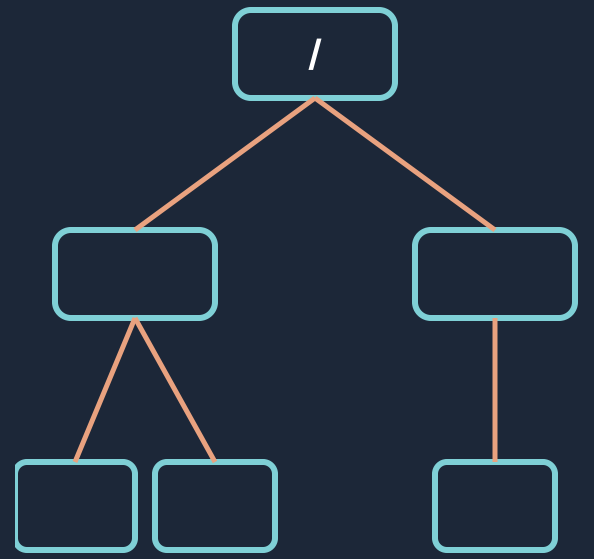
OPERATING SYSTEMS · LECTURE 13

File System Implementation

Files, directories and allocation · Windows & Linux



Lecturer: Aidana Karibayeva

01 What is a file system?

The structure an OS uses to **store, access and organise** files on a storage device — HDD, SSD or USB drive.

Two parts

Files
store related data

Directory structure
organises and describes all files

File names

`filename.cpp`

Extension shows the format. Linux names are **case-sensitive**. Once named, a file is independent of its creator — copied or e-mailed, it keeps its name.

02 File attributes

ATTRIBUTE	MEANING
Name	symbolic name — the only human-readable attribute
Identifier	unique number of the file inside the file system
Type	needed when the system supports several file formats
Location	pointer to the device and place where the file is stored
Size	current size (bytes, words...), possibly the maximum size
Protection	who may read, write, execute
Date, time, user ID	creation, last modification, last use

03 Six basic file operations

The OS provides a **system call** for each operation.

Create

① find free space ② add an entry to the directory

Write

name + data; writes at the **write pointer**

Read

name + where to put the next block; uses the **read pointer**

Reposition (seek)

move the current-position pointer — no actual I/O

Delete

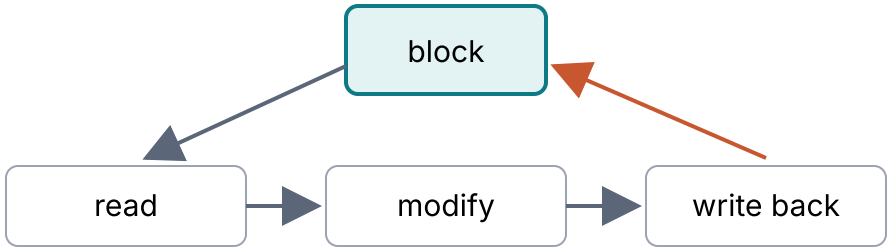
free the space and remove the directory entry

Truncate

erase contents, keep attributes; length becomes 0

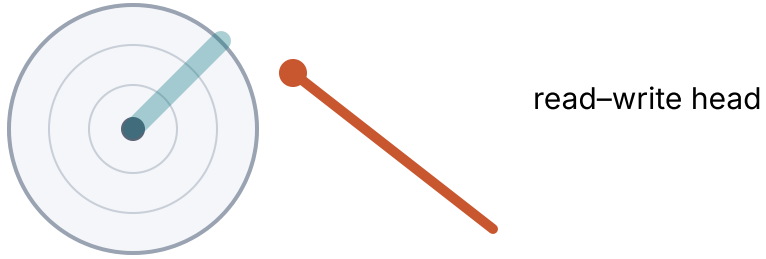
04 Why disks suit file systems

Rewrite in place



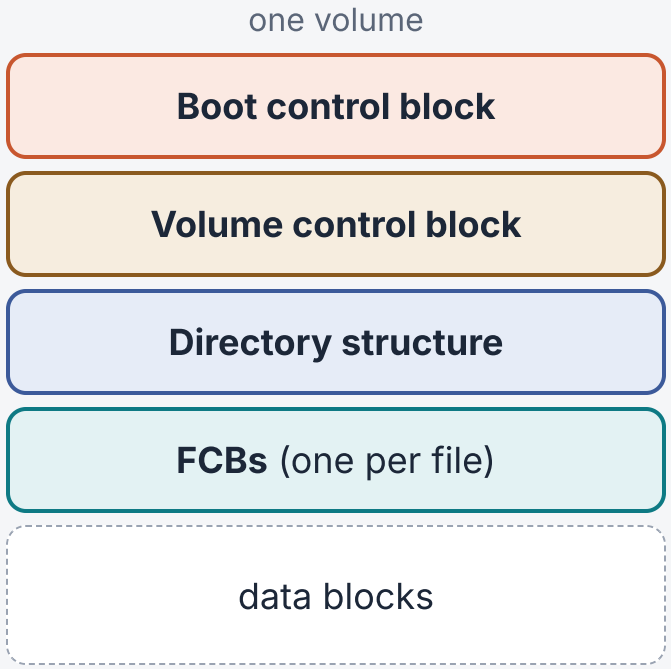
Read a block, change it, write it back to the **same place**.

Direct access



Any block can be reached directly → files can be read **sequentially or at random**.

05 On-disk structures



STRUCTURE	HOLDS	UFS (UNIX)	NTFS (WINDOWS)
Boot control	info to boot an OS from the volume; empty if no OS	boot block	partition boot sector
Volume control	number and size of blocks, free-block count and pointers	superblock	master file table
Directory	file names and their numbers	names + inode numbers	master file table
FCB	permissions, owner, size, location of data blocks	inode	row in MFT

06 Common file systems

FAT
WINDOWS (OLD)

File Allocation Table; simple, used on USB sticks

NTFS
WINDOWS

permissions, compression, encryption

ext
LINUX / UNIX

Extended File System (ext2, ext3, ext4)

HFS
MACOS (OLD)

Hierarchical File System

APFS
MACOS / IOS

Apple File System, the modern Apple format

07 File types and extensions

TYPE	EXTENSIONS	PURPOSE
Executable	`exe, com, bin`	ready-to-run machine code
Object	`obj, o`	compiled, not linked
Source code	`c, java, pas, asm`	program source
Batch	`bat, sh`	commands for the shell
Text / document	`txt, doc, tex, rtf`	text and word-processor files
Archive	`zip, tar, arc`	several files compressed into one
Multimedia	`mpeg, mov, rm`	audio / video
Markup	`xml, html`	structured text
Library	`lib, a, so, dll`	routines for programmers
Print / view	`pdf, jpg, gif`	printing or viewing

08 Directories

A directory is itself a file that holds information about other files.

Entry contains

`name` `type` `address`
`current length` `max length`
`last accessed` `last updated`
`owner ID` `protection`

Operations

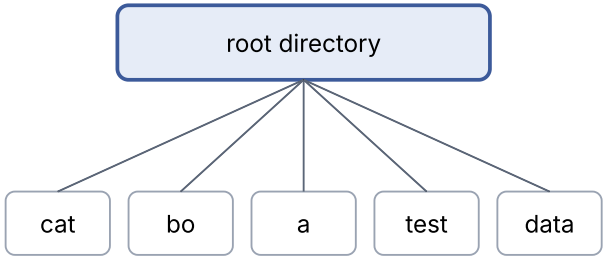
`search` `create` `delete` `list`
`rename` `traverse`

Why directories

Efficiency	files are found faster
Naming	same name for different files of different users
Grouping	e.g. all Java programs, all games

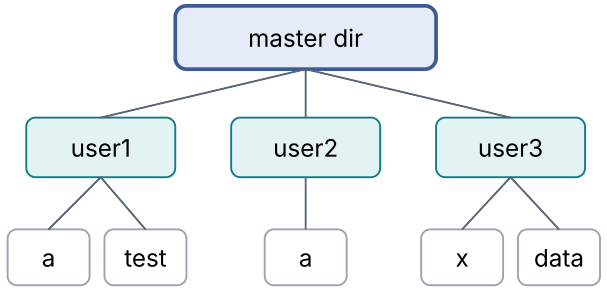
09 Directory structures

Single-level



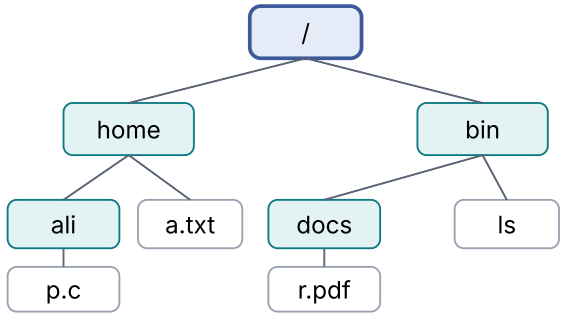
× no two files with one name
× no grouping

Two-level



✓ path name per file; same name for different users
✓ efficient search

Tree-structured



✓ efficient search + grouping
✓ absolute and relative paths

10 Contiguous allocation

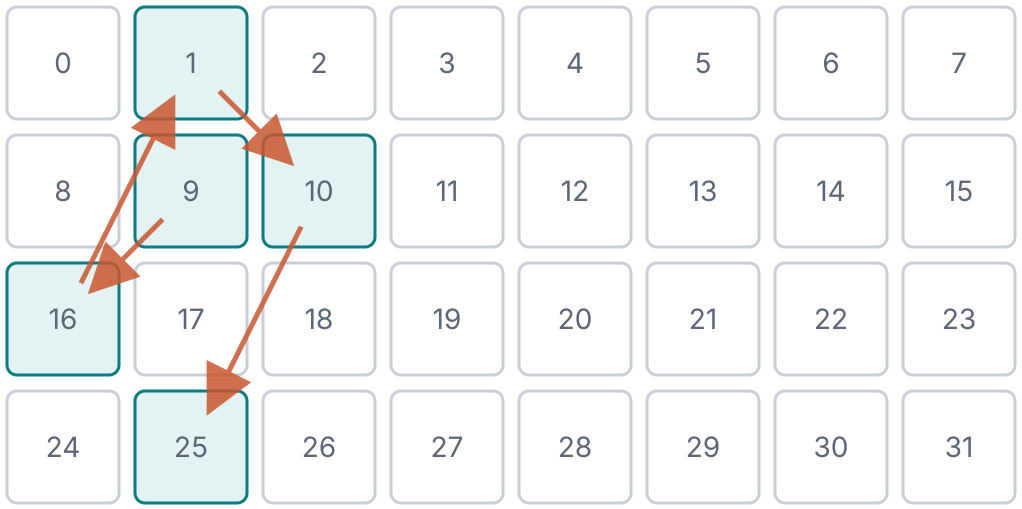


FILE	START	LENGTH
count	0	2
tr	14	3
mail	19	6
list	28	4

Block *i* of a file starting at *b* is at **b + i − 1**. Fast sequential and direct access.

External fragmentation → needs compaction. File size must be declared at creation.

11 Linked allocation



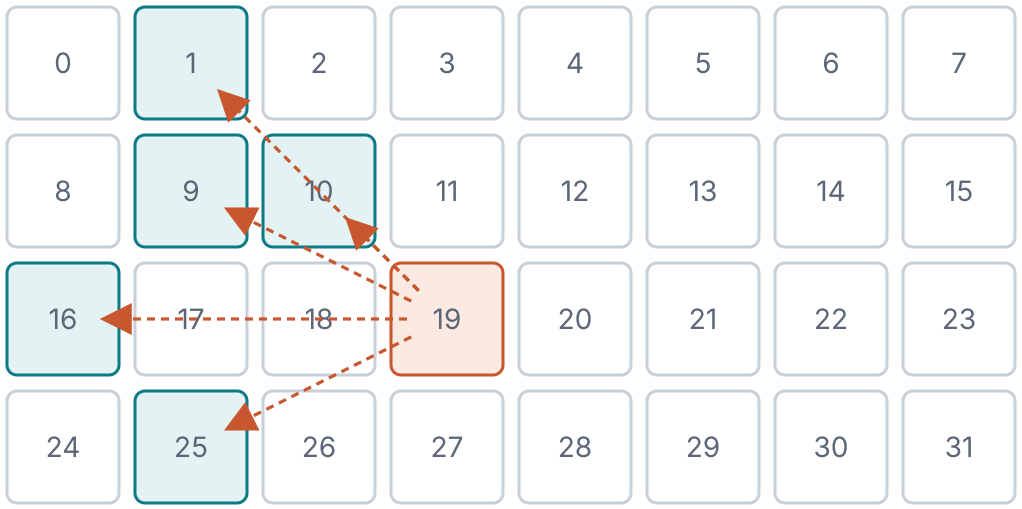
Each block holds a **pointer to the next**. Directory entry: `jeep · start 9 · end 25`

`9` → `16` → `1` → `10` → `25` → `nil`

✓ no external fragmentation; file can grow while free blocks exist

× internal fragmentation in the last block · pointer overhead in every block · a lost pointer truncates the file · **sequential access only**

12 Indexed allocation



An **index block** lists all blocks of the file. Directory entry: `jeep · index 19`

9
16
1
10
25
−1

✓ no external fragmentation · **sequential and direct access** — the most popular method

13 Allocation methods compared

	CONTIGUOUS	LINKED	INDEXED
Directory stores	start + length	start (+ end)	index block
Direct access	yes	no	yes
External fragmentation	yes	no	no
File growth	hard — size declared in advance	easy	easy (until index full)
Overhead	none	pointer in each block	one index block per file