Lecture 12

File Systems, Protection & Course Review

File Allocation · i-nodes · Protection Rings · Security Goals · Stage 4 · Summary

SENG 21213 · Computer Architecture & Operating Systems

Learning Objectives — By the end of this lecture you should be able to:

- 1. Compare contiguous, linked (FAT), and indexed (i-node) file allocation methods for different operations
- 2. Apply Unix i-node multi-level indirection to compute maximum file sizes
- 3. Explain the x86 protection ring model and describe user-to-kernel ring transitions
- 4. Apply Unix permission bits to determine access for owner, group, and others
- 5. State the four OS security goals (availability, confidentiality, integrity, authenticity) with examples
- 6. Synthesise all 12 lectures by tracing a system call through all layers of the OS

Section 1 · File Systems

📖 **Stallings Reference**

- Chapter 12: File Management
- Section 12.1 — Overview, Section 12.2 — File Organisation, Section 12.4 — File Directories

1.1 File Concepts and Attributes

A file is a named, persistent collection of related data stored on secondary storage. It is the basic unit of long-term OS storage management.

Attribute	Description	Example
Name	Human-readable identifier; may be case-sensitive or insensitive	kernel.c, README.md, NTLDR
Identifier (inode)	Internal OS identifier — not human-readable; uniquely identifies file within file system	inode 1234
Type	Regular file, directory, symbolic link, block device, character device, pipe, socket	S_IFREG, S_IFDIR, S_IFLNK
Location	Pointer to file data on disk: device ID +	(dev=sda1, first_block=0x4A20)

Attribute	Description	Example
	starting block address	
Size	Current file size in bytes; may also track allocated block count	2,048 bytes; 1 block
Timestamps	Created (ctime), last modified (mtime), last accessed (atime)	2025-09-01 09:14:22
Protection	Access control: owner, group, others × read, write, execute (Unix) or ACL (Windows)	rwxr-xr-- (754 octal)
Owner	UID and GID of file owner	UID=1000 (suresh), GID=1000

1.2 File Allocation Methods

Method	Mechanism	Advantages	Disadvantages
Contiguous	Files occupy consecutive disk blocks. Directory holds (start_block, length).	Simple; fast sequential AND random access; minimal seek time.	External fragmentation; fixed maximum file size; difficult to grow.
Linked (FAT)	Each block has a pointer to the next. Directory holds head pointer. FAT (File Allocation Table) keeps all pointers in memory.	No external fragmentation; files can grow easily.	Poor random access (must follow chain); pointer overhead; FAT itself can be large.
Indexed (i-node)	Each file has an index block (i-node) containing an array of data block pointers.	Fast random access (one indirection); no external fragmentation; supports large files via multi-level indirection.	Wasted index block for tiny files; two disk reads per data access (index + data).

★ Unix i-node Multi-Level Indirection

- Direct pointers: 12 × (4 KB/block) = 48 KB maximum without indirection
- Single indirect: 1 pointer to a block of 1024 pointers (on 4-byte pointer, 4KB block) → 1024 × 4KB = 4 MB
- Double indirect: 1024 × 1024 × 4KB = 4 GB
- Triple indirect: $1024^3 \times 4KB = 4$ TB
- Total max file size ≈ 4 TB (32-bit block numbers) or much larger with 64-bit ext4.
- Most files are small — 99% of files fit entirely in direct + single-indirect pointers.

1.3 Directory Structure

Organisation	Description	OS Example
Single-level	All files in one flat directory. Simple but no organisation.	Early MS-DOS root directory
Two-level	Separate directory per user. Avoids	Early UNIX /home/user/

Organisation	Description	OS Example
	username conflicts between users.	
Tree-structured	Arbitrary depth of subdirectories. Absolute and relative paths.	Modern Linux, Windows, macOS
Acyclic graph	Allows shared directories via hard links. Reference counting for deletion.	UNIX hard links: ln file link
General graph	Allows cycles via symbolic links. Requires cycle detection for traversal.	UNIX symlinks: ln -s

Section 2 · Protection, Security, and Privilege Rings

2.1 x86 Protection Rings

Ring	Name	Access Level	What Runs Here
Ring 0	Kernel mode	All privileged instructions; full hardware access; can modify page tables, load GDT/IDT, manage interrupts.	OS kernel, device drivers
Ring 1	Supervisor	Partial privilege (rarely used)	Some OS architectures (OS/2, Multics) — device drivers in some designs
Ring 2	Supervisor	Partial privilege	Rarely used in practice
Ring 3	User mode	Cannot execute privileged instructions; cannot access kernel memory; must use system calls.	All user application code

★ Ring Transitions

- User → Kernel: via SYSENTER (fast) or INT 0x80 (legacy) system call instruction. CPU automatically switches rings, loads kernel SS:ESP, saves user SS:ESP on kernel stack.
- Kernel → User: via SYSEXIT or IRET. CPU restores user SS:ESP from kernel stack, switches to Ring 3.
- Security principle: OS ALWAYS validates system call arguments BEFORE acting — the kernel does not trust user-provided pointers or values.
- Your OS bootloader + kernel_entry.asm runs in Ring 0. User processes (when you implement them) run in Ring 3.

2.2 Access Control — Unix Permission Model

Unix simplifies ACLs into three permission groups per file, each with three permission bits:

Group	Who	Permission Bits	Example (rwxr-xr--)
Owner (u)	The user who owns the file	r (read=4), w (write=2), x (execute=1)	rwx = 4+2+1 = 7
Group (g)	Members of the file's group	r (read=4), w (write=2), x (execute=1)	r-x = 4+0+1 = 5
Others (o)	All other users on the system	r (read=4), w (write=2), x (execute=1)	r-- = 4+0+0 = 4

2.3 Four Security Goals

Goal	Description	Threat It Addresses	Mechanism
Availability	System and data remain accessible to authorised users.	DDoS attacks, hardware failure, ransomware	Redundancy, rate limiting, backups
Confidentiality	Data cannot be read by unauthorised parties.	Eavesdropping, side-channel attacks, privilege escalation	Encryption (TLS, AES), access control, memory isolation
Data Integrity	Data cannot be modified without authorisation.	SQL injection, man-in-the-middle, malware	Hash-based verification (SHA-256), digital signatures, write permissions
Authenticity	Identities of users, processes, and messages are verified.	Impersonation, replay attacks, session hijacking	Certificates (PKI), Kerberos, HMAC, MFA

Section 3 · OS Assignment — Stage 4 and Course Review

★ Milestone 4: RAM Disk File System

- ramdisk.c — Allocate a contiguous region of physical memory (e.g. 1 MB at 0x200000). Implement: rd_read(uint32_t block, uint8_t *buf) and rd_write(uint32_t block, uint8_t *buf) for 512-byte blocks.
- fs.h — Define: superblock_t (magic, block_count, inode_count, free_bitmap_offset), inode_t (type, size, direct[8], single_indirect), dirent_t (inode_num, name[28]).
- fs.c — Implement: fs_init() (write superblock + clear inodes + mark all blocks free), fs_create(path), fs_open(path)→fd, fs_read(fd,buf,n), fs_write(fd,buf,n), fs_close(fd), fs_unlink(path).
- Shell integration: add 'ls', 'touch filename', 'cat filename', 'write filename text' commands.
- Test: touch hello.txt; write hello.txt 'Hello, SENG OS!'; cat hello.txt → displays 'Hello, SENG OS!'

3.1 Full Course Summary

Lecture	Topic	Core Concept	OS Milestone
L01	Computer System Fundamentals	MAR/MBR registers; 9-step interrupt sequence	—

Lecture	Topic	Core Concept	OS Milestone
L02	Memory Hierarchy & I/O	Hit ratio formula; DMA 7-step process	—
L03	Data Representation	2's complement; IEEE 754 single-precision encoding	—
L04	Sequential Circuits	D flip-flop; FSM design; CISC vs RISC	—
L05	Processor Architecture	CPI formula; Amdahl's Law; 8 addressing modes	—
L06	Pipelining & Parallelism	5-stage pipeline hazards; 2-bit predictor; Flynn's taxonomy	—
L07	Assembly & HW/SW	cdecl calling convention; x86 boot sequence; CR0	—
L08	OS Overview	3 objectives; 7 services; ISA/ABI/API; 4 evolution eras	Stage 0
L09	Process Control	8 PCB fields; 5-state/7-state models; 7-step context switch	Stage 1
L10	Threads & Concurrency	ULT vs KLT; race conditions; Dijkstra's 6 requirements	Stage 2
L11	Semaphores & Deadlocks	3-semaphore producer-consumer; 4 Coffman conditions; Banker's	Stage 3
L12	File Systems & Protection	i-node indirection; protection rings; 4 security goals	Stage 4

3.2 Emerging Trends

Trend	Description
RISC-V	Open, royalty-free ISA. Growing commercial adoption (SiFive, StarFive, ESWIN). Enables custom processors without ARM/Intel licensing fees. Ideal for RISC study.
Non-Volatile Memory (NVM)	Intel Optane (3D XPoint): byte-addressable, persistent, ~300 ns latency (vs 80 µs for NVMe SSD). Sits between DRAM and SSD. Challenges OS storage stack assumptions.
Hardware Security	Intel SGX: hardware-isolated enclaves for sensitive code/data. ARM TrustZone: secure/normal world partition. TEEs protect against OS-level attackers.
Memory-Safe OS	Microsoft developing parts of Windows kernel in Rust. Redox OS (Rust microkernel). Eliminates buffer overflow and use-after-free vulnerabilities at compile time.
Containerisation	Linux namespaces + cgroups = Docker containers. One kernel; isolated user-space instances. OS abstractions evolving for cloud-native workloads.

Section 4 · Examination Practice

✎ Exercise 12.1 [14 marks] — File System

- (a) [4] A Unix i-node has 12 direct pointers, 1 single-indirect, 1 double-indirect, 1 triple-indirect. Block size = 4 KB. Pointers are 4 bytes. (i) Maximum file size using direct pointers only? (ii) With single-indirect? (iii) With double-indirect? (iv) With triple-indirect? Show all calculations.
- (b) [4] A directory /project contains files: main.c (17 KB), data.bin (200 KB), readme.txt (3 KB). All

files use indexed allocation with the i-node structure above. (i) How many index blocks (i-nodes) are needed for each file? (ii) How many data blocks for each file? (iii) What is the total disk space consumed including metadata?

- (c) [6] Compare the three file allocation methods for the following operations. Rank from best (1) to worst (3) and justify: (i) Sequential read of a 1 GB file. (ii) Random read of byte at position 500,000,000 in a 1 GB file. (iii) Appending 4 KB to the end of a 100 MB file. (iv) Deleting a 500 MB file. (v) Detecting file system corruption after a power failure.

✎ Exercise 12.2 [12 marks] — Protection and Security

- (a) [3] A file has permissions: -rwxr-x--- (and owner=alice, group=developers). (i) Can alice write to the file? (ii) Can bob (member of 'developers') execute the file? (iii) Can carol (not in 'developers') read the file? (iv) What octal number represents these permissions?
- (b) [4] A user-space program does: int *ptr = (int*)0x00000000; *ptr = 42; (i) On a modern OS, which mechanism prevents this from succeeding? (ii) What exception is raised? (iii) What does the OS do with the faulting process? (iv) What would happen on an 8086 (Real Mode, no protection) running MS-DOS?
- (c) [5] Your kernel runs in Ring 0. A user process at Ring 3 wants to read() from a file. (i) How does the user process transfer control to the kernel? Name the specific x86 instruction. (ii) What happens to the privilege level during the system call? (iii) The kernel receives the file descriptor and buffer pointer from user space. Why can't the kernel simply trust the pointer? (iv) What two checks must the kernel perform on the user-provided buffer pointer before accessing it?

✎ Exercise 12.3 [9 marks] — Synthesis / OS Assignment

- (a) [3] Trace the complete path of a write('Hello') system call through your SENG OS Stage 0–4 implementation: starting from the user shell, through the system call interface, kernel, file system layer, RAM disk, and back. Name each module/function called at each step.
- (b) [3] Your Stage 3 PMM uses a bitmap with 1 bit per 4 KB frame. The kernel occupies 256 KB at the start of memory. The system has 4 MB of physical memory. (i) How many frames total? (ii) How many frames are marked used at init? (iii) How large is the bitmap in bytes?
- (c) [3] Describe how you would extend your Stage 2 mutex to support priority inheritance. (i) What data must the mutex store? (ii) When a high-priority thread blocks on a locked mutex, what priority change occurs and to which thread? (iii) When is the priority restored? Why is priority inheritance important for real-time systems?