CS Exit Ethiopia

Complete Study Notes

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 6

How to use these study notes

START01 A practical study cycle

Start with a small set of questions before reading.

Mark the exact thing that stopped you: an unknown term, a rule, a calculation, a trace or a choice between alternatives.

Use the question lookup to open the relevant section.

Read one section, close it, and explain the rule in your own words.

Solve the question again and say why each other choice fails.

Then change one assumption or input and solve the variation.

This checks understanding beyond recognition.

A useful 60-minute session: 10 minutes of cold questions, 20 minutes of focused reading, 20 minutes of closed-book solving and 10 minutes updating an error log.

Adjust the timing to your needs.

Revisit mistakes after a gap, for example the next day, several days later and the following week.

These intervals are a practical starting plan, not a fixed learning law.

Record the section ID, your mistaken rule, the corrected rule and one boundary case.

Keep the log short enough to review.

Mix related courses during revision: programming with data structures; databases with security; networking with administration.

Explain how the same word changes meaning across contexts.

START02 What to emphasize

Emphasize definitions, key differences, algorithm costs, SDLC models, databases, networking and security.

Use short oral questions to check understanding and correct common misconceptions.

Combine those habits with the tracing, calculation, design and diagnosis demanded by the Ministry learning outcomes.

Allocate revision by both exam share and your error log.

A course with fewer items can still deserve more time if your foundations are weak.

The blueprint is a planning baseline, not a prediction of exact questions.

Definitions: distinguish nearby concepts, not just their names.

Calculations: write units, formulas, assumptions and intermediate steps.

Code and SQL: check validity before tracing the text; state any changes before analyzing a corrected version.

Algorithms and protocols: know the input, decision rule, state change, termination and failure cases.

Design choices: state the criterion.

A choice can change when cost, reliability, consistency or latency matters more.

Course group	Items in the 100-item blueprint	Revision implication
Web Programming	9	Reserve extra practice for HTML, CSS, JavaScript, PHP, client/server behavior and safe data handling.
Data Structures and Algorithms	7	Trace operations and compare their assumptions and costs.
Each of the other fourteen courses	6 each	Study all sixteen; do not abandon a whole course based on a short note or an alleged repeated-question list.

START03 How to handle exam traps

Underline NOT, FALSE, BEST and EXCEPT before comparing choices.

Check whether the stem supplies enough facts to select one answer.

A long scenario is not automatically an analysis question, and a familiar answer letter is not evidence.

When code is invalid, a diagram is absent, or two choices express the same rule, identify the clarification needed before answering.

Learn the correct rule and the smallest clarification needed.

Do not memorize an arbitrary key as a universal fact.

Distinguish worst case from average case, and an upper bound from a tight bound.

Check language version, SQL dialect, addressing assumptions, graph weights and tie-breaking rules.

For subnetting and scheduling, calculate rather than guess from a familiar number.

Use timed mixed papers only after you can explain your errors.

Revisit the reading sections behind recurring mistakes.

Operating System

Complete Study Notes

OS01 | Objectives, resource management and evolution

An OS allocates CPU time, memory, storage, devices and communication resources, and provides services to programs.

Convenience means making tasks easy through consistent interfaces, files and device abstractions.

Efficiency means using resources effectively; it is not synonymous with the fastest response for every process.

Application development tools use OS services but application development itself is not a core resource-management function.

Ability to evolve means adding hardware support, features and repairs through maintainable interfaces/modules.

Backward compatibility preserves supported older application behavior; it is not a promise that every old binary works.

Scalability means handling greater workload/resources; portability means adapting to other platforms.

Fault tolerance continues correct service despite specified failures; recovery restores service afterward.

Updates and monitoring help administration, but automatic updates are not a defining property of every OS.

Operating System module

Oracle Solaris 11

OS02 | Batch, multiprogramming, time-sharing and real time

Batch jobs run with little interaction after submission.

Multiprogramming keeps several jobs available so another can use the CPU while one waits; it does not require fixed time slices.

Time-sharing uses rapid scheduling/preemption to support interactive users.

A single CPU interleaves execution; multiple CPUs can execute work truly in parallel.

Shared-memory multiprocessors do not require each CPU to have private main memory.

Single-user and multiuser describe concurrent user access, not the number of applications.

Network systems expose remote services; distributed systems coordinate work across machines, sometimes hiding location.

Real-time correctness includes timing: hard deadlines must be met, while soft deadline misses degrade service. “Real time” does not simply mean fast.

Spooling and multitasking are mechanisms, not proof of a particular OS generation.

Operating System module

OS03 | Kernel, shell, privilege and boot

The kernel implements core resource management, protection and system-call handling.

A shell reads/interprets user commands; a GUI is another interface.

Browsers, office tools and editors are applications.

Device drivers are commonly kernel components, although user-space drivers also exist.

Kernel mode has privileged operations; supervisor mode often means the same privileged mode, so offering both as exclusive answers is ambiguous.

Firmware/bootloader stages execute before transferring control to a kernel loaded into RAM.

The kernel is the first core OS component, not the first program ever executed at power-on.

Loadable modules can add drivers to a running kernel; “kernel modules cannot be loaded while running” is false in modular systems.

A resident kernel remains active through the session.

Utilities such as antivirus and disk tools can combine user processes with privileged components.

Operating System module

OS04 | APIs, system calls and virtualization

A system call requests a protected kernel service, such as creating a process, reading a file or opening a socket.

A user-facing API/library may wrap that request; not every library function enters the kernel.

Distinguish the application API from the direct user/kernel boundary when describing access to OS services.

Process control, file/device management, information retrieval and communication are service categories.

Video editing and a database-specific write_database() are application operations, not standard generic syscall categories/names.

Virtualization presents a virtual machine environment, often allowing guest operating systems on one host through a hypervisor.

It differs from virtual memory, which translates a process’s addresses.

Kernel, shell, API and VM are not interchangeable labels.

Operating System module

OS05 | Programs, processes and PCB contents

A program is stored instructions; a process is an executing instance with address space and resources.

A process does not universally “manage applications” or “provide a user interface.” The PCB records identity, state, PC/registers, scheduling information, memory/accounting details and I/O/open-resource state.

A saved PC identifies where execution resumes.

A context switch saves one execution context and restores another; useful application work is interrupted by this overhead.

An interrupt or system call can switch privilege without switching process.

Task is OS-specific terminology: it may mean process, thread or schedulable entity.

No universal process-versus-task distinction follows from the word alone.

Operating System module

OS06 | Process states, queues and schedulers

New processes are created/admitted; ready means eligible for CPU but not currently running.

Running uses a CPU.

Waiting/blocked means unable to proceed until an event; terminated means execution has ended.

Running→ready follows time-slice expiry or another preemption.

Running→waiting follows blocking I/O. I/O completion normally gives waiting→ready, not a guaranteed immediate running state.

The ready queue contains eligible work; device queues hold blocked requests.

The broad textbook job queue contains processes in the system, while an admission queue more narrowly holds jobs awaiting memory/admission.

Long-term scheduling admits jobs; medium-term scheduling suspends/swaps them; short-term scheduling selects ready work.

The dispatcher performs the handoff/context restore after selection.

A bare “priority” causes running→ready only if a preemptive policy chooses another process.

Operating System module

UIC CPU Scheduling

OS07 | Creation, fork/exec and termination

Processes arise during initialization, user requests, service requests or a running process's creation call.

In POSIX-style systems fork() creates a child: zero is returned in the child and the child PID in the parent on success.

The two have distinct address spaces, although resources may be shared through explicit mechanisms. exec replaces the current process image; it does not itself create another process.

Normal completion and a program-detected error exit are voluntary; an unrecoverable fault or being killed is involuntary. exit() terminates normally through the runtime; abort() requests abnormal termination.

A reference to nonexistent/protected memory is a fatal fault, unlike an application deciding to exit after bad input. "Terminate exit" is not a precise standard category.

Parent termination does not universally kill all descendants.

Linux man-pages fork(2)

Linux man-pages execve(2)

OS08 | Threads and mapping models

A thread has its own PC, registers and stack.

Threads in a process share code, address space and usually open-file resources.

A process can contain many threads; a thread belongs to one process at a time.

Threads are not a group of processes and need not execute sequentially.

Independent processes can exist; cooperating ones communicate/share data.

User-level threads are managed by a library; kernel threads are known to the OS.

In many-to-one mapping, a blocking kernel call can block all user threads and one kernel execution context cannot use multiple CPUs simultaneously.

One-to-one permits another kernel thread to run when one blocks.

Many-to-many multiplexes user threads over several kernel threads.

Speed and reliability depend on implementation; the management layer alone guarantees neither.

Application names such as browser or Excel do not prove a particular version's threading model.

Operating System module

OS09 | IPC and blocking semantics

Shared memory lets processes access a common region and requires synchronization for conflicting updates.

Message passing exchanges data through send/receive operations.

Pipes provide streams; message queues preserve messages; sockets connect endpoints; RPC requests a remote operation.

HTTP is an application protocol over a communication mechanism and can carry interprocess communication, so it is not universally “not IPC.”

Blocking send waits for the required handoff/buffering/receipt condition defined by its API.

Blocking receive waits for a message.

Nonblocking operations return promptly with data, progress or a would-block result. “Sender blocked until received” identifies blocking send under rendezvous semantics.

File locks coordinate access rather than transfer arbitrary payloads, though locking is synchronization within IPC.

Do not equate asynchronous messaging with no synchronization ever.

Operating System module

OS10 | Races, critical sections and correctness

A race occurs when results depend on uncoordinated interleaving of conflicting accesses.

A critical section is code accessing shared state that needs protection, not the resource itself.

A correct exclusion solution requires mutual exclusion, progress and bounded waiting under stated assumptions.

Mutual exclusion is a desired safety property; races, starvation and deadlock are possible failures.

Keep critical regions small and ensure every conflicting access uses the same protection.

A barrier waits until a defined set of participants reaches a point; a monitor encapsulates protected operations with condition synchronization.

Locking can itself introduce deadlock if acquisition order is inconsistent.

Example

Worked microexample

- Two threads read counter=0, each computes 1 and each stores 1. Two increments produce 1 instead of 2. Protecting the entire read-modify-write with one lock prevents that lost update.

Operating System module

OS11 | Semaphores and resource-count traces

A semaphore provides atomic wait/P/down and signal/V/up operations.

Under the nonnegative-token model, wait consumes one available token or blocks when none is available; signal supplies a token and may wake a waiter.

Implementations using negative internal counts encode waiters differently, so state the model.

A binary semaphore represents one permit; a counting semaphore tracks several.

A mutex normally has ownership rules that a general semaphore lacks.

Example

Worked microexample

- Initialize a mutex semaphore to 1. Each thread does wait(m); critical work; signal(m). For a buffer of N slots use empty=N, full=0 and mutex=1: producer waits empty then mutex, inserts, unlocks then signals full. Waiting for empty while already holding mutex can prevent the consumer from freeing a slot.

Arpaci-Dusseau and Arpaci-Dusseau, OSTEP Semaphores

OS12 | Turn variables and atomic primitives

Strict alternation uses turn=0/1: process i waits while turn≠i, then enters and sets turn to the other process on leaving.

It enforces turns but can violate progress if the other process stays outside its critical region.

A naive lock-variable test followed by a separate set races because both processes can observe free before either sets it.

Test-and-set and compare-and-swap combine checking/updating atomically; ordinary check-then-write does not.

Peterson’s classic algorithm uses flags plus turn for two processes under its memory-model assumptions; a semaphore can coordinate more than two.

Disabling local interrupts does not exclude other CPUs and is not a general user-space locking solution.

Interrupts can notify/synchronize events, so they cannot be universally excluded from every synchronization-related classification.

UIC Process Synchronization

OS13 | Scheduling metrics and calculation method

For arrival A, first start S and completion C, response=S−A and turnaround=C−A. For a single CPU burst B with no I/O, waiting=C−A−B. With I/O, subtract all CPU and blocked time to isolate ready-queue waiting.

Throughput=completed jobs/elapsed time; CPU utilization=busy time/elapsed time.

Minimize response/turnaround/waiting under the selected goal and maximize useful throughput/utilization.

First construct a time line with arrivals, idle gaps, preemptions and overhead, then calculate per-process values before averages.

Do not confuse first response with final completion.

Example

Worked microexample

- A job arrives at 2, starts at 5, uses 4 CPU units without interruption and finishes at 9: response=3, turnaround=7, waiting=3. Six completions in two hours means throughput=3 jobs/hour.

Operating System module

OS14 | Preemptive versus non-preemptive decisions

Non-preemptive scheduling keeps a running CPU burst until completion, blocking or voluntary yielding.

Preemptive scheduling can return running work to ready, usually after a timer quantum or a more urgent arrival.

FCFS and non-preemptive SJF are non-preemptive; RR and SRTF are preemptive.

Priority can be either, so its name alone cannot decide.

Cooperative describes relying on voluntary release.

CPU preemption saves a context and reallocates processing time; taking an occupied printer or lock may be unsafe.

This is why time-slicing does not automatically remove the no-resource-preemption condition for deadlock.

Frequent switches have cost; non-preemptive policies can delay interactive work behind long bursts.

Operating System module

UIC CPU Scheduling

OS15 | FCFS and the convoy effect

FCFS runs ready bursts in arrival order and ordinarily does not preempt.

If the CPU is idle, jump to the next arrival.

Long CPU-bound work can make many short jobs wait: the convoy effect.

A finite burst with finitely many jobs ahead eventually starts, but waiting can still be large.

FCFS does not universally have the worst turnaround for every possible workload.

Example

Worked microexample

- All ready at time 0, tie order P1=24,P2=3,P3=3: schedule 0-24 P1,24-27 P2,27-30 P3. Waiting times 0,24,27 average to 17. Turnarounds 24,27,30 average to 27. If arrivals are not simultaneous, recompute using their actual times.

Operating System module

OS16 | SJF/SJN and shortest remaining time

Non-preemptive SJF/SJN chooses the shortest next burst among currently ready jobs.

SRTF reconsiders on arrivals and selects the shortest remaining work; a shorter newcomer can preempt.

Future burst lengths normally need estimates.

SJF minimizes average waiting for the usual fixed simultaneous ready set with known bursts and no preemption/switch cost; that theorem is not a workload-independent comparison against every preemptive policy.

Repeated short arrivals can starve a long job.

The shortest total original burst is not necessarily the shortest remaining burst after execution has begun.

Example

Worked microexample

- P1 arrives 0 with burst 8; P2 arrives 1 with burst 2. SRTF runs P1 0-1, P2 1-3, P1 3-10. P1 waiting=2 and P2 waiting=0. Non-preemptive SJF cannot interrupt P1 once it has started.

Operating System module

OS17 | Priority, starvation and aging

Priority scheduling chooses the highest-priority ready job.

Check whether a smaller or larger numeric value means higher priority.

State whether a high-priority arrival preempts.

Fixed-priority scheduling can starve low-priority jobs under continued high-priority load.

SJF can also starve long jobs; starvation is not exclusive to fixed-priority scheduling.

Aging increases waiting jobs' effective priority according to a rule, reducing indefinite postponement when the priority scheme and workload permit.

It is not a deadlock cure.

Priority inversion occurs when urgent work waits on a resource held by lower-priority work; priority inheritance can temporarily raise the holder's priority.

Operating System module

OS18 | Round Robin, quantum and bounded turns

RR serves a ready queue cyclically for at most quantum q per turn.

Unfinished jobs return to the tail; completed/blocked jobs leave.

Specify how arrivals at a quantum boundary enter the queue.

A large quantum approaches FCFS; tiny quanta increase switch overhead.

RR gives recurring opportunities, not identical total CPU consumption for jobs of different lengths.

With n continuously ready jobs, no priority bypass and zero overhead, a just-requeued job waits at most (n−1)q for its next turn.

This bounds one interval, not total lifetime waiting when arrivals or blocking are unrestricted.

Include context-switch costs if given.

Example

Worked microexample

- P1=5,P2=3, both arrive 0, q=2: P1 0-2,P2 2-4,P1 4-6,P2 6-7,P1 7-8. Completions 8,7; waiting=8−5=3 and 7−3=4; average 3.5.

Operating System module

OS19 | Multilevel queues and feedback

A multilevel queue separates classes, such as foreground interactive and background batch, and chooses both between queues and within them.

Fixed interqueue priority may starve lower queues; allocating time shares between classes can avoid this.

A multilevel feedback queue allows movement according to observed behavior, commonly demoting CPU-intensive jobs and periodically boosting priority.

Do not infer movement from a simple multilevel-queue label.

State initial class, quantum per level, promotion/demotion rules and interqueue policy before tracing.

A binary search tree is a data structure that might implement a queue; it is not itself a CPU selection policy. "Infinite" is not a standard scheduling-policy category.

Operating System module

OS20 | Fair-share, proportional shares and guarantees

Fair-share scheduling allocates according to stated weights, potentially per user/group rather than per process.

Equal per-process RR can let a user obtain more aggregate CPU by launching more processes.

Proportional shares address that policy choice; fair need not mean equal.

Lottery scheduling assigns tickets and selects a winning ticket each turn.

With weights 3:1, expected shares are 75% and 25%; short-run results vary and this is not a deterministic no-starvation deadline.

Stride-style scheduling tracks progress toward shares deterministically.

Any “guaranteed scheduling” answer needs the actual guarantee and assumptions; finite FCFS and RR can also avoid indefinite postponement.

Example

Worked microexample

- If user A runs four ready tasks and B runs one, plain RR gives A about 80% overall. A policy granting each user half the CPU instead gives A’s tasks about 12.5% each and B’s task 50%.

Arpaci-Dusseau and Arpaci-Dusseau, OSTEP Proportional Share

Operating System module

OS21 | How to evaluate scheduling claims

There is no universally best policy for response, waiting, fairness and throughput simultaneously.

State the workload and objective, apply the algorithm and compare measured results.

Maximum waiting at submission is not determined from burst times alone if future arrivals, priorities or overhead are unconstrained.

Known finite workloads are different from an open stream with potential starvation.

An item asking for one non-preemptive algorithm while offering FCFS and SJN has multiple answers.

Equal share needs a process/user/group definition. “Highest average turnaround” requires a concrete workload; do not memorize FCFS as an absolute.

RR is a conventional interactive choice because turns improve responsiveness, subject to quantum and overhead.

Operating System module

OS22 | Deadlock and its four necessary conditions

A set is deadlocked when each member waits for an event/resource that only another blocked member can provide.

Mere waiting for another holder is normal blocking; indefinite unfair delay can be starvation without a cycle of dependencies.

Mutual exclusion allows at most one holder of a nonshareable resource.

Hold-and-wait means retaining a resource while requesting another.

No resource preemption means release must be voluntary.

Circular wait forms a closed chain of such waits.

All four are necessary in the standard reusable-resource model.

Their availability as general system policies does not prove a deadlock exists at this instant; inspect actual allocations/requests.

Preventing any one condition prevents that class of deadlock.

A preemptive CPU scheduler can coexist with deadlocked locks.

Example

Worked microexample

- P1 holds lock A and requests B; P2 holds B and requests A. Neither releases before acquiring the other: nonshareable locks, retained holdings, no forced release and a cycle all occur.

Operating System module

OS23 | Resource graphs and missing-diagram limits

In a resource-allocation graph, P→R is a request and an instance R→P is an assignment.

Follow these directions to form waits.

No cycle implies no deadlock; with one instance of every resource type, a cycle implies deadlock.

With multiple instances a cycle alone is insufficient: another instance may permit completion.

A wait-for graph replaces single-instance resource nodes with P→Q when P waits for Q.

For a complete single-instance example, suppose P0 holds R0 and requests R1, while P1 holds R1 and requests R0.

The wait-for edges P0→P1 and P1→P0 form a cycle, so neither can proceed.

With multiple resource instances, inspect all assignments and available instances before concluding that a cycle is a deadlock.

Operating System module

OS24 | Prevention: identify the condition removed

Prevention constrains how requests are made.

Share a resource where safe to remove mutual exclusion; request all needed resources before holding any to remove hold-and-wait; permit safe resource withdrawal/restart to attack no-preemption; impose a global acquisition order to remove circular wait.

Not every physical resource is shareable or safely preemptable.

“If waiting, release all held resources and try again” removes simultaneous holding while waiting.

Distinguish voluntary resource release from forced preemption when describing resource reclamation.

Prevention can waste utilization or cause starvation; avoiding deadlock is not the same as treating every process fairly.

Operating System module

OS25 | Avoidance and the Banker safety test

Avoidance allows requests only if the resulting state remains safe.

Banker requires maximum claims.

Compute Need=Max−Allocation.

Set Work=Available and repeatedly find an unfinished process with Need≤Work componentwise; simulate completion by adding its Allocation to Work.

Finishing all gives a safe sequence.

Tentatively grant a request only if it is within Need and Available and the safety test succeeds.

Unsafe does not mean already deadlocked.

Banker is avoidance, not CPU scheduling, detection or prevention by breaking a condition.

Example

Worked microexample

- One resource type: Available=1; P1 holds 1 and needs 1 more; P2 holds 2 and needs 2 more. P1 can finish, returning its allocation so Work becomes 2; P2 can then finish. Safe order P1,P2. A maximum claim must be known before testing.

University of Illinois Chicago, Deadlocks

OS26 | Detection, recovery and intentional ignoring

Detection asks whether the current state is deadlocked.

Recovery can terminate selected processes or reclaim safely preemptable resources and roll affected work back to a checkpoint.

Termination and rollback are both possible recovery tools; choose according to the resource and process conditions.

Repeatedly sacrificing the same victim can cause starvation.

Checking on every request versus periodically trades overhead against detection delay; no universal best interval exists.

Ignoring a sufficiently rare problem is sometimes called the ostrich policy, but real systems still prevent or detect selected deadlocks. “Deadlock acceptance” can mean intentional ignoring, not necessarily an invalid approach.

A detection graph, avoidance safety graph and ordinary scheduling queue solve different problems.

University of Illinois Chicago, Deadlocks

OS27 | Address binding, dynamic loading and linking

Compile-time binding uses absolute addresses if placement is known.

Load-time binding relocates when placed into memory.

Execution-time binding translates while running and permits movement with hardware support.

Logical/virtual addresses are generated for a process; physical addresses select RAM.

An MMU translates logical addresses to physical addresses, not the reverse.

Dynamic loading brings a routine in when needed.

Dynamic linking resolves references to external libraries at load/runtime, allowing suitable library code pages to be shared; static linking copies selected code into the executable.

Swapping traditionally moves a whole process to backing store; paging moves individual pages.

These differ from a context switch, which primarily changes execution state.

Operating System module

OS28 | Contiguous allocation and fit policies

A contiguous allocation gives a process one continuous region.

In the standard base/limit model, base is that process’s physical starting address and limit is its permitted length.

Check $0 \le logical_offset < limit$, then $physical = base + offset$.

The relocation register does not normally hold the OS base.

If limit instead means last valid offset, the inequality changes; specify the convention.

First-fit selects the first adequate hole; next-fit continues from the previous search position; best-fit selects the smallest adequate hole; worst-fit selects the largest.

No policy universally optimizes both search time and future storage utilization.

Binary-fit is not one of these standard names.

Example

Worked microexample

- Holes 100,500,200,300 and request 180: first-fit selects 500, best-fit 200, worst-fit 500. For base=4000,length=100, logical offset 99 maps to 4099; offset 100 traps.

Operating System module

OS29 | Internal and external fragmentation

Internal fragmentation is unused space inside an allocated unit.

External fragmentation is sufficient total free space split into holes that cannot satisfy a required contiguous allocation.

Coalescing joins adjacent free holes; compaction moves allocated regions to consolidate free space and requires relocation support.

Basic fixed-frame paging avoids external fragmentation of page-frame allocation but can waste the final page internally.

Variable-length segmentation/contiguous partitions can fragment externally.

This is not a statement that every allocator inside a paging OS is fragmentation-free.

Track units and round allocations up rather than assuming all requests exactly fill blocks.

Example

Worked microexample

- A 10 KiB object in 4 KiB pages needs ceil(10/4)=3 pages: 12 KiB allocated and 2 KiB unused. Two 3 KiB free holes cannot serve one contiguous 5 KiB request although 6 KiB is free.

Operating System module

OS30 | Paging, page tables and TLB

Pages and physical frames have equal fixed size P. Divide virtual address v into page=floor(v/P) and offset=v mod P. The page table maps a page to a frame and protection/status flags.

Physical address=frame×P+offset.

A TLB caches translations; a TLB miss may require a table walk while the page is already resident.

Table size depends on the number of virtual pages and entry size, not just physical RAM.

Protection and present bits must be checked before access.

A page-table lookup is not a CPU scheduling operation.

Example

Worked microexample

- Page size 1024, v=2053: page 2, offset 5. If page 2 maps to frame 7, physical address=7173. With 16-bit virtual addresses and 1 KiB pages, there are 64 virtual pages and a 6-bit page number.

Operating System module

OS31 | Segmentation and contrasts with paging

Segmentation uses logical variable-length units such as code, stack and arrays.

A segment-table entry provides base, limit and permissions.

For (segment,offset), validate the segment and check the bound, then add base+offset.

An out-of-range offset traps; it must not spill into the next segment.

Under the basic models, paging uses fixed units/page tables and can waste space internally; segmentation uses variable logical units/segment tables and can fragment externally.

Segments reflect program structure whereas ordinary pages are largely transparent to applications.

Apply all three contrasts when comparing paging and segmentation.

Example

Worked microexample

- Segment 1 has base 8000 and length 500. Offset 120 maps to 8120; offset 500 is invalid under the length convention.

Operating System module

OS32 | Virtual memory and page-fault handling

Virtual memory provides translated, protected address spaces and allows nonresident portions of a program to be backed by storage.

It does not add physical RAM or remove all capacity constraints.

Demand paging brings a page in when accessed.

On a nonresident access: trap, validate mapping/permissions, find a frame or choose a victim, save a dirty victim if needed, obtain contents, update mapping and restart.

Not every fault reads a disk: some allocate zero-filled pages or establish shared mappings.

An illegal access is rejected, not automatically loaded.

Excessive active-page demand can cause thrashing: repeated faults dominate useful computation.

Reducing active load or adding suitable memory can help.

Example

Worked microexample

- If access costs 100 ns and an additional page-in costs 5 ms, a fault probability of 10^−4 adds roughly 500 ns per access; rare long delays still matter.

Operating System module

OS33 | Replacement-policy traces

Replacement chooses a victim when no suitable free frame is available.

FIFO evicts the earliest loaded page; a hit does not refresh insertion time.

LRU evicts the least recently accessed and updates recency on every access.

Optimal evicts the page used farthest in the future, serving as an offline lower bound.

LFU uses reference counts and needs tie/aging rules.

Write a row per reference, identify hits before evicting, and count initial loads as faults.

FIFO may exhibit Belady's anomaly when more frames cause more faults; LRU and optimal have the stack property.

Page buffering retains useful free/modified pages to reduce effective fault cost.

Example

Worked microexample

- Three empty frames, references 1,2,3,1,4,1: FIFO faults at 1,2,3,4 and the last 1: five faults. LRU updates page 1 on its hit, evicts 2 for 4, and ends with a hit on 1: four faults.

Operating System module

OS34 | Device controllers, drivers and I/O services

A controller is device hardware exposing status, control and data registers.

A driver converts OS requests into device-specific operations.

Device independence and uniform naming hide brand-specific details; error handling reports/retries supported failures.

Polling checks readiness repeatedly; interrupts request CPU attention; DMA transfers blocks after setup without CPU work for every byte.

Both the kernel and drivers participate in I/O; specify the abstraction level being discussed.

Plug-and-play discovers/configures compatible new hardware but may still require a driver.

Input/output/storage/NIC devices are OS-managed hardware; user accounts are identities.

Printer-paper and network errors can be handled/reported, but continued operation during total power loss requires hardware support, not OS software alone.

Operating System module

OS35 | Buffering, caching, spooling and clocks

Buffering temporarily holds data in transit to absorb speed/transfer-size differences.

Caching retains copies to accelerate reuse.

Spooling queues jobs for a device such as a printer, usually on disk, so producers need not own it throughout job creation.

These have different purposes even if one implementation combines them.

The clock/timer supports timekeeping, timeouts, accounting and preemptive scheduling.

A system monitor reports CPU, memory, storage and network usage; it measures rather than directly creating more resources.

Disk initialization/filesystem creation is OS/tool work, firmware begins boot, and drive firmware may remap bad sectors.

A broad “OS does all disk functions” claim needs these responsibility boundaries.

Operating System module

OS36 | Files, descriptors and ordinary operations

A file has content and metadata such as name, size, owner, permissions and timestamps. open obtains an open-file reference, read/write transfer bytes, seek changes an offset, close releases a reference, truncate changes size and unlink removes a name.

A file descriptor is a process-local integer handle; the associated open-file state contains the current offset and access mode.

Descriptor and offset are not synonyms.

Read advances the offset on seekable files and may return fewer bytes than requested; zero normally indicates end-of-file.

Sockets also support ordinary read/write, alongside socket/connect/send operations, so read is not uniquely “not a network call.” FAT32, NTFS and ext4 are filesystem formats; HTTP is an application communication protocol.

Operating System module

Linux man-pages read(2)

OS37 | Directories, mounting and links

A single-level directory requires unique names in one namespace.

Two-level directories separate users.

A tree allows nested directories and absolute/relative paths; links can create shared graphs.

Modern general-purpose systems commonly present hierarchical namespaces, but a feature name alone does not prove that every system excludes simpler arrangements.

A hard link is another name for the same underlying file object, usually within one filesystem; a symbolic link stores a pathname and can point across filesystems or become dangling.

Removing one hard link does not erase content while other names/open references retain it.

Mounting attaches a filesystem at a mount point; it does not format the device.

NFS provides remote filesystem access; FTP transfers files rather than itself mounting a local filesystem.

Operating System module

OS38 | Contiguous, linked and indexed file allocation

Contiguous allocation uses consecutive blocks; logical block i is start+i and supports direct access, but growth and external fragmentation are problems.

Extents allow several contiguous runs.

Linked allocation chains blocks and supports growth without reserving a complete contiguous file; locating block i requires traversing preceding links.

FAT stores chains in a table; caching that table does not change the chain into a direct arithmetic mapping.

Indexed allocation uses a per-file array/tree of block pointers and supports direct lookup with metadata overhead.

A Unix inode can combine direct pointers with indirect levels.

Check pointer size, block size, index capacity and whether metadata reads are cached.

Example

Worked microexample

- A 1024-byte index block with 4-byte pointers holds 256 block addresses. With 1024-byte data blocks, one such index level addresses 256 KiB of file data, excluding metadata and any reserved entries.

Operating System module

OS39 | Protection, authentication and threats

Authentication verifies identity; authorization checks permitted operations.

An ACL lists principals and rights; owner/group/other bits are a simpler policy model, not universal defaults granting all owners full access or everyone read.

Read, write, execute, append and delete are different rights, with filesystem-specific rules.

Least privilege limits damage.

Password hashing verifies secrets without storing plaintext; one-time passwords expire after use.

Encryption protects confidentiality under correct key use but does not automatically provide integrity.

Trojans masquerade as useful software, worms propagate independently and viruses attach to hosts; denial of service harms availability.

Modern OS families can support cryptography, authentication and firewalls, but support/configuration depends on product.

CLI, GUI, speech/natural language and VR are possible interfaces, not a closed list excluding the last two.

Operating System module

OS40 | Linux, Android and product-history limits

Linux is a Unix-compatible kernel written from scratch by Linus Torvalds; a distribution combines it with userland, libraries and tools.

Android's base kernel layer is Linux; its framework and applications are higher layers.

AOSP is open source, while a shipped device can include proprietary applications/services.

Windows and macOS are both proprietary products, although macOS contains open-source components, so "which is not open source?" offering both is ambiguous.

Microsoft Office is an application suite.

Windows is the Microsoft family commonly described as succeeding the MS-DOS desktop era; Solaris originated at Sun. "First widely used OS," "first multitasking OS," and generation/function pairings require a definition and historical evidence; UNIX was not the first system ever to multitask. "Most mobile devices" needs a population and date; architecture facts do not prove market share.

Operating System module

Android Open Source Project architecture

Linux Kernel Organization, About

Oracle Solaris 11

OS41 | Identify the operating-system conditions

Check whether the stated conditions support one answer or several before selecting a response.

A general system-call purpose should say requesting a kernel service; a process purpose should say executing a program.

CPU, memory and I/O management span multiple managers; device, process, file and security management describe particular responsibilities.

Virtual memory provides isolation and translated space; it does not remove every memory constraint.

Specify the OS/API, scheduling variant, arrivals/quantum, exact graph, base-limit convention or scope of comparison needed to answer the question.

Support the answer with the relevant operating-system rule and stated assumptions.

A process/resource graph question requires its actual edges; theory alone cannot reconstruct a missing diagram.

Operating System module

OS-S01 | UNIX origins and the first four historical generations

UNIX development began at Bell Labs in 1969, led by Ken Thompson and Dennis Ritchie with other collaborators. Ritchie is therefore a co-creator, not the sole inventor of the first operating system.

The later UNIX time-sharing system described by Ritchie and Thompson is multiuser and interactive and supports asynchronous processes. Its later versions incorporated multiprogramming. This history does not make UNIX the first system ever to multitask.

In the first four generations of the cited NYU teaching taxonomy, the first generation uses vacuum tubes and manual single-user operation without a separate resident OS. The second generation pairs transistors with uniprogrammed batch systems; the resident OS starts successive jobs without being reloaded for every job.

The third generation pairs integrated circuits with multiprogramming. The fourth emphasizes personal computers. These are characteristic-era associations, not claims that each feature existed exclusively in one era.

Nokia Bell Labs - 50 years of Unix

Ritchie and Thompson - The UNIX Time-Sharing System

NYU OS 2250 - History of Operating Systems

OS-S02 | Thread management costs and execution privilege

In the conventional library-thread comparison, user-level thread creation and switching can avoid entering the kernel. Kernel-managed thread creation and management commonly incur additional kernel work and are therefore generally slower to create and manage in this comparison. This is a typical comparison, not a universal speed or reliability guarantee.

Being scheduled or managed by the kernel does not give application thread code kernel privilege. Ordinary applications and supporting utilities execute in restricted user mode and request protected services through libraries and system calls. Kernel component code executes in privileged kernel mode and performs permitted low-level hardware operations.

Operating System module · page 28

Operating System module · page 29

Operating System module · page 53

OS-S03 | Memory allocation and page-policy decisions

Memory management tracks which primary-memory regions are free or in use, assigns memory to processes and data, and deallocates it when no longer needed or when a process terminates. Allocation and deallocation concern memory ownership; CPU selection belongs to scheduling.

A buddy allocator manages power-of-two block sizes. It takes a smallest sufficient block, splitting larger blocks when necessary, and can merge a freed block with its free buddy. This is not the worst-fit rule of selecting the largest arbitrary hole.

A fetch policy decides when contents are brought into memory; demand paging fetches a valid nonresident page when it is accessed. A cleaning policy concerns writing modified contents to backing storage. A replacement policy chooses which resident page to evict. These decisions can cooperate during one page fault, but they are different decisions.

Operating System module · page 5

Operating System module · page 31

Operating System module · page 38

Operating System module · page 40

Operating System module · page 42

Linux kernel documentation - Physical Memory, free_area

OS-S04 | Communication endpoints and timed I/O events

In the descriptor-based socket API, socket() creates a communication endpoint and returns its descriptor. connect() associates the socket with the requested peer address; send() transmits data to another socket. Receiving through read() uses the ordinary descriptor-input interface described in the file-services lesson.

IRQ means interrupt request. A controller can raise an interrupt to request CPU attention. In a clock-tick scheduler, the scheduler initializes a running process's quantum counter; each clock interrupt decrements it, and reaching zero triggers scheduling. Clock support also maintains time of day and usage accounting.

Direct message communication names the receiving process explicitly. Indirect communication uses a mailbox or port. This naming distinction is independent of whether a send or receive blocks: naming a destination does not determine synchronization semantics.

Linux man-pages - socket(2)

Linux man-pages - connect(2)

Linux man-pages - send(2)

Operating System module · page 45

Operating System module · page 46

UIC Operating Systems - Processes, message-passing naming and synchronization

OS-S05 | Compilation versus operating-system execution services

A compiler translates source code toward an executable representation. For example, the conventional GCC toolchain distinguishes preprocessing, compilation proper, assembly and linking. A compiler uses operating-system services as a development tool; it is not the runtime kernel component that selects ready processes, manages memory or handles protected service requests.

GCC - Options Controlling the Kind of Output

OS-S06 | Filesystem links in Linux

Linux provides link() to create another hard-link name for an existing file. The old and new names identify the same file, so hard links are not a Windows-only facility. unlink() removes a filesystem name; when that name is a symbolic link, it removes the link itself. Removing a name and erasing the target’s contents are different operations.

Linux man-pages · link(2)

Linux man-pages · unlink(2)