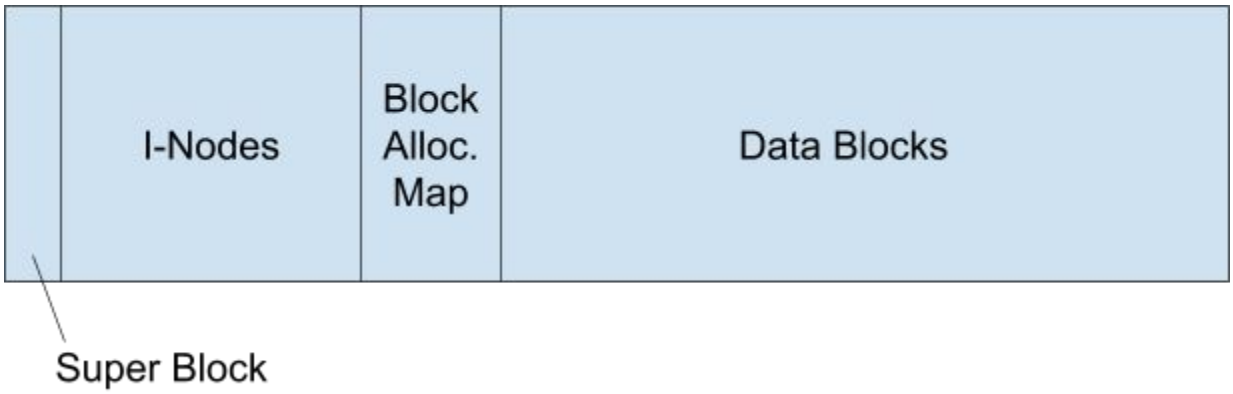
Curtis Belmonte
COS 318, Fall 2015
Princeton University

Final Project: Unix File System

Overall File System Layout

The file system I implemented is divided into four logical sections. In order, these are the super block, i-nodes, block allocation map, and data blocks. These sections are contiguous on disk and each serve a particular function. The details of each section are outlined below.



Super Block

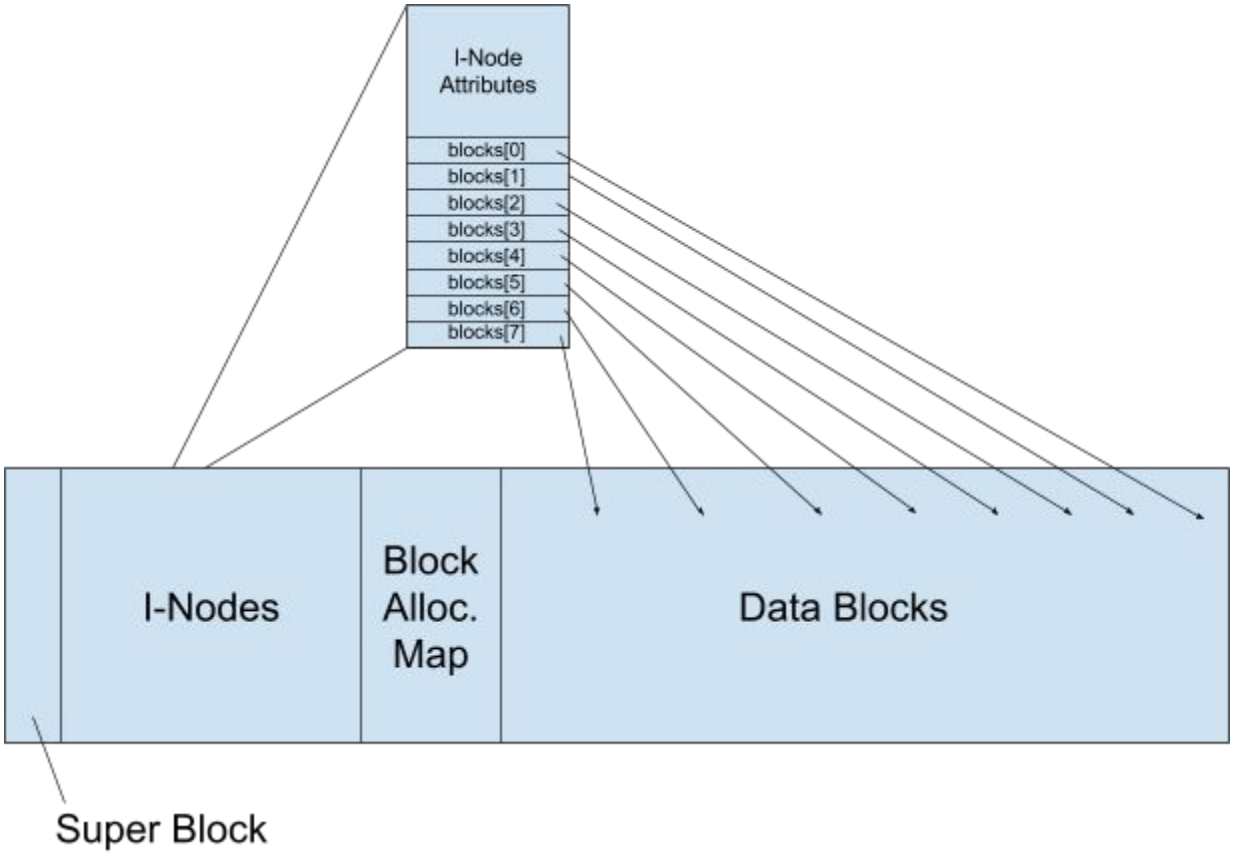
The super block contains various persistent information about the file system, such as the overall file system size, the starting block of each section, and the number of blocks each section contains, all of which are set when the system is initialized. The super block also contains a “magic number” at a particular address, which indicates to processes attempting to access the file system that the disk has been formatted for use.

As the name suggests, the super block takes up exactly one block of disk space, and it is located at block index 0. In my file system implementation, the super block struct also remains in memory in order for persistent system info to be accessible to file system functions without needing to make many disk I/O operations.

I-Nodes

Each distinct file or directory in the file system has a corresponding i-node that defines various metadata about that file. In particular, a file’s i-node tracks its size, type (file or directory), the number of extant links and open file descriptors to it, and addresses of the data blocks on disk

where data for the file are stored. An i-node and the data blocks associated with it will not be freed until there are no longer any links or file descriptors to that inode.



In my file system implementation, an i-node can contain up to 8 direct links to data blocks where file data are stored, for a maximum file size of 4096 bytes. My i-node struct is padded to 32 bytes, allowing exactly 16 i-nodes to be stored in a single disk block.

Free Block Data Structure and Management (Block Allocation Map)

The block allocation map consists of a set of boolean flags that indicate whether each data block is in use by some i-node. Each data block is represented in the map by a single 0 or 1 byte at the corresponding index. A data block is allocated and set aside for an i-node simply by setting its usage byte to 1 in the block allocation map and adding its index to the i-node's list of data blocks. Freeing a block works similarly, by setting the usage byte to 0 in the block allocation and removing it from the i-node struct to which it was assigned. When all bytes in the block allocation map are set to 1, there are no free blocks remaining in the file system.

In my file system implementation, there are 1536 data blocks that can be allocated. Therefore, the block allocation map takes up precisely three contiguous blocks (512×3) on disk.

Files and Directories (Data Blocks)

All non-metadata for files and directories in the file system is stored in data blocks. Files consist of a series of bytes stored within data blocks specified by their i-node. This data can be read from or written to a file by first opening a file descriptor table entry for the file. The file descriptor table contains an entry for each file that is currently open for reading or writing, along with positional information regarding where in the file to write to/read from.

Directories are also implemented as files in my file system. The data blocks for a directory contain entries for each of the files and subdirectories contained in that directory. Each directory entry includes a file name and the i-node number that the entry corresponds to. My directory entry struct is padded to 64 bytes, allowing exactly 8 entries to be stored in a single disk block. These entries are kept contiguous by continually swapping the last entry in a directory with an entry that has been removed. Directory navigation through the file system begins at the root node, and the current directory i-node is stored at all times.

Link/Unlink Implementation

Linking a file refers to creating another reference to a file in the file system, such that the original file and the new link point to the same data and are indistinguishable. Unlinking is the opposite process, by which a link to a file within a directory is removed, reducing the number of links that that refer to the file. If the last link to a file is removed in this way, then the file will be deleted and its blocks will be reclaimed by the file system.

In my file system implementation, creating a link to a file means creating a new directory entry with the name of the new link, containing the same i-node value as the old file. This way, both entries refer to the same i-node, and the link count of the i-node to which they refer can be incremented. This file is found by scanning through directory entries, comparing string names.

In order to unlink a file, you remove that corresponding entry from the current directory and decrement the link count of the i-node to which it refers. If this would set the link count to 0 and there are no open file descriptors, the i-node and data blocks for the file will be freed on disk.