File System Implementation

Revision:

- **VSFS On-Disk Layout**

A typical Very Simple File System (VSFS).

- Superblock: file system metadata: total inodes, total data blocks, addresses.
- Inode bitmap: 1 = allocated inode, 0 = free.
- Data bitmap: 1 = allocated block, 0 = free.
- Inode table: array of inodes; each inode ~128-256 bytes.
- Data blocks/region: user file data.

- **Inodes**

Each file has one inode identified by inode number.

Inode Field	Meaning
size	file size in bytes
block pointers	12 direct, 1 indirect, 1 double, 1 triple
permissions, UID, GID, timestamps	metadata

- **File Size from Pointer Structure**

Block size = B
Pointer size = 4 bytes; pointers per indirect block = B / 4

Example for 4 KB blocks, given:

Direct pointers = 12, then 12 × 4 KB
Indirect block = 1 × (1024 × 4 KB)
Double indirect = 1024 × (1024 × 4 KB)
Triple indirect = 1024 × (1024 × (1024 × 4 KB))

Then, total max file size:

$= (12 + 1024 + 1024^2 + 1024^3) \times 4\ KB$

- **Directories**

A directory is a file that stores: (name, inodeNumber). It includes special entries: **.** (dot for self) and **..** (dot-dot for parent). Deletion leaves gaps and a new entry can reuse space.

- **Access Paths**

Steps for: *open("/foo/bar")*

➔ Read inode of root
➔ Read directory data, find "foo", get inode #
➔ Read inode of "foo"
➔ Read "foo" directory, find "bar", get inode #
➔ Read inode of "bar", permissions check, return file descriptor

Steps for: *read(fd)*

➔ Consult inode
➔ Find block address
➔ Read data block
➔ Update inode access time

Steps for: *write(fd)*

If write allocates a new block:

➔ read data bitmap, write data bitmap
➔ read inode, write inode
➔ write data block

Steps for: *File creation*

➔ Expensive process that involves inode bitmap, inode table, directory data, and directory inode.

- **Caching & Buffering**

Caching important blocks in memory:
• Static partitioning: Allocate a fixed size of memory for caching file system blocks.
• Dynamic partitioning: Allocate memory more flexibly across virtual memory and file system.

Technique	Benefit	Risk
Standard caching	Eliminates repeated I/O (static/dynamic)	cache misses

Write buffering	Batching, scheduling, avoids unnecessary writes	crash may lose recent writes
fsync()	forces durability	slower

- **Important Formulae:**

Type	Key Formulas
Inode, block / sector lookup	inodeOffset = i × inodeSize blockNo = inodeOffset / blockSize sector = (inodeOffset + inodeStart) / sectorSize
File size from pointer model	(direct + indirect + double + triple) × blockSize
Max pointers per indirect block	blockSize / pointerSize (e.g., 4096 / 4 = 1024)
Double/Triple expansion	multiply levels: 1024^2, 1024^3

Questions:

Q1. A filesystem has block size = 4 KB, Inode size = 256 B, sector size = 512 B, and the Inode table starts at byte offset = 0. Find which block and sector contain inode number 9000.

Ans. inodeOffset = 9000 × 256 = 2,304,000 bytes
block = 2,304,000 / 4000 = 576
sector = 2,304,000 / 512 = 4500

Q2. Block size = 4 KB. File uses 12 direct blocks + 100 pointers in its indirect block. Find file size.

Ans. 12 × 4096 = 49,152 bytes
100 × 4096 = 409,600 bytes
Total = 458,752 bytes

Q3. Block size = 4 KB, pointer size = 4 B. Inode supports: 12 direct + single + double + triple. What is the maximum file size this architecture can hold?

Ans. $(12 + 1024 + 1024^2 + 1024^3) \times 4$ KB

= 4,402,345,721,856 bytes ≈ 4.4 TB

Q4. Given, Block size = 4096 B, pointer size = 4 B. How many pointers fit in a single indirect block?

Ans. 4096 / 4 = 1024 pointers

Q5. For `open("/os/notes/week1.txt")`, how many inode and directory block reads (no caching) occur?

Ans.
root inode read
root dir block read -> find "os"
inode for /os read
/os directory block read -> find "notes"
inode for /os/notes read
/os/notes directory block read -> find "week1.txt"
inode for file read

Total = 7 reads

Q6. Given, Block size = 4 KB, inode size = 256 B. The inode table occupies 5 blocks. How many inodes exist?

Ans. One block holds 4096 / 256 = 16 inodes
5 blocks: 5 × 16 = 80 inodes