Operating Systems

Comprehensive Study Guide

Chapters 10 · 13 · 14

Ch.10 — File-System Interface	Ch.13 — I/O Systems	Ch.14 — Protection
• 31 Terms • 6 Comparisons • 7 Enum Lists	• 33 Terms • 5 Comparisons • 7 Enum Lists	• 27 Terms • 6 Comparisons • 6 Enum Lists

Terms · Comparisons · Enumeration Lists · 100% Comprehensive Coverage

Chapter 10

File-System Interface

File Concept · Structure Types · Access Methods · Directory · Mounting · Sharing · Protection

Section 1 — Terminology & Definitions

Term	Definition / Details
File	A contiguous logical address space; an abstract data type (ADT). Files contain Data (numeric, character, binary) or Programs. File structure types: (1) None — sequence of words/bytes, (2) Simple record structure — Lines, Fixed length, Variable length, (3) Complex Structures — Formatted document, Relocatable load file. OS can simulate the last two by inserting appropriate control characters. Who decides structure: Operating System or the Program.
File Name	The only information kept in human-readable form. Every other attribute is stored numerically or as a pointer.
File Identifier	A unique tag (number) that identifies the file within the file system. Not human-readable; used internally by the OS.
File Type	Needed for systems that support different types of files. Indicated by extension. Types: executable (exe, com, bin), object (obj, o), source code (c, cc, java, pas, asm), batch (bat, sh), text (txt, doc), word processor (wp, tex, rtf, doc), library (lib, a, so, dll), print/view (ps, pdf, jpg), archive (arc, zip, tar), multimedia (mpeg, mov, rm, mp3, avi).
File Location	A pointer to the file's location on the storage device. Maintained in the directory structure on disk.
File Size	The current size of the file, stored as file metadata in the directory structure.
File Protection	Controls who can perform reading, writing, and executing on the file. Stored as file metadata.
File Time/Date/User ID	Metadata stored with the file for protection, security, and usage monitoring.
File Attributes (Metadata)	All information kept in the directory structure: Name, Identifier, Type, Location, Size, Protection, Time/Date/User ID. Many variations including extended file attributes such as file checksum. Both directory structure and files reside on disk; backups kept on tapes.

Term	Definition / Details
File Operations (ADT)	File is an ADT — a set of objects together with a set of operations: Create, Write, Read, Reposition within file (seek), Delete, Truncate (cut off). Also: Open(Fi) — search directory structure on disk for entry Fi and move content of entry to memory; Close(Fi) — move content of entry Fi in memory back to directory structure on disk.
Open File Table	Kernel data structure for managing open files. Data managed for each open file: (1) File pointer — pointer to last read/write location per process, (2) File-open count — counter of times file is open; allows removal when last process closes it, (3) Disk location of file — cache of data access information, (4) Access rights — per-process access mode information.
File Pointer	A pointer to the last read/write location in an open file. Maintained per process — each process that has the same file open has its own file pointer.
File-Open Count	A counter of the number of times a file is currently open across all processes. When it reaches zero (last process closes), data is removed from the open-file table.
File Locking	Provided by some OSes and file systems. Mediates (manages) access to a file. Mandatory — access is denied depending on locks held and requested (OS enforces). Advisory — processes can find status of locks and decide what to do themselves (cooperative).
Sequential Access	Simplest access method. Data read/written in order. Operations: read next, write next, reset (go to beginning), no read after last write (rewrite). Most common access method.
Direct Access (Relative Access)	Reads or writes any block in any order. Operations: read n, write n, position to n, read next, write next, rewrite n. n = relative block number. Sequential access can be simulated: cp=0 (reset), read cp then cp=cp+1 (read next), write cp then cp=cp+1 (write next).
Index Access	Uses an index file containing pointers to logical records in a relative file. For large files, index itself may need to be indexed (multi-level index). Built on top of direct access.
Directory	A collection of nodes containing information about all files. Both directory structure and files reside on disk. Backups of both kept on tapes.
Volume	An entity containing a file system. Each volume containing a file system tracks its info in a device directory or volume table of contents. Disk/partition can be used raw (without a file system) or formatted with a file system.
Partition	A subdivision of a disk (also called minidisk or slice). Disks or partitions can be RAID protected against failure. Entity containing file system known as a volume. As well as general-purpose file systems there are many special-purpose file systems frequently all within the same OS.

Term	Definition / Details
Single-Level Directory	A single directory for all users. Problems: (1) Naming problem — all files must have unique names across all users, (2) Grouping problem — cannot logically group files by type or owner.
Two-Level Directory	Separate directory per user (User File Directory — UFD). Master File Directory (MFD) has one entry per user. Features: path names, same file name allowed for different users, efficient searching. Limitation: no grouping capability.
Tree-Structured Directory	General tree with arbitrary depth. Features: efficient searching, grouping capability, current directory (working directory), absolute path names (from root), relative path names (from current directory). Creating a new file/subdirectory done in current directory. Deleting a directory deletes the entire subtree rooted at it (e.g., deleting "mail" ⇒ deletes all files under it).
Acyclic-Graph Directory	Allows shared subdirectories and files — unlike tree where each file has exactly one parent. Shared files can have several different names (aliases). More flexible than tree structure. Allows same file to appear in multiple directories.
File System Mounting	A file system must be mounted before it can be accessed. An unmounted file system is mounted at a mount point — a location in the existing directory tree where the new file system will be attached and accessible to users.
File Sharing	Sharing files on multi-user systems through a protection scheme. On distributed systems, shared across a network using NFS (standard UNIX) or CIFS (standard Windows). Access methods: manually via FTP, automatically via distributed file systems, semi-automatically via the World Wide Web.
Client-Server File Sharing	Client-server model allows clients to mount remote file systems from servers. Server can serve multiple clients. Client and user-on-client identification is insecure or complicated. Standard OS file calls are translated into remote calls. NFS = standard UNIX protocol; CIFS = standard Windows protocol.
Remote File System Failures	Remote file systems add new failure modes due to network failure or server failure. Recovery from failure can involve state information about status of each remote request. Stateless protocols such as NFS include all information in each request, allowing easy recovery but providing less security.
Distributed Information Systems	Systems such as LDAP, DNS, NIS, Active Directory that implement unified access to information needed for remote computing (distributed naming services).
File Protection	File owner/creator should be able to control: what can be done (type of access) and by whom (which users). Types of access: Read, Write, Execute, Append (attach), Delete, List. Unix model: mode of access (RWX) for three user classes —

Term	Definition / Details
	(a) owner access 7 = 111 RWX, (b) group access 6 = 110 RW only, (c) public access 1 = 001 Execute only. Ask manager to create a group (unique name G), add users, define appropriate access per file or subdirectory.
Access Control List (ACL)	A per-object/file list specifying which users or groups have which access rights. More fine-grained than three-class Unix model. Windows: per-file security with Full Control, Modify, Read & Execute, Read, Write, Special Permissions — defined per user and group.

⚖ Section 2 — Comparisons & Differentiations

File Structure Types

Type	Description	Example
None (sequence of bytes)	Unstructured — just a sequence of words or bytes. OS imposes no structure.	Unix/Linux files — OS sees only bytes
Simple Record Structure — Lines	Fixed or variable length records/lines.	Text files with newlines
Simple Record Structure — Fixed Length	Records of exact same size.	Old database flat files
Simple Record Structure — Variable Length	Records of different sizes.	CSV files with varying fields
Complex — Formatted Document	Rich structure with formatting metadata.	Word processor files (.docx, .tex)
Complex — Relocatable Load File	Binary with relocation information for loader.	Object files (.o, .obj), executables

Sequential Access vs. Direct Access vs. Index Access

Aspect	Sequential	Direct	Index
Access order	In sequence only (next)	Any block (random — read/write n)	Via index → record in relative file
Key operations	read next, write next, reset	read n, write n, position to n, rewrite n	Search index → locate logical record
Simulation	Base method	Sequential simulated via cp	Built on top of direct access

Aspect	Sequential	Direct	Index
		pointer	
Best for	Tape, log files	Disks, databases	Large indexed files, databases
Block number	Sequential	n = relative block number	Index lookup → block number

Directory Structure Types Comparison

Type	Naming	Grouping	Sharing	Notes
Single-Level	PROBLEM — all files must be unique	PROBLEM — no grouping	All users share same space	Simplest; for single-user systems
Two-Level	Solved — same name for different users	No grouping capability	Separate per-user directories (UFD)	Path names supported
Tree-Structured	Absolute & relative paths	Full grouping capability	Working directory concept	Deleting dir deletes entire subtree
Acyclic-Graph	Aliases — same file, many names	Full grouping + file sharing	Shared files and directories	Most flexible; links to shared files

Mandatory vs. Advisory File Locking

Aspect	Mandatory	Advisory
Enforcement	OS enforces — access denied based on locks held/requested	Application enforces — processes check lock status themselves
Who decides	Operating System enforces automatically	Process/application decides what to do
Also called	Compulsory locking	Consultative locking

NFS vs. CIFS

Aspect	NFS (Network File System)	CIFS (Common Internet File System)
Platform	Standard UNIX/Linux	Standard Windows
Protocol type	Stateless — all info in each request	Stateful

Aspect	NFS (Network File System)	CIFS (Common Internet File System)
Recovery	Easy (stateless design)	More complex recovery
Security	Less secure	More secure

Unix File Permission Classes (RWX Model)

Class	Symbol	Binary	Decimal	Meaning
Owner	a (rwx)	1 1 1	7	Read + Write + Execute
Group	b (rwx)	1 1 0	6	Read + Write only
Public	c (rwx)	0 0 1	1	Execute only

Section 3 — Enumeration Questions

7 File Attributes (Metadata) Stored in Directory Structure

7 File Attributes (Metadata) Stored in Directory Structure
1. **Name** — Only information kept in human-readable form.
2. **Identifier** — Unique tag (number) identifying file within the file system.
3. **Type** — Needed for systems that support different file types.
4. **Location** — Pointer to the file's location on the storage device.
5. **Size** — Current size of the file.
6. **Protection** — Controls who can read, write, or execute the file.
7. **Time, Date, and User Identification** — Data for protection, security, and usage monitoring.

8 File Operations (ADT)

8 File Operations (ADT)
1. **Create** — Allocate space and create a new directory entry.
2. **Write** — Write data to the file at a specified position.
3. **Read** — Read data from the current read position.
4. **Reposition within file (Seek)** — Move the file pointer to a specified position.

5. **Delete** — Remove directory entry and free allocated file space.
6. **Truncate** — Cut off / erase file contents while keeping attributes (resets length to zero).
7. **Open(Fi)** — Search directory structure on disk for entry Fi and move content of entry to memory.
8. **Close(Fi)** — Move content of entry Fi in memory back to directory structure on disk.

4 Data Needed to Manage Open Files

4 Data Needed to Manage Open Files
1. **File Pointer** — Pointer to last read/write location, per process that has the file open.
2. **File-Open Count** — Counter of number of times file is open — allows removal from open-file table when last process closes it.
3. **Disk Location of the File** — Cache of data access information (where the file is on disk).
4. **Access Rights** — Per-process access mode information.

6 Directory Operations

6 Directory Operations
1. **Search for a file** — Find a specific file by name within the directory.
2. **Create a file** — Add a new file entry to the directory.
3. **Delete a file** — Remove a file entry (rm <file-name>).
4. **List a directory** — Display the contents of a directory.
5. **Rename a file** — Change the name of an existing file entry.
6. **Traverse the file system** — Search/walk through the tree structure of directories.

3 Goals of Directory Structure Design

3 Goals of Directory Structure Design
1. **Efficiency** — Locating a file quickly.
2. **Naming** — Convenient to users — two users can have same name for different files; same file can have several different names.
3. **Grouping** — Logical grouping of files by properties (e.g., all Java programs, all games).

6 Types of File Access Rights (Protection)

6 Types of File Access Rights (Protection)
1. Read — View the file contents.
2. Write — Modify the file contents.
3. Execute — Run the file as a program.
4. Append — Add data to the end of the file only.
5. Delete — Remove the file from the file system.
6. List — View the file name and attributes in a directory listing.

10 Common File Types (Extensions & Functions)

10 Common File Types (Extensions & Functions)
1. executable — exe, com, bin — ready-to-run machine-language program.
2. object — obj, o — compiled machine language, not yet linked.
3. source code — c, cc, java, pas, asm, a — source code in various languages.
4. batch — bat, sh — commands to the command interpreter/shell.
5. text — txt, doc — textual data and documents.
6. word processor — wp, tex, rtf, doc — various word-processor formats.
7. library — lib, a, so, dll — libraries of routines for programmers.
8. print or view — ps, pdf, jpg — ASCII or binary file for printing or viewing.
9. archive — arc, zip, tar — related files grouped into one, sometimes compressed.
10. multimedia — mpeg, mov, rm, mp3, avi — binary file containing audio or A/V information.

Chapter 13
I/O Systems

I/O Hardware · Application Interface · Kernel Subsystem · STREAMS · Performance

Section 1 — Terminology & Definitions

Term	Definition / Details
I/O Management	A major component of OS design and operation. Important aspect of computer operation. I/O devices vary greatly; various methods to control them; performance management critical; new types of devices frequent. Ports, busses, and device controllers connect to various devices. Device drivers encapsulate device details and present uniform device-access interface to I/O subsystem. Incredible variety of I/O devices: Storage, Transmission, Human-interface.
Port	A connection point for a device. Signals from I/O devices interface with the computer through ports.
Bus	A communication pathway connecting devices. Can be daisy chain or shared direct access. Examples: PCI bus (connects graphics, IDE disk controller, expansion bus interface, keyboard), SCSI bus (connects multiple disks).
Controller (Host Adapter)	Electronics that operate a port, bus, or device. Sometimes integrated into motherboard, sometimes a separate circuit board (host adapter). Contains processor, microcode, private memory, bus controller. Some talk to per-device controllers with bus controller, microcode, and memory.
Device Registers	Devices usually have registers where device driver places commands, addresses, and data to write, or reads data after command execution. Types: Data-in register, Data-out register, Status register, Control register. Typically 1–4 bytes or FIFO buffer. Devices have addresses used by Direct I/O instructions or Memory-mapped I/O.
Direct I/O Instructions	I/O instructions that directly control devices by addressing device registers through I/O port addresses (e.g., 000–00F for DMA controller, 020–021 for interrupt controller, 040–043 for timer, 320–32F for hard-disk controller, 3D0–3DF for graphics controller).
Memory-Mapped I/O	Device data and command registers mapped into the processor address space. CPU accesses device registers using regular memory load/store instructions. Especially useful for large address spaces (e.g., graphics controllers).

Term	Definition / Details
Polling	CPU repeatedly reads the status register until I/O is complete (busy-wait cycle). Step 1 is busy-wait — reasonable if device is fast, inefficient if slow. Polling in 3 instruction cycles: read status, logical-and to extract status bit, branch if not zero. CPU misses a cycle → data overwritten/lost. CPU could switch to other tasks? But risk losing data.
Interrupt	CPU interrupt-request line triggered by I/O device — checked by processor after each instruction. Interrupt handler receives interrupts. Maskable — to ignore or delay some interrupts. Interrupt vector dispatches interrupt to correct handler (context switch at start and end). Based on priority. Some nonmaskable. Interrupt chaining if more than one device at same interrupt number. Also used for: exceptions (terminate process, crash system), page faults, system calls (trap). Multi-CPU systems can process interrupts concurrently.
Interrupt Vector Table	Maps interrupt numbers to handler addresses. x86 examples: 0=divide error, 1=debug exception, 2=null interrupt, 3=breakpoint, 4=INTO-detected overflow, 5=bound range exception, 6=invalid opcode, 7=device not available, 8=double fault, 13=general protection, 14=page fault, 16=floating-point error, 17=alignment check, 18=machine check, 19–31=Intel reserved, 32–255=maskable interrupts.
Maskable Interrupt	An interrupt that can be disabled (masked) by the CPU or OS to delay or ignore it during critical operations. Interrupt vectors 32–255 in x86 are maskable interrupts.
Nonmaskable Interrupt	An interrupt that cannot be disabled and must always be handled immediately. Used for critical hardware errors.
Interrupt Chaining	When more than one device shares the same interrupt number, interrupt handlers are chained — called in sequence until the correct one handles the interrupt.
DMA (Direct Memory Access)	Used to avoid programmed I/O (one byte at a time) for large data movement. Requires DMA controller. Bypasses CPU to transfer data directly between I/O device and memory. OS writes DMA command block into memory: source address, destination address (buffer X), read/write mode, count of bytes (C). DMA controller grabs bus from CPU (bus mastering), transfers bytes to buffer X, increases memory address, decreases C until C=0. When C=0, DMA interrupts CPU to signal completion.
Device Driver	Software layer encapsulating device-specific details. Presents uniform device-access interface to I/O subsystem. Each OS has its own I/O subsystem structures and device driver frameworks. New devices talking already-implemented protocols need no extra work. Device-driver layer hides differences among I/O controllers from kernel.
Device Classification Dimensions	Devices vary in: (1) Data-transfer mode: character or block. (2) Access method: sequential or random. (3) Transfer schedule: synchronous or asynchronous. (4) Sharing:

Term	Definition / Details
	dedicated or sharable. (5) Device speed: latency, seek time, transfer rate, delay between operations. (6) I/O direction: read-only, write-only, or read-write.
Block Device	Transfers data in fixed-size blocks (e.g., disk drives). Commands: read, write, seek. Supports raw I/O, direct I/O, file-system access. Memory-mapped file access possible (file mapped to virtual memory, clusters brought via demand paging). Uses DMA.
Character Device	Transfers data character by character (e.g., keyboards, mice, serial ports). Commands: get(), put(). Libraries layered on top allow line editing.
Network Socket	Interface varying enough from block and character devices to have its own interface. Unix and Windows NT/9x/2000 include socket interface — separates network protocol from network operation. Includes select() functionality. Approaches: pipes, FIFOs, streams, queues, mailboxes.
Clocks and Timers	Provide current time, elapsed time, and timer. Normal resolution about 1/60 second. Some systems provide higher-resolution timers. Programmable interval timer used for timings and periodic interrupts. ioctl() on Unix covers clocks and timers.
ioctl()	Unix system call covering odd/specific I/O aspects such as clocks, timers, and direct device manipulation. Sends arbitrary bits to device control register and data to device data register — an escape/back door for device-specific control.
Blocking I/O	Process is suspended until I/O is completed. Easy to use and understand. Insufficient for some needs (e.g., interactive apps). (Synchronous I/O — process waits.)
Nonblocking I/O	I/O call returns as much data as currently available without waiting for completion. Used for user interfaces and data copy (buffered I/O). Implemented via multi-threading. Returns quickly with count of bytes read or written. Uses select() to check if data ready then read()/write() to transfer.
Asynchronous I/O	Process runs while I/O executes concurrently. Difficult to use. I/O subsystem signals process when I/O is completed. More complex programming model.
I/O Scheduling	Some I/O request ordering via per-device queue. Some OSes try fairness. Some implement Quality of Service (e.g., IPQOS).
Device-Status Table	A kernel data structure tracking the status of all I/O devices (idle or busy). Contains per-device request queues with file, operation, address, and length for each pending request. Allows OS to manage multiple concurrent device requests.
Buffering	Storing data in memory while transferring between devices. Purposes: (1) cope with device speed mismatch, (2) cope with device transfer size mismatch, (3) maintain copy semantics.

Term	Definition / Details
	Double buffering — two copies of the data (kernel and user, varying sizes, full/being processed and not-full/being used). Copy-on-write can be used for efficiency.
Caching	Faster device/memory holding a copy of data. Always just a copy (unlike buffering where buffer may be the only copy). Key to performance. Sometimes combined with buffering.
Spooling	Holds output for a device that can serve only one request at a time (e.g., printing). Manages requests so multiple processes can send output simultaneously.
Device Reservation	Provides exclusive access to a device. System calls for allocation and de-allocation. Must watch out for deadlock.
Error Handling	OS can recover from disk read, device unavailable, transient write failures. Retry a read or write. Advanced systems (Solaris FMA, AIX) track error frequencies and stop using a device with increasing retryable errors. Most return error number/code when I/O fails. System error logs hold problem reports.
I/O Protection	All I/O instructions defined to be privileged. I/O must be performed via system calls (trap to kernel). Memory-mapped and I/O port memory locations must be protected too. Prevents user processes from accidentally or purposefully disrupting normal operation.
Kernel I/O Data Structures	Kernel keeps state info for I/O components: open file tables, network connections, character device state. Many complex data structures track buffers, memory allocation, dirty blocks. Some systems use object-oriented methods and message passing to implement I/O. Windows uses message passing — message with I/O info passed from user mode into kernel, modified as it flows to device driver and back to process.
Life Cycle of an I/O Request	Reading a file from disk: (1) Determine device holding file, (2) Translate name to device representation, (3) Physically read data from disk into buffer, (4) Make data available to requesting process, (5) Return control to process. Path: system call → kernel checks if satisfiable → if yes: return data; if no: send to device driver + block process → interrupt when done → return completion/error code.
STREAMS	Full-duplex communication channel between user-level process and device in Unix System V. Components: (1) STREAM head — interfaces with user process, (2) Driver end — interfaces with device, (3) Zero or more STREAM modules between them — each has read queue and write queue. Message passing between queues. Flow control to indicate available/busy. Asynchronous internally, synchronous where user process communicates with stream head.
I/O Performance	I/O is a major factor in system performance. Demands CPU to execute device driver/kernel I/O code. Context switches due to interrupts. Data copying. Network traffic especially stressful. I/O improvement layers: application code → kernel

Term	Definition / Details
	code → device-driver code → device-controller code (hardware) → device code (hardware).

⚖ Section 2 — Comparisons & Differentiations

Polling vs. Interrupts vs. DMA

Aspect	Polling	Interrupts	DMA
CPU involvement	CPU loops checking status register (busy-wait)	CPU checks after each instruction; handler called when done	CPU sets up transfer; DMA does actual transfer autonomously
Efficiency	Inefficient for slow devices; wastes CPU cycles	Efficient — CPU free until interrupted	Most efficient for large data — CPU not involved in actual transfer
Best for	Fast devices	General purpose I/O	Large bulk data transfers (disk, network)
CPU free during transfer?	No — busy-wait	Yes	Yes
Instruction cycles needed	3 (read status, AND, branch)	Interrupt overhead (context switch)	Setup overhead, then autonomous
Risk	Data lost if miss a cycle	Interrupt latency	Bus contention

Blocking vs. Nonblocking vs. Asynchronous I/O

Aspect	Blocking	Nonblocking	Asynchronous
Process behavior	Suspended until I/O completes	Returns immediately with available data	Continues running while I/O executes
Ease of use	Easy to use and understand	Moderate	Difficult to use
Implementation	Simple	Multi-threading, select()	I/O subsystem signals via callback/signal
Notification	N/A — resumes when done	Returns count of bytes read/written	I/O subsystem signals when done
Also called	Synchronous I/O	—	—

Buffering vs. Caching vs. Spooling

Aspect	Buffering	Caching	Spooling
Purpose	Temporary storage during transfer between devices	Copy in faster storage for repeated access	Hold output for device that serves one request at a time
Data role	May be ONLY copy of data in transit	Always just a copy (master elsewhere)	Queue of outputs waiting to be served
Benefit	Handles speed/size mismatches; copy semantics	Performance — avoids repeated slow device access	Multiple processes share exclusive device
Combined?	Sometimes combined with caching	Sometimes combined with buffering	—
Example	Double buffering (kernel+user)	CPU cache, disk buffer cache	Print spooling

Device Classification: 6 Dimensions

Dimension	Variation 1	Variation 2	Example
Data-transfer mode	Character (stream)	Block	Terminal / Disk
Access method	Sequential	Random	Modem / CD-ROM
Transfer schedule	Synchronous	Asynchronous	Tape / Keyboard
Sharing	Dedicated	Sharable	Tape / Keyboard
Device speed	Latency, seek time, transfer rate, delay between operations	—	—
I/O direction	Read-only	Write-only / Read-write	CD-ROM / Graphics / Disk

Direct I/O Instructions vs. Memory-Mapped I/O

Aspect	Direct I/O Instructions	Memory-Mapped I/O
Mechanism	Dedicated I/O port addresses; special CPU instructions (IN/OUT)	Device registers mapped into processor address space

Aspect	Direct I/O Instructions	Memory-Mapped I/O
CPU instructions	Special I/O instructions	Regular memory load/store instructions
Best for	Simple/smaller I/O address spaces	Large address spaces (especially graphics)
Example	000–00F = DMA, 040–043 = timer	Graphics controller mapped to memory address range

Section 3 — Enumeration Questions

5 Steps of Polling (For Each Byte of I/O)

5 Steps of Polling (For Each Byte of I/O)
1. **Step 1** — Read busy bit from status register until it is 0 (device is free — busy-wait cycle).
2. **Step 2** — Host sets read or write bit and if writing copies data into data-out register.
3. **Step 3** — Host sets command-ready bit to signal the controller.
4. **Step 4** — Controller sets busy bit, executes the transfer.
5. **Step 5** — Controller clears busy bit, error bit, and command-ready bit when transfer is done.

DMA Command Block Information

DMA Command Block Information
1. **Source address** — Where data is coming from (disk address or memory address).
2. **Destination address** — Where data is going to (buffer at address X in memory).
3. **Read or write mode** — Direction of the data transfer.
4. **Count of bytes (C)** — How many bytes to transfer. DMA decreases C; when C=0 transfer is complete.
5. **Location of command block** — OS writes this location to DMA controller so it can read the transfer instructions.

4 Broadly Grouped I/O Device Types (OS View)

4 Broadly Grouped I/O Device Types (OS View)
1. **Block I/O** — Disk drives — commands: read, write, seek. Raw I/O, direct I/O, file-

system access, memory-mapped file access. Uses DMA.
2. Character I/O (Stream) — Keyboards, mice, serial ports — commands: get(), put(). Libraries for line editing layered on top.
3. Memory-Mapped File Access — File mapped to virtual memory; clusters brought via demand paging.
4. Network Sockets — Varies enough to have own interface. select() functionality. Pipes, FIFOs, streams, queues, mailboxes.

3 Purposes of Buffering

3 Purposes of Buffering
1. Cope with device speed mismatch — Data produced faster than consumed (or vice versa) — buffer absorbs the difference.
2. Cope with device transfer size mismatch — Devices transfer in different chunk sizes — buffer adapts.
3. Maintain copy semantics — Ensures data written to buffer reflects version at time of write call, not later modifications.

STREAM Components (3 Parts)

STREAM Components (3 Parts)
1. STREAM Head — Interfaces with the user process at the top of the STREAM pipeline.
2. Zero or More STREAM Modules — Intermediate processing between head and driver. Each has read queue and write queue. Message passing between queues with flow control (available/busy).
3. Driver End — Interfaces directly with the I/O device at the bottom of the pipeline.

7 I/O Performance Improvement Techniques

7 I/O Performance Improvement Techniques
1. Reduce number of context switches — Fewer switches = more CPU time for useful work.
2. Reduce data copying — Minimize number of times data is copied between buffers.
3. Reduce interrupts — Use large transfers, smart controllers, or polling where appropriate.
4. Use DMA — Let DMA controller handle bulk data movement without CPU involvement.

5. **Use smarter hardware devices** — Offload processing to intelligent device controllers.
6. **Balance CPU, memory, bus, and I/O** — Balance all subsystems for highest overall throughput.
7. **Move user-mode processes/daemons to kernel threads** — Reduce context switch overhead for I/O-heavy daemons.

5 Steps: Reading a File from Disk for a Process

5 Steps: Reading a File from Disk for a Process
1. **Step 1** — Determine device holding the file.
2. **Step 2** — Translate name to device representation (locate on disk).
3. **Step 3** — Physically read data from disk into buffer.
4. **Step 4** — Make data available to the requesting process.
5. **Step 5** — Return control to the process.

Chapter 14
Protection

Goals · Principles · Domains · Access Matrix · Implementation · Revocation · Language-Based

Section 1 — Terminology & Definitions

Term	Definition / Details
Protection vs. Security	Protection — any mechanism for controlling access of processes or users to resources defined by the OS. Security — defense of the system against internal and external attacks (denial-of-service, worms, viruses, identity theft, theft of service). Systems first distinguish among users to determine who can do what.
User ID / Security ID	Includes name and associated number, one per user. Associated with all files and processes of that user to determine access control.
Group ID	Allows a set of users to be defined and controls managed as a group. Associated with each process and file.
Privilege Escalation	Allows user to change to effective ID with more rights than their current ID. Used to temporarily gain elevated privileges.
Protection Object	Computer consists of a collection of objects (hardware or software). Each object has a unique name and can be accessed through a well-defined set of operations. Protection problem: ensure each object is accessed correctly and only by authorized processes.
Principle of Least Privilege	Programs, users, and systems should be given just enough privileges to perform their tasks. Limits damage if entity has a bug or is abused. Can be static (during life of system, during life of process) or dynamic (changed by process as needed via domain switching or privilege escalation). "Need to know" — similar concept for access to data.
Grain (of Protection)	The granularity of privilege management. Rough-grained: simpler, easier, but least privilege done in large chunks (e.g., Unix: user or root). Fine-grained: more complex/overhead, but more protective (e.g., file ACLs, RBAC).
Domain of Protection	Domain can be user, process, or procedure. Access-right = <object-name, rights-set> where rights-set is a subset of all valid operations on the object. Domain = set of access-rights. A process must execute in some domain. D1 example: <O1, {read,write}>, <O2,{execute}>, <O3,{read,write}>.

Term	Definition / Details
Access Right	Defined as <object-name, rights-set>. A process in a domain can invoke only those operations listed in its access rights for an object. Process in Di tries "op" on Oj → "op" must be in the access matrix at Access(i,j).
setuid (Set User ID bit)	Unix domain switching via file system. Each file has a domain bit (setuid bit). When file is executed and setuid=on, user-id is set to owner of the file being executed. When execution completes, user-id is reset to original.
su command	Unix command that temporarily switches to another user's domain when that domain's password is provided. Domain switching via password.
sudo command	Unix command prefix that executes a specified command in another domain if the original domain has privilege or the correct password is given. Domain switching via commands.
Access Matrix	View protection as a matrix. Rows = domains. Columns = objects. Access(i,j) = set of operations process in Domain_i can invoke on Object_j. If process in Di tries "op" on Oj, "op" must be in access matrix. User who creates object can define its access column. Access matrix design separates MECHANISM (OS provides matrix+rules, ensures authorized manipulation) from POLICY (user dictates who can access what in what mode).
Dynamic Protection (Access Matrix)	Access matrix expanded to support dynamic protection via special operations: add/delete access rights. Special access rights: (1) owner of Oi, (2) copy op from Oi to Oj (denoted "*"), (3) control — Di can modify Dj access rights, (4) transfer/switch — switch from Di to Dj. Copy and Owner applicable to objects; Control applicable to domain objects.
Copy Right (*)	Denoted by asterisk (*) in access matrix. Allows process in a domain to copy a specific access right to another domain for the same object (within same column). Example: D1 has write* on F3 → can copy write permission to D3 for F3. Before copy: D3 has no access to F3. After copy: D3 gains read access to F3.
Switch Right	Special access right in access matrix allowing domain switching. Domain switching supported by providing "switch" access to other domains as columns. Example: D1 can switch to D2 (has switch in D2 column); D2 can switch to D3 and D4.
Mechanism vs. Policy (Access Matrix)	Mechanism: OS provides access-matrix + rules; ensures matrix manipulated only by authorized agents and rules strictly enforced. Policy: user dictates who can access what object and in what mode. Separation is a key OS design principle.
Global Table	Access matrix implementation Option 1: store ordered triples <domain, object, rights-set> in one table. Search for <Di, Oj, Rk> with M ∈ Rk for operation M on Oj in Di. Disadvantage: table could be large (won't fit in main memory); difficult to group objects (object all domains can read requires many

Term	Definition / Details
	entries).
Access Control List (ACL)	Access matrix implementation Option 2: each column = access list for one object. Per-object list of ordered pairs <domain, rights-set> defining all domains with non-empty access rights. Can have default set (if M ∈ default set, allow access). Access (i,j) defines who can perform what operation. Every access to object must be checked — many objects/rights can be slow.
Capability List	Access matrix implementation Option 3: each row = capability list for one domain. List of objects together with allowed operations on them. Object represented by name or address (called a capability). Possession of capability means access is allowed. Capability list associated with domain but NEVER directly accessible by domain — protected object maintained by OS, accessed indirectly (like a "secure pointer"). Idea can be extended to applications.
Lock-Key Mechanism	Access matrix implementation Option 4: compromise between access lists and capability lists. Each object has list of unique bit patterns called locks. Each domain has list of unique bit patterns called keys. Process in domain can access object only if domain has key that matches one of the object's locks. Keys can be passed freely between domains — easy revocation.
RBAC (Role-Based Access Control)	Solaris 10's protection approach to implement least privilege. Privilege = right to execute a system call or use an option within a system call. Can be assigned to processes. Users assigned roles granting access to privileges and programs. Enable a role via password to gain its privileges. Similar to access matrix. Protection can be applied to non-file resources.
Revocation of Access Rights	Options to remove the access right of a domain to an object: Immediate vs. delayed, Selective vs. general, Partial vs. total, Temporary vs. permanent. Access List revocation: simple — search and delete entry; immediate, general or selective, total or partial, permanent or temporary. Capability List revocation: must locate capability in system first before it can be revoked.
Capability Revocation Methods	(1) Reacquisition — periodic delete; require and deny if revoked. (2) Back-pointers — set of pointers from each object to all capabilities of that object; delete all (used in Multics). (3) Indirection — capability points to global table entry which points to object; delete entry from global table (not selective, used in CAL). (4) Keys — unique bits associated with capability generated when created; master key associated with object; key must match master key for access; revoke by creating new master key.
Language-Based Protection	Specification of protection in a programming language allows high-level description of policies for allocation and use of resources. Language implementation provides software for

Term	Definition / Details
	protection enforcement when automatic hardware-supported checking unavailable. Interprets protection specifications to generate calls on whatever protection system is provided by hardware and OS.
JVM-Based Protection	Protection handled by the Java Virtual Machine. A class is assigned a protection domain when loaded by JVM. Protection domain indicates what operations the class can and cannot perform. If a library method performs a privileged operation, the stack is inspected to ensure the operation can be performed. Stack inspection determines level of trust of currently running code at runtime. Example: untrusted applet's call to get() succeeds because trusted URL loader asserts the privilege of opening specific URL (lucent.com).
Stack Inspection	Technique for determining level of trust of currently running code at runtime in JVM. When privileged operation requested, call stack is inspected to ensure all callers in the chain have required permissions. The untrusted applet's get() call succeeds because the trusted URL loader uses doPrivileged to assert opening permission.

⚖ Section 2 — Comparisons & Differentiations

Protection vs. Security

Aspect	Protection	Security
Definition	Mechanism for controlling access of processes/users to OS-defined resources	Defense against internal and external attacks
Focus	Internal access control — who can do what to what	External and internal threats: DOS, worms, viruses, identity theft
Mechanism	Access matrix, domains, capabilities, ACLs	Cryptography, firewalls, intrusion detection, authentication
Scope	Within OS resource management framework	Broader — includes network and physical security

4 Access Matrix Implementation Options

Option	Approach	Advantage	Disadvantage
Global Table	Ordered triples <domain, object,	Simple to understand	Large (won't fit in memory); hard to

Option	Approach	Advantage	Disadvantage
	rights-set> in one table		group objects
Access Lists (ACL)	Per-object list of <domain, rights-set> pairs	Corresponds to user needs; easy to determine who accesses object	Every access must be checked; non-local for domain; slow with many objects
Capability Lists	Per-domain list of objects + allowed operations (capability = "secure pointer")	Fast access; useful for localizing domain info	Revocation is inefficient; must locate capability first
Lock-Key	Objects have locks (bit patterns); domains have keys; access if key matches	Effective, flexible; keys passed freely; easy revocation	More complex than ACL or capability alone

Access List vs. Capability List

Aspect	Access List (ACL)	Capability List
Matrix view	Column of access matrix (per object)	Row of access matrix (per domain)
Question answered	Who can access this object and how?	What can this domain access and how?
Revocation	Simple — delete entry; immediate, selective, total/partial, permanent/temporary	Difficult — must locate capability in system first
Information locality	Non-local for domain (must search all objects to find domain's rights)	Localized — all in domain's capability list
Examples	Unix rwx bits, Windows ACLs	Hydra OS, Cambridge CAP computer

4 Capability Revocation Methods

Method	How It Works	Used In
Reacquisition	Periodically delete all capabilities; domain must re-request; denied if revoked	General approach
Back-pointers	Each object has pointers to	Multics

Method	How It Works	Used In
	all its capabilities; delete all pointed-to capabilities	
Indirection	Capability → global table entry → object; revoke by deleting table entry (not selective)	CAL
Keys	Capability has key; object has master key; match required; revoke by creating new master key	Flexible systems

Static vs. Dynamic Least Privilege

Aspect	Static	Dynamic
When set	Fixed during life of system or process	Changed by process as needed
Mechanism	Pre-determined privilege assignments	Domain switching, privilege escalation
Flexibility	Less flexible	More flexible
Example	Process always runs with same user privileges	sudo, setuid, RBAC role activation

Rough-Grained vs. Fine-Grained Protection

Aspect	Rough-Grained	Fine-Grained
Simplicity	Simpler, easier to manage	More complex, more overhead
Least privilege	Privileges in large chunks	More precise — smaller privilege units
Protection level	Less protective	More protective
Example	Unix: user privileges or root only	File ACLs, RBAC roles (Solaris)

Section 3 — Enumeration Questions

3 Unix Domain Switching Mechanisms

3 Unix Domain Switching Mechanisms

1. **setuid bit — via File System** — Each file has domain bit (setuid). When executed with setuid=on: user-id set to file owner's ID. Resets to original when execution completes.

2. **su command — via Password** — Temporarily switches to another user's domain when that domain's password is provided.

3. **sudo command — via Commands** — Executes specified command in another domain if original domain has privilege or correct password is given.

4 Special Access Rights in Access Matrix

4 Special Access Rights in Access Matrix
1. **Owner of Oi** — Controls who gets access rights to object Oi. Applicable to objects.
2. **Copy (*)** — Allows copying access right from Oi to Oj (denoted by *). Copies within same column (same object). Applicable to objects.
3. **Control** — Di can modify Dj's access rights. Applicable to domain objects.
4. **Transfer/Switch** — Allows switching from domain Di to domain Dj. Models domain switching in the matrix.

4 Access Matrix Implementation Options

4 Access Matrix Implementation Options
1. **Global Table** — Ordered triples <domain, object, rights-set>. Simple but very large; hard to group objects; won't fit in main memory.
2. **Access Lists for Objects (ACL)** — Each column = per-object list of <domain, rights-set>. Can have default set. Corresponds to user needs. Every access must be checked — slow with many objects.
3. **Capability Lists for Domains** — Each row = per-domain list of objects + allowed operations (capability). Protected by OS — never directly accessible by domain (like a "secure pointer"). Revocation is inefficient.
4. **Lock-Key Mechanism** — Objects have locks; domains have keys. Access if key matches a lock. Keys passed freely between domains. Easy revocation. Compromise between ACL and capability.

4 Revocation Options (General — Applies to Any Method)

4 Revocation Options (General — Applies to Any Method)
1. **Immediate vs. Delayed** — Immediate: revocation takes effect right away. Delayed: revocation takes effect after some time.
2. **Selective vs. General** — Selective: revoke for specific domain only. General: revoke for all domains.

3. Partial vs. Total — Partial: revoke only some access rights. Total: revoke all access rights.
4. Temporary vs. Permanent — Temporary: access can be granted again later. Permanent: access is removed forever.

4 Capability Revocation Methods

4 Capability Revocation Methods
1. Reacquisition — Periodic delete; domain must re-request; denied if revoked.
2. Back-pointers — Pointers from each object to all its capabilities; delete all. Used in Multics.
3. Indirection — Capability → global table entry → object. Delete entry from global table to revoke. Not selective. Used in CAL.
4. Keys — Unique bits associated with capability. Master key on object. Revoke by creating new master key. Policy: who can create/modify keys?

Trade-offs Among Access Matrix Implementations

Trade-offs Among Access Matrix Implementations
1. Global Table is simple — But can be very large and difficult to group objects.
2. Access Lists correspond to user needs — But determining rights for a domain is non-local and slow. Every access must be checked — slow with many objects/rights.
3. Capability Lists localize domain information — But revocation of capabilities can be inefficient.
4. Lock-Key is effective and flexible — Keys passed freely; easy revocation.
5. Most systems use combination — First access → search access list; if allowed → create capability + attach to process; additional accesses → use capability; after last access → destroy capability.