Reading from and Writing to .omni File

File Handle Management

Pattern Used: fstream

Pros

Fixed-size structures enable direct offset calculation without scanning.

Data region blocks are uniformly sized, allowing index-based addressing.

Read Patterns

Sequential read of entire table, filter active entries into memory structures.

Block Chain Traversal (File Read)

Follow linked list via `nxt` pointers until chain terminates `nxt = 0`.

Serialization/Deserialization

Approach: Raw Binary Reinterpretation

- **Zero marshalling overhead** - direct memory copy
- **Fixed structure sizes** - predictable disk layout
- **Fast I/O** - single read/write per structure

Structure-Specific Handling

OMNIHeader

- **Written:** Once during `format()`, updated in destructor
- **Read:** Once during `init()`
- Contains all filesystem-level configuration

UserInfo

- **Written:** Entire table rewritten in destructor
- **Read:** Entire table loaded into AVL tree at init
- `is_active` flag distinguishes valid from empty slots

FileEntry

- **Written:** Entire table rewritten in destructor
- **Read:** Loaded into `files[]` vector at init and AVLTree<FileMetaData>
- `type = 3` marks unused entries, `type = 2` marks deleted

BlockHdr

- **Written:** When allocating/modifying file blocks
- **Read:** When traversing block chains

Handling File Growth

Block Allocation on Create

Strategy: Allocate all blocks upfront when file created.

Block Allocation on Edit (`editFile`)

Strategy:

1. Calculate blocks required for new size
2. Allocate additional blocks if current chain insufficient
3. Extend chain via `BlockHdr.nxt` pointers

Write Pattern:

- Seek to starting block based on `index / block_size`
- Write across block boundaries, allocating new blocks as needed
- Update `BlockHdr.size` to reflect actual bytes used

Block Reuse on Delete

Strategy: Walk chain, mark each block free, return indices to free list.

Free Space Management

Free List Structure

All blocks start free, offsets computed and stored in `free_segments` during `init()`.

Allocation Strategy: First-Fit

Characteristics:

- O(1) allocation
- No fragmentation analysis
- Simplest implementation

Deallocation Strategy

Simply appends to free list; no coalescing or sorting.

Memory vs. Disk Per Operation

Loaded at Init (Stays in Memory)

Structure	Size	Purpose
`vector<FileEntry> files`	max_files × 320 bytes	All file/directory entries
`vector<Tree> Root`	max_files × ~24 bytes	Directory tree structure
`AVLTree<UserInfo> users`	~max_users × 320 bytes	All user accounts
`AVLTree<FileMetadata> Mds`	~max_files × 384 bytes	Path → metadata mapping

`vector<uint32_t> free_segments`	num_blocks × 4 bytes	Free block indices

Example Memory Usage (max_files=10,000, max_users=100):

- Files: ~3.2 MB
- Tree: ~0.24 MB
- Users: ~0.032 MB
- Metadata: ~3.84 MB
- Free segments: ~40 KB (assuming 10,000 blocks)
- **Total: ~7.4 MB**

Read from Disk Per Operation

`readFile`

1. Traverse block chain from `FileEntry.inode`
2. For each block: read `BlockHdr` + data bytes
3. **Disk reads:** 1 + (file_size / block_size) seeks + reads

`createFile`

1. No reads - only writes to allocated free blocks
2. **Disk writes:** needed_blocks × (BlockHdr + data)

`getMetadata`

- **Zero disk I/O** - served from `Mds` AVL tree

`listDirectory`

- **Zero disk I/O** - served from `Root[]` and `files[]`

`getStats`

- **Zero disk I/O** - aggregates in-memory structures

Written to Disk Per Operation

File Operations (`createFile`, `editFile`, `deleteFile`)

- **BlockHdr + data** for each affected block

Shutdown (Destructor)

1. Write `OMNIHeader` (512 bytes)
2. Write user table (max_users × 320 bytes)
3. Write file entry table (max_files × 320 bytes)

Total Shutdown Write: ~512 + (100 × 320) + (10,000 × 320) = ~3.2 MB for typical config