

Exercise walk-through

7.5

File systems

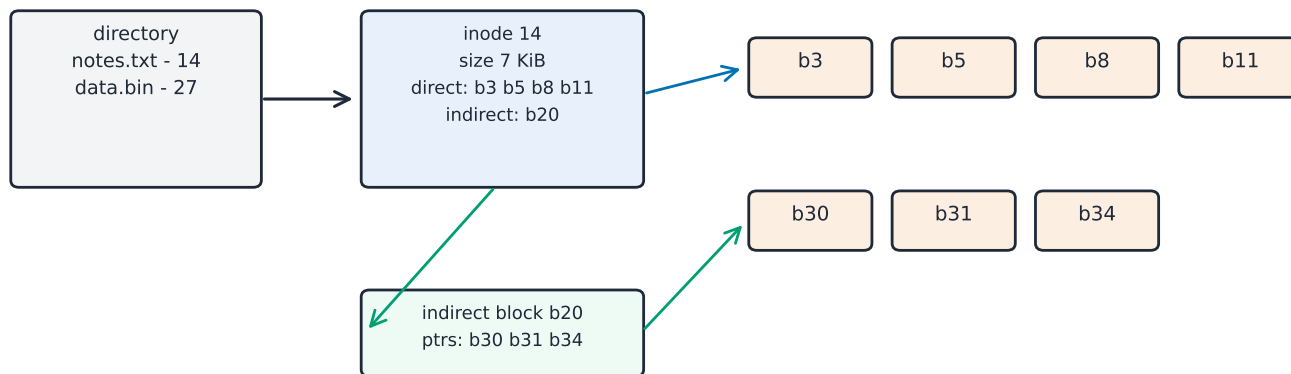
University Computer Science

File systems: names to blocks

- 1 **Sizes** 1 KiB blocks, 4 direct + 256-pointer indirect: max file $(4 + 256) \times 1 \text{ KiB} = 260 \text{ KiB}$; 4 KiB blocks give about 4 MiB.
- 2 **Price** Block 6 needs the indirect block first: two reads for one logical block; a 100-block file costs 101 reads.
- 3 **Crash** Data-then-inode orphans a block; inode-then-data lies about size - **journaling** writes the intent first and replays.

File system layout

name to inode to blocks — direct first, indirect for more



1 KiB blocks, 256 pointers per indirect block: max file $(4 + 256) \times 1 \text{ KiB} = 260 \text{ KiB}$

Directory entries to inodes to blocks, indirect for the big files.

Exercises 2.1-2.3

2.1 With 2 KiB blocks and 512 pointers per indirect block, compute the max file with 4 direct pointers. [3]

2.2 Reading block 9 (4 direct pointers): how many disk reads, and why? [3]

2.3 Does deletion erase the data? Answer and explain in one line. [3]

Exercises 2.1-2.3

2.1 With 2 KiB blocks and 512 pointers per indirect block, compute the max file with 4 direct pointers. [3]

2.1

- With 4 direct pointers: $(4 + 512) \times 2 \text{ KiB} = 1032 \text{ KiB}$ ✓.

2.2 Reading block 9 (4 direct pointers): how many disk reads, and why? [3]

2.2

- Two: the indirect block (pointer $9 - 4 = 5$), then the data block ✓.

2.3 Does deletion erase the data? Answer and explain in one line. [3]

2.3

- No – deletion clears the pointers; the bytes survive until the blocks are reused ✓.

Exercise 3.1

3.1 A file system uses 4 KiB blocks, 8 direct pointers, and one indirect block holding 1024 pointers.

Compute the maximum file size. Compute the reads for block 12 with a cached inode (name the pointer index). Compute the total reads for a sequentially read 3000-block file (indirect block read once). State what changes if the indirect block is evicted mid-read.

[8]

Exercise 3.1

3.1 A file system uses 4 KiB blocks, 8 direct pointers, and one indirect block holding 1024 pointers.

Compute the maximum file size. Compute the reads for block 12 with a cached inode (name the pointer index). Compute the total reads for a sequentially read 3000-block file (indirect block read once). State what changes if the indirect block is evicted mid-read. [8]

3.1

- Max file: $(8 + 1024) \times 4 \text{ KiB} = 4128 \text{ KiB} \approx 4.03 \text{ MiB}$ ✓.
- Block 12: two reads – the indirect block (pointer $12 - 8 = 4$), then the data ✓.
- 3000-block file: $3000 + 1 = 3001$ reads ✓.
- Mid-read eviction: the indirect block is re-read on the next beyond-direct access – occasional extra reads, not incorrect ones ✓.

Exercise 3.2

3.2 An append writes three things: the data block, then the free-list bit, then the inode size.

State what is lost if the crash lands between data and free-list, and between free-list and inode. Name the write order rule that never loses user data (only space). Explain how journaling removes the ordering constraint, and its per-write cost in one line. [7]

Exercise 3.2

3.2 An append writes three things: the data block, then the free-list bit, then the inode size.

State what is lost if the crash lands between data and free-list, and between free-list and inode. Name the write order rule that never loses user data (only space). Explain how journaling removes the ordering constraint, and its per-write cost in one line. [7]

3.2

- Between data and free-list: nothing user-visible – an orphan block the free list no longer offers (lost space) ✓.
- Between free-list and inode: the size lies – reads past the old end return garbage ✓.
- Rule: data before metadata (inode last) – a crash then loses only space, never user bytes ✓.
- Journaling writes the intent first and replays it after the crash, so any order recovers; cost: one extra journal write per update ✓.