CS Exit Ethiopia

Revision Outlines

Computer Science Exit Examination

1 selected course | 18 November 2026

Selected courses

01. Operating System

https://csexitethiopia.com

abelasnake2003@gmail.com | Telegram: @Abechu7

Course contents

Each course begins on a new page. PDF bookmarks provide question, study-note section and revision-topic navigation.

Operating System 3

Operating System

Revision Outlines

Objectives and resource management | Exam focus

- OS objectives and services
  - CPU, memory, storage allocation
  - Device and communication resources
  - Convenience: interfaces, files, device abstractions
  - Efficiency: effective resource use
  - Efficiency ≠ every process's fastest response
  - Development tools use OS services
  - Application development ≠ core resource management
- Evolution and related properties
  - Maintainable interfaces/modules
  - Hardware support, features, repairs
  - Backward compatibility: supported older behavior
  - Compatibility ≠ every old binary
  - Scalability: greater workload/resources
  - Portability: other platforms
  - Fault tolerance: correct service despite failures
  - Recovery: service restoration afterward
  - Updates/monitoring: administrative help
  - Automatic updates: not universally defining

Batch, sharing and real time | Exam focus

- Execution organization
  - Batch: little post-submission interaction
  - Multiprogramming: CPU work during another job's wait
  - Several available jobs
  - Multiprogramming ≠ fixed time slices
  - Time-sharing: rapid scheduling/preemption, interaction
  - Single CPU: interleaved execution

- Multiple CPUs: true parallelism
- Shared memory ≠ mandatory private memories

• System categories and timing
- Single/multiuser: concurrent users, not applications
- Network systems: remote services
- Distributed systems: coordinated machines
- Location hiding possible
- Real-time correctness: timing included
- Hard deadlines: mandatory
- Soft misses: degraded service
- Real time ≠ simply fast
- Spooling/multitasking: mechanisms, not generation proofs

Kernel, shell, privilege and boot | Exam focus

• OS components and privilege
- Kernel: resources, protection, system calls
- Shell: command interpretation
- GUI: another interface
- Browsers, office tools, editors: applications
- Drivers: commonly kernel, sometimes user-space
- Kernel mode: privileged operations
- Supervisor/kernel mode: often synonyms
- Exclusive mode options: ambiguity

• Boot sequence and resident components
- Firmware/bootloader before kernel
- Kernel loaded into RAM
- First core OS component
- Kernel ≠ first power-on program
- Loadable modules: running-kernel driver additions
- Modular systems allow runtime loading
- Resident kernel: session-long activity
- Antivirus/disk tools: utilities
- User processes plus privileged components

APIs, system calls and virtualization | Exam focus

- System call versus library API
  - System call: protected kernel-service request
  - Process creation, file read, socket opening
  - API/library wrappers
  - Library function ≠ mandatory kernel entry
  - Application API versus user/kernel boundary
  - OS-service interface: scope required
- Services and virtual machines
  - Process control
  - File/device management
  - Information retrieval and communication
  - Video editing: application operation
  - write_database(): application-specific, not generic syscall
  - Virtualization: virtual machine environment
  - Hypervisor: guests on one host
  - Virtual memory: process-address translation
  - Kernel/shell/API/VM: distinct labels

Processes, programs and context

- Program versus process
  - Program: stored instructions
  - Process: executing instance
  - Address space and resources
  - Process ≠ universal application manager/UI
  - PCB: identity and state
  - Saved PC/registers: resume execution
  - Scheduling information
  - Memory/accounting details
  - I/O and open-resource state
- Context switch and task terminology
  - Save one context, restore another
  - Application-work interruption: overhead

- Interrupt/syscall: privilege-only switch possible
- Privilege switch ≠ process switch
- Task: OS-specific meaning
- Process/thread/schedulable entity
- Process-versus-task: no universal distinction

Process states, queues and schedulers

• Process-state transitions
- New: created/admitted
- Ready: eligible, not running
- Running: CPU in use
- Waiting/blocked: event required
- Terminated: execution ended
- Running→ready: quantum expiry/preemption
- Running→waiting: blocking I/O
- I/O completion: waiting→ready
- Completion ≠ immediate running guarantee

• Queues and scheduling roles
- Ready queue: eligible work
- Device queues: blocked requests
- Job queue: system processes
- Admission queue: awaiting memory/admission
- Long-term: job admission
- Medium-term: suspension/swapping
- Short-term: ready-work selection
- Dispatcher: handoff/context restore
- Priority preemption: policy-dependent running→ready

Creation, fork/exec and termination

• Process creation mechanisms
- Initialization, user requests, service requests
- Running process's creation call
- POSIX fork(): child creation

- Child return: 0
- Parent return: child PID
- Success assumption
- Distinct address spaces
- Explicit resource sharing possible
- exec: current image replacement
- exec ≠ new process creation

• Termination categories
- Voluntary: normal completion/program-detected error
- Involuntary: unrecoverable fault/killing
- exit(): normal runtime termination
- abort(): abnormal termination request
- Nonexistent/protected memory: fatal fault
- Bad-input exit: application decision
- Terminate exit: imprecise category
- Parent exit ≠ universal descendant killing

Threads and mapping models

• Thread ownership and sharing
- Own PC, registers, stack
- Shared code and address space
- Usually shared open files
- Process: potentially many threads
- Thread: one process at a time
- Threads ≠ group of processes
- Thread execution need not be sequential
- Independent versus cooperating processes
- Cooperation: communication/shared data

• Thread management and mappings
- User-level: library management
- Kernel threads: OS-known
- Many-to-one: blocking call blocks all
- One context: no simultaneous multi-CPU use

- One-to-one: another thread runs when blocked
- Many-to-many: users multiplexed over kernel threads
- Speed/reliability: implementation-dependent
- Management layer ≠ guarantees
- Browser/Excel names ≠ threading-model proof

IPC and blocking semantics

- IPC mechanisms
  - Shared memory: common region
  - Conflicting updates require synchronization
  - Message passing: send/receive
  - Pipes: streams
  - Message queues: message boundaries
  - Sockets: endpoints
  - RPC: remote operation
  - HTTP: application communication protocol
  - HTTP can carry IPC
- Blocking and nonblocking contracts
  - Blocking send: API-defined wait condition
  - Handoff, buffering or receipt
  - Blocking receive: wait for message
  - Nonblocking: prompt data/progress/would-block
  - Rendezvous: sender waits until receipt
  - File locks: access coordination
  - Locks ≠ arbitrary payload transfer
  - Locking: IPC synchronization
  - Asynchronous messaging ≠ synchronization-free

Races and critical-section correctness

- Race and critical-section criteria
  - Race: conflicting uncoordinated interleavings
  - Critical section: protected shared-state code
  - Critical section ≠ resource itself

- Mutual exclusion: safety property
- Progress
- Bounded waiting
- Stated correctness assumptions
- Failures: races, starvation, deadlock

• Synchronization structure
- Small critical regions
- Same protection for every conflicting access
- Barrier: defined participants reach point
- Monitor: encapsulated protected operations
- Condition synchronization
- Inconsistent lock order: deadlock risk

• Worked example Lost-update example
- Two reads: counter=0
- Each computes/stores 1
- Two increments → 1, not 2
- Entire read-modify-write under one lock

Semaphores and resource counts

• Semaphore operations and models
- Atomic wait/P/down
- Atomic signal/V/up
- Nonnegative wait: consume token or block
- Signal: token supply, possible wakeup
- Negative counts: alternative waiter encoding
- State the count model
- Binary semaphore: one permit
- Counting semaphore: several permits
- Mutex: ownership rules
- General semaphore: no equivalent ownership requirement

• Worked example Semaphore and bounded-buffer example
- Mutex initial value: 1
- wait(m); critical work; signal(m)

- N slots: empty=N, full=0, mutex=1
- Producer: wait empty, then mutex
- Insert; unlock; signal full
- Holding mutex while waiting empty
- Consumer unable to free slot

Turn variables and atomic primitives

• Strict alternation and lock-variable races
- turn=0/1
- Process i: wait while turn≠i
- Enter, then hand turn over
- Alternation enforces turns
- Outside peer: progress violation
- Separate test/set: race
- Both may observe free

• Atomic synchronization primitives
- Test-and-set; compare-and-swap
- Atomic check/update
- Ordinary check-then-write: non-atomic
- Peterson: flags plus turn
- Two processes; memory-model assumptions
- Semaphore: more than two possible
- Local interrupt disabling excludes no other CPUs
- Not general user-space locking
- Interrupts: event notification/synchronization
- Interrupt synchronization exclusions: overbroad

Scheduling metrics and calculations | Exam focus

• Scheduling equations and goals
- A: arrival; S: start; C: completion
- Response=S−A
- Turnaround=C−A
- Single burst B, no I/O

- Waiting=C−A−B
- I/O: subtract CPU and blocked time
- Ready-queue waiting only
- Throughput=completed_jobs/elapsed_time
- CPU_utilization=busy_time/elapsed_time
- Goal-dependent response/turnaround/wait minimization
- Useful throughput/utilization maximization

• Timeline-first calculation method
- Arrivals and idle gaps
- Preemptions and overhead
- Per-process values before averages
- First response ≠ final completion

• Worked example Scheduling-metric example
- A=2; S=5; B=4; C=9
- Uninterrupted CPU burst
- Response=3; turnaround=7; waiting=3
- Six completions/two hours=3 jobs/hour

Preemptive and non-preemptive decisions | Exam focus

• CPU scheduling variants
- Non-preemptive: completion/block/yield ends burst
- Preemptive: running→ready allowed
- Timer quantum or urgent arrival
- FCFS/non-preemptive SJF: non-preemptive
- RR/SRTF: preemptive
- Priority: either variant
- Cooperative: voluntary release

• CPU versus resource preemption
- Save context, reallocate CPU time
- Occupied printer/lock: unsafe withdrawal possible
- Time-slicing ≠ removing no-resource-preemption
- Frequent switches: overhead
- Non-preemptive long bursts delay interaction

FCFS and convoy effects | Exam focus

- FCFS behavior and limits
  - Ready bursts: arrival order
  - Ordinarily non-preemptive
  - Idle CPU: jump to next arrival
  - Convoy: long burst delays short jobs
  - Finite work ahead: eventual start
  - Eventual service ≠ small waiting
  - FCFS ≠ universally worst turnaround
- Worked example FCFS simultaneous-arrival example
  - All arrive 0; P1=24,P2=3,P3=3
  - Tie order: P1,P2,P3
  - P1:0-24; P2:24-27; P3:27-30
  - Waiting 0,24,27 → mean 17
  - Turnaround 24,27,30 → mean 27
  - Different arrivals: recompute actual times

SJF/SJN and remaining time | Exam focus

- Shortest-job scheduling rules
  - SJF/SJN: shortest ready next burst
  - Non-preemptive selection
  - SRTF: reconsider arrivals
  - Shortest remaining work; newcomer may preempt
  - Future bursts: estimates needed
  - SJF minimum-average-wait theorem
  - Fixed simultaneous ready set
  - Known bursts; no preemption/switch cost
  - Theorem ≠ universal preemptive-policy comparison
- Shortest-job traps
  - Repeated short arrivals: long-job starvation
  - Original burst ≠ remaining burst
- Worked example SRTF example
  - P1: arrival 0, burst 8

- P2: arrival 1, burst 2
- P1:0-1; P2:1-3; P1:3-10
- Waiting: P1=2, P2=0
- Non-preemptive SJF: no P1 interruption

Priority, starvation and aging | Exam focus

- Priority scheduling assumptions
  - Highest-priority ready job
  - Numeric convention: smaller or larger wins
  - Arrival preemption: specify variant
  - Continued high-priority load: low-priority starvation
  - SJF: long-job starvation too
  - Priority/SJF options: multiple valid answers
- Aging and priority inversion
  - Aging: waiting raises effective priority
  - Explicit aging rule
  - Workload/priority scheme conditions
  - Reduced indefinite postponement
  - Aging ≠ deadlock cure
  - Inversion: urgent waits on low-priority holder
  - Inheritance: temporarily raise holder priority

Round Robin and quantum bounds | Exam focus

- Round Robin queue behavior
  - Cyclic ready queue
  - At most q per turn
  - Unfinished→tail; completed/blocked→leave
  - Quantum-boundary arrival ordering
  - Large q approaches FCFS
  - Tiny q increases switch overhead
  - Recurring opportunities ≠ equal total consumption
- RR next-turn waiting bound
  - n continuously ready jobs

- No priority bypass; zero overhead
- Next-turn wait≤(n−1)q
- One interval, not lifetime waiting
- Unrestricted arrivals/blocking: bound limitations
- Include specified context-switch costs

• Worked example RR trace example
- P1=5,P2=3; arrivals=0; q=2
- P1:0-2; P2:2-4; P1:4-6
- P2:6-7; P1:7-8
- Completions: P1=8,P2=7
- Waiting: 8−5=3; 7−3=4
- Mean waiting=3.5

Multilevel queues and feedback | Exam focus

• Multilevel scheduling
- Separate foreground interactive/background batch classes
- Between-queue and within-queue selection
- Fixed interqueue priority: lower-queue starvation
- Class time shares: starvation avoidance
- Feedback queues: behavior-based movement
- CPU-intensive demotion
- Periodic priority boosting

• Multilevel trace assumptions and traps
- Plain multilevel ≠ automatic movement
- Initial class
- Quantum per level
- Promotion/demotion rules
- Interqueue policy
- Binary search tree: implementation data structure
- Tree ≠ CPU selection policy
- Infinite: not standard policy category

Fair-share and proportional guarantees | Exam focus

- Fair-share allocation
  - Stated weights
  - Per-user/group versus per-process shares
  - More processes under RR: aggregate advantage
  - Proportional shares: policy choice
  - Fair ≠ necessarily equal
- Lottery, stride and guarantee limits
  - Lottery: tickets and winning selection
  - Weights 3:1 → expected 75%:25%
  - Short-run variation
  - Lottery ≠ deterministic starvation deadline
  - Stride: deterministic share-progress tracking
  - Guaranteed scheduling: specify guarantee/assumptions
  - Finite FCFS/RR: no indefinite postponement possible
- Worked example User-share example
  - A: four ready tasks; B: one
  - Per-process RR: A≈80% overall
  - Per-user halves: A=50%,B=50%
  - Each A task≈12.5%
  - B’s task=50%

Evaluating scheduling claims | Exam focus

- Workload-dependent scheduling comparisons
  - Response, waiting, fairness, throughput tradeoffs
  - Workload and objective first
  - Apply algorithm; compare measured results
  - Burst times alone: insufficient maximum wait
  - Unconstrained arrivals, priorities, overhead
  - Finite workload versus open stream
  - Open stream: potential starvation
- Scheduling-question ambiguity
  - FCFS and SJN: both non-preemptive

- Equal share: process/user/group scope
- Highest turnaround: concrete workload required
- FCFS worst: not absolute
- RR: conventional interactive choice
- Turns improve responsiveness
- Quantum and overhead conditions

Deadlock and necessary conditions | Exam focus

- Deadlock versus blocking/starvation
  - Every member waits on blocked peer
  - Ordinary holder wait: normal blocking
  - Starvation: unfair indefinite delay
  - Starvation need not contain dependency cycle
  - Mutual exclusion: one nonshareable-resource holder
  - Hold-and-wait: retain while requesting
  - No resource preemption: voluntary release only
  - Circular wait: closed dependency chain
- Four-condition interpretation
  - Standard reusable-resource model
  - All four necessary
  - General policies ≠ current deadlock proof
  - Inspect actual allocations and requests
  - Prevent one condition: prevent class
  - CPU preemption can coexist with deadlocks
- Worked example Two-lock deadlock example
  - P1 holds A, requests B
  - P2 holds B, requests A
  - No release before acquisition
  - Nonshareable locks
  - Retained holdings
  - No forced release
  - Circular dependency

Resource graphs and cycle tests | Exam focus

- Resource-allocation graph rules
  - P→R: request
  - R-instance→P: assignment
  - Follow directed waits
  - No cycle → no deadlock
  - Single instance/type: cycle → deadlock
  - Multiple instances: cycle insufficient
  - Available instance may permit completion
  - Wait-for graph: resource nodes replaced
  - P→Q: P waits for Q
- Single-instance cycle example
  - P0 holds R0, requests R1
  - P1 holds R1, requests R0
  - Wait-for edges: P0→P1, P1→P0
  - Cycle: neither proceeds
  - Multiple instances: inspect every assignment
  - Check available instances before conclusion

Prevention and removed conditions | Exam focus

- Deadlock prevention mechanisms
  - Constrain resource requests
  - Safe sharing removes mutual exclusion
  - All-before-holding requests remove hold-and-wait
  - Safe withdrawal/restart attacks no-preemption
  - Global acquisition order removes circular wait
  - Physical resources: sharing/preemption limitations
- Release-and-retry interpretation
  - Release all holdings when waiting
  - Removes simultaneous holding and waiting
  - Voluntary release versus forced preemption
  - Reclaiming description: ambiguous condition target
  - Prevention: utilization waste possible

- Prevention: starvation possible
- Deadlock freedom ≠ fairness

Banker avoidance and safety testing | Exam focus

- Banker safety algorithm
  - Avoidance: resulting state remains safe
  - Known maximum claims required
  - Need=Max−Allocation
  - Work=Available
  - Find unfinished process: Need≤Work componentwise
  - Simulate completion: Work+=Allocation
  - Finish all → safe sequence
  - Request≤Need and request≤Available
  - Tentative grant requires successful safety test
  - Unsafe ≠ already deadlocked
  - Banker: avoidance, not scheduling/detection/prevention
- Worked example One-resource safety example
  - Available=1
  - P1: allocation=1, remaining need=1
  - P2: allocation=2, remaining need=2
  - Finish P1 → Work=2
  - Then finish P2
  - Safe sequence: P1,P2
  - Maximum claims known before testing

Detection, recovery and ignoring | Exam focus

- Detection and recovery
  - Detection: current-state deadlock
  - Terminate selected processes
  - Reclaim safely preemptable resources
  - Roll back to checkpoint
  - Termination/rollback: both recovery methods
  - Recovery-choice questions require criterion

- Repeated victim selection: starvation
- Detection timing and intentional ignoring
  - Every-request versus periodic checking
  - Overhead versus detection-delay tradeoff
  - No universal best interval
  - Ostrich: ignore sufficiently rare problem
  - Real systems: selected prevention/detection
  - Deadlock acceptance: intentional ignoring possible
  - Detection graph versus avoidance safety graph
  - Scheduling queue: different problem

Address binding, loading and linking

- Address-binding stages
  - Compile-time: known placement, absolute addresses
  - Load-time: relocation on placement
  - Execution-time: runtime translation
  - Movement requires hardware support
  - Logical/virtual: process-generated addresses
  - Physical: RAM addresses
  - MMU: logical→physical
  - Reverse-direction bank stem: malformed
- Loading, linking and swapping
  - Dynamic loading: routine on demand
  - Dynamic linking: external references load/runtime
  - Suitable library pages shared
  - Static linking: executable code copies
  - Traditional swapping: whole process→backing store
  - Paging: individual pages
  - Context switch: execution-state change

Contiguous allocation and fit policies

- Contiguous base/limit translation
  - One continuous process region

- Base: process physical start
- Limit: permitted length
- Check 0≤logical_offset<limit
- Physical=base+offset
- Relocation register ≠ normally OS base
- Last-valid-offset limit: changed inequality
- Specify limit convention

• Hole-selection policies
- First-fit: first adequate hole
- Next-fit: resume previous search
- Best-fit: smallest adequate hole
- Worst-fit: largest hole
- Search time versus future utilization
- No universally optimal policy
- Binary-fit: not standard listed policy

• Worked example Fit and boundary examples
- Holes 100,500,200,300; request 180
- First-fit=500; best-fit=200; worst-fit=500
- Base=4000; length=100
- Offset 99→4099
- Offset 100: trap

Internal and external fragmentation

• Fragmentation and consolidation
- Internal: unused allocated-unit space
- External: free total split inadequately
- Contiguous request cannot fit holes
- Coalescing: adjacent free holes joined
- Compaction: move allocated regions
- Consolidate free space
- Compaction requires relocation support

• Allocation-model fragmentation limits
- Fixed-frame paging: no external frame fragmentation

- Final page: possible internal waste
- Variable segmentation/contiguous partitions: external fragmentation
- Paging OS ≠ every allocator fragmentation-free
- Track units; round allocations upward
- Requests need not fill blocks

• Worked example Fragmentation examples
- 10 KiB object; 4 KiB pages
- ceil(10/4)=3 pages
- Allocate 12 KiB; waste 2 KiB
- Two 3 KiB holes: total 6 KiB
- Contiguous 5 KiB request fails

Paging, page tables and TLB

• Paging translation
- Equal fixed page/frame size P
- Page=floor(v/P)
- Offset=v mod P
- Page table: frame plus protection/status
- Physical=frame×P+offset
- TLB: translation cache
- TLB miss: possible table walk
- TLB miss ≠ nonresident page

• Page-table sizing and validation
- Virtual-page count and entry size
- Table size ≠ physical RAM alone
- Check protection and present bits
- Table lookup ≠ CPU scheduling

• Worked example Paging calculation example
- P=1024; v=2053
- Page=2; offset=5
- Frame=7 → physical=7173
- 16-bit virtual space; 1 KiB pages
- 64 virtual pages

- Page number: 6 bits

Segmentation versus paging

- Segment translation
  - Variable logical units: code/stack/arrays
  - Segment table: base/limit/permissions
  - Address pair: segment,offset
  - Validate segment and bound
  - Physical=base+offset
  - Out-of-range offset: trap
  - No spill into next segment
- Basic paging/segmentation contrasts
  - Paging: fixed units, page tables
  - Paging: internal waste
  - Segmentation: variable units, segment tables
  - Segmentation: external fragmentation
  - Segments reflect program structure
  - Ordinary pages: largely application-transparent
  - A/B/C contrasts all true
  - Multiple true contrasts: no unique choice
- Worked example Segment boundary example
  - Segment 1: base=8000, length=500
  - Offset 120→8120
  - Offset 500: invalid
  - Length-based limit convention

Virtual memory and page faults

- Virtual-memory purpose and fault sequence
  - Translated, protected address spaces
  - Nonresident portions: storage backing
  - No added physical RAM
  - Capacity constraints remain
  - Demand paging: load on access

- Nonresident access: trap
- Validate mapping/permissions
- Find frame or select victim
- Save dirty victim if needed
- Obtain contents; update mapping; restart

• Fault exceptions and thrashing
- Page fault ≠ mandatory disk read
- Zero-filled pages; shared mappings
- Illegal access: reject, not load
- Excess active-page demand: thrashing
- Repeated faults dominate computation
- Reduce active load
- Add suitable memory

• Worked example Page-fault cost example
- Access: 100 ns
- Additional page-in: 5 ms
- Fault probability: 10^−4
- Added delay≈500 ns/access
- Rare long delays matter

Page-replacement traces

• Replacement policy rules
- Victim needed: no suitable free frame
- FIFO: earliest loaded page
- FIFO hit: insertion time unchanged
- LRU: least recently accessed
- LRU recency: every access
- Optimal: farthest future use
- Optimal: offline lower bound
- LFU: reference counts
- LFU tie/aging rules required

• Replacement trace method and properties
- One row per reference

- Identify hits before eviction
- Initial loads count as faults
- Belady’s anomaly: more frames, more faults
- FIFO may exhibit anomaly
- LRU/optimal: stack property
- Page buffering: useful free/modified pages retained
- Reduced effective fault cost

• Worked example FIFO versus LRU example
- Three empty frames
- References: 1,2,3,1,4,1
- FIFO: faults 1,2,3,4,last 1
- FIFO total: five faults
- LRU: refresh 1 on hit
- Evict 2 for 4
- Last 1: hit
- LRU total: four faults

Controllers, drivers and I/O services

• I/O hardware and software roles
- Controller: hardware status/control/data registers
- Driver: OS requests→device operations
- Device independence and uniform naming
- Hide brand-specific details
- Error handling: supported reports/retries
- Polling: repeated readiness checks
- Interrupts: CPU attention requests
- DMA: setup then block transfer
- No per-byte CPU work

• I/O responsibility boundaries
- Kernel/driver answer: abstraction scope required
- Plug-and-play: compatible hardware discovery/configuration
- Driver may still be needed
- Input/output/storage/NIC: managed hardware

- Accounts: identities
- Paper/network errors: report/handle
- Total power loss: hardware support required
- OS software alone insufficient

Buffering, caching, spooling and clocks

• Buffering versus caching versus spooling
- Buffer: temporary in-transit data
- Absorb speed/transfer-size differences
- Cache: copies for faster reuse
- Spool: queued device jobs
- Printer jobs: usually disk-backed
- Producers avoid continuous device ownership
- Combined implementation, different purposes

• Timers, monitoring and disk responsibilities
- Timer: timekeeping and timeouts
- Accounting and preemptive scheduling
- Monitor: CPU/memory/storage/network usage
- Measurement ≠ resource creation
- Disk initialization/filesystem creation: OS/tools
- Boot begins: firmware
- Bad-sector remapping: drive firmware possible
- OS-all-disk-functions claim: scope boundaries

Files, descriptors and operations

• File content, metadata and handles
- Metadata: name, size, owner
- Permissions and timestamps
- open: obtain reference
- read/write: byte transfer
- seek: offset change
- close: release reference
- truncate: size change

- unlink: name removal
- Descriptor: process-local integer handle
- Open-file state: offset/access mode
- Descriptor ≠ offset

• Read semantics and filesystem formats
- Seekable read: offset advances
- Short reads possible
- Zero read: normally EOF
- Sockets: ordinary read/write supported
- socket/connect/send: socket operations
- read ≠ uniquely non-network call
- FAT32/NTFS/ext4: filesystem formats
- HTTP: application communication protocol

Directories, mounts and links

• Directory organizations
- Single-level: unique names, one namespace
- Two-level: users separated
- Tree: nested directories
- Absolute and relative paths
- Links: shared graphs
- Modern systems: commonly hierarchical
- Feature label ≠ excluding simpler arrangements

• Links and filesystem attachment
- Hard link: same file, another name
- Usually within one filesystem
- Symlink: stored pathname
- Cross-filesystem targets; dangling links possible
- Unlink one name ≠ content deletion
- Remaining names/open references retain content
- Mount: attach filesystem at mount point
- Mount ≠ format
- NFS: remote filesystem access

- FTP: transfer, not itself mounting

File-block allocation methods

- Contiguous, extent and linked allocation
  - Contiguous: consecutive blocks
  - Logical block i→start+i
  - Direct access
  - Growth and external fragmentation problems
  - Extents: several contiguous runs
  - Linked: block chains
  - Growth without whole-file contiguous reservation
  - Block i: traverse preceding links
  - FAT: chains in table
  - Cached FAT ≠ direct arithmetic mapping
- Indexed allocation and inode lookup
  - Per-file pointer array/tree
  - Direct lookup; metadata overhead
  - Unix inode: direct and indirect pointers
  - Pointer size and block size
  - Index capacity
  - Metadata-read caching assumptions
- Worked example Index-capacity example
  - 1024-byte index; 4-byte pointers
  - 1024/4=256 block addresses
  - Data blocks: 1024 bytes
  - One level: 256 KiB data
  - Exclude metadata/reserved entries

Protection, authentication and threats

- Identity and access control
  - Authentication: verify identity
  - Authorization: permitted operations
  - ACL: principals and rights

- Owner/group/other: simpler bit model
- Owner full access: not universal
- Everyone read: not universal
- Read/write/execute/append/delete: distinct rights
- Filesystem-specific permission rules

• Security controls, threats and interfaces
- Least privilege: limited damage
- Password hashing: no stored plaintext
- One-time passwords: expire after use
- Encryption: confidentiality with correct keys
- Encryption ≠ automatic integrity
- Trojan: useful-software disguise
- Worm: independent propagation
- Virus: attached host
- DoS: availability harm
- Cryptography/authentication/firewalls: product-dependent support/configuration
- Interfaces: CLI, GUI, speech/natural language, VR
- Interface list: not closed

Linux, Android and history limits

• Kernel, distributions and product licensing
- Linux: Unix-compatible, written from scratch
- Linus Torvalds
- Distribution: kernel/userland/libraries/tools
- Android base kernel: Linux
- Framework/applications: higher layers
- AOSP: open source
- Devices: proprietary apps/services possible
- Windows/macOS: proprietary products
- macOS: open-source components too
- Both proprietary options: ambiguous answer
- Microsoft Office: application suite

• Product and historical-claim limits

- Windows: Microsoft, after MS-DOS desktop era
- Solaris: Sun origin
- First widely used: definition/evidence required
- First multitasking: definition/evidence required
- UNIX ≠ first ever multitasking
- Generation/function pairing: historical evidence
- Mobile majority: population/date required
- Architecture facts ≠ market-share proof

Repairing ambiguous bank questions

- Missing or multiple answer concepts
  - Requested concept omitted
  - Several valid examples: reject uniqueness
  - System call: kernel-service request
  - Process: executing program
  - CPU/memory/I/O: multiple managers
  - Device/process/file/security choices: missing umbrella
  - Virtual memory: isolation, translated space
  - Virtual memory ≠ unlimited memory
- Minimal wording repairs
  - Specify OS versus API
  - Scheduling variant
  - Arrivals and quantum
  - Exact process/resource graph
  - Base-limit convention
  - Comparison scope
  - Marked answer ≠ evidence
  - Missing edges: unrecoverable from theory