources (rectangles)
- **Edges**: Request edges (process → resource) and Assignment edges (resource → process)
- **Cycle detection**: If cycle exists, deadlock present
- **Algorithm**: Use DFS to detect cycles

Wait-for Graph (Simplified version):

- **Nodes**: Only processes
- **Edges**: Process Pi → Process Pj (Pi waits for resource held by Pj)
- **Cycle detection**: Simpler than resource allocation graph

Detection Algorithm (Multiple instances):

- Similar to Banker's algorithm but works backward
- **Available**: Currently available resources

- **Allocation**: Currently allocated resources
- **Request**: Current resource requests
- **Algorithm**:
    a. Mark processes that can complete with available resources
    b. When process completes, add its allocation to available
    c. Repeat until no more processes can be marked
    d. Unmarked processes are in deadlock

• Recovery Methods

Kill a process (Abort one or more processes)

Detailed Explanation:

- **Victim selection criteria**:
    - **Priority**: Lower priority processes killed first
    - **Computation time**: Kill processes that have used less CPU time
    - **Resources held**: Kill processes holding many resources
    - **Progress**: Kill processes that have made less progress
- **Termination strategies**:
    - **Abort all deadlocked processes**: Simple but expensive
    - **Abort one at a time**: Check if deadlock resolved after each termination
- **Consequences**:
    - **Work loss**: Partial computations lost
    - **Cascade effect**: Other processes may be affected
    - **Starvation**: Same process repeatedly selected

Preempt resources and give them to others

Detailed Explanation:

- **Victim selection**: Choose process to preempt resources from
- **Rollback strategies**:

    - **Total rollback**: Abort process and restart from beginning
    - **Partial rollback**: Roll back to safe state (requires checkpoints)
- **Starvation prevention**:
    - **Cost factors**: Include number of rollbacks in selection criteria
    - **Time limits**: Limit how often same process can be victim
- **Implementation challenges**:
    - **State saving**: Must save process state before preemption
    - **Rollback complexity**: Determining safe rollback points
    - **Atomicity**: Ensuring operations are atomic during rollback

(d) IGNORE DEADLOCK (OSTRICH APPROACH)

• Many OS (e.g., Windows, UNIX) ignore deadlock because it occurs rarely

Detailed Explanation:

- **Philosophy**: "If deadlocks occur rarely, ignore the problem"
- **Reasoning**:
    - **Cost vs. benefit**: Prevention/avoidance overhead > deadlock cost
    - **Frequency**: Deadlocks are rare in well-designed systems
    - **User intervention**: Users can manually resolve deadlocks
- **When used**:
    - **Personal computers**: Single-user systems with manual recovery
    - **Non-critical systems**: Where occasional hang is acceptable
- **Manual recovery methods**:
    - **Process termination**: Kill processes using task manager
    - **System restart**: Reboot system to clear deadlock
    - **Resource release**: Manually close applications to free resources
- **Disadvantages**:
    - **System hangs**: May require manual intervention

    - **Work loss**: Unsaved work may be lost
    - **User frustration**: Unpredictable system behavior
- **Not suitable for**:
    - **Critical systems**: Medical, aerospace, financial systems
    - **Server systems**: Where availability is crucial
    - **Real-time systems**: Where deadlock cannot be tolerated

SUMMARY OF BULLET POINTS

Scheduling Criteria (6 points)

1. CPU Utilization - Keep hardware busy
2. Throughput - Maximize completed processes
3. Turnaround Time - Total time from submission to completion
4. Waiting Time - Time spent waiting for CPU
5. Response Time - Time to first response
6. Fairness - Equal opportunity for all processes

Deadlock Conditions (4 points - all must be present)

1. Mutual Exclusion - Non-sharable resources
2. Hold and Wait - Holding resources while waiting for others
3. No Preemption - Cannot forcibly take resources
4. Circular Wait - Circular chain of resource dependencies

Deadlock Prevention (4 strategies - eliminate one condition)

1. Make resources sharable (eliminate mutual exclusion)
2. Request all resources at once (eliminate hold and wait)
3. Allow resource preemption (eliminate no preemption)
4. Order resources and request in sequence (eliminate circular wait)

Deadlock Detection Methods (2 approaches)

1. Resource allocation graphs for single-instance resources
2. Detection algorithms similar to Banker's algorithm for multiple instances

Recovery Methods (2 approaches)

1. Process termination - Kill deadlocked processes
2. Resource preemption - Take resources and rollback processes

Each bullet point represents a fundamental concept that you must understand thoroughly for your exams. The detailed explanations above provide the depth needed to write comprehensive answers.