Akash Pathuri, arp229

Michael Elkhouri, mre66

Operating Systems Design CS 416

5/2/2022

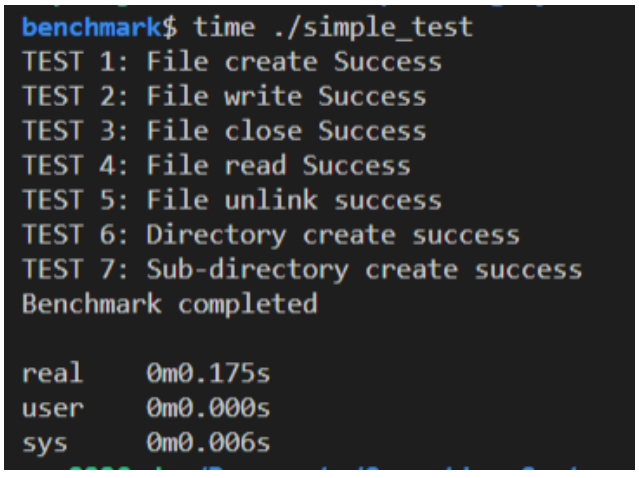
Project 4 RU File System Report

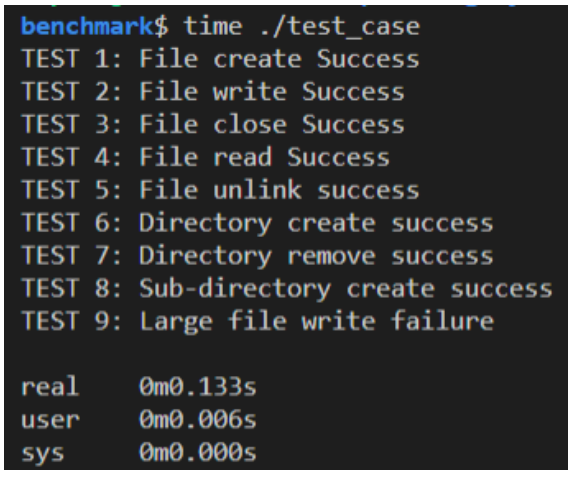
Block Usage

1. Block 0 is our superblock
2. Block 1 is the inode bitmap
3. Block 2 is the data bitmap
4. Block 3 through 66 (Max_INUM / Block_Size) will store all the inodes of the file system
5. Any block after stores the data blocks
    a. Simple test used 7 data blocks
    b. Test_case used 23 data blocks

Benchmark Time (time varied based on the performance of the computer)

1. ./Simple_test: 0.175s avg
2. ./test_case: 0.133s avg





Code Implementation

Compiling and running our code is the same way as the project described. To implement this file system library we had to research about the specific structures we needed to implement and read the function descriptions carefully.

Bitmapping:

get_avail_ino() and get_avail_blockno() are almost identical search methods. The key difference is that get_avail_ino() searches the inode indices/bitmaps and get_avail_blockno() searches the data block indices/bitmaps.

Inode Operations:

readi and writei are two very similar methods that share the first two steps of each other's schema. The difference is that readi allows the caller to hold the data inside the inode while writei modifies it.

Directory Operations

dir_find(), dir_add(), dir_remove() are 3 vital functions for the file system we are building. Finding directories proved to be a simple loop through the given directory's data block and find the specific file name given, if not found return -1. Dir_add was tricky to implement because we had to make sure we were traversing and handling each case correctly which led to plenty of debugging. dir_remove() uses dir_find as the main driver of the function and then handles the removal of the passed in directory.

get_node_by_path():

The get_node_by_path method is a key component of the filesystem as it tries to find the inode of any given path. Multiple file operations require the reference to the inode of a specific file or a directory. This method is implemented recursively.

Files System Setup:

Superblock is the main data structure that contains important information about the file system. Additionally two global variables inodes_per_block and dirent_per_block hold the number of inode structures that can fit in a block and the number of dirents structures that can fit in a block respectively.

FUSE File Operations:

The operation methods for the FUSE File systems were implemented using the steps provided for each operation in the code files.

Difficulties encountered

At each section we came across issues that would later be found through testing and connecting methods in a way where data is handled correctly and systematically. Issues like segmentation faults, syntax errors, and logic errors were common at first however as we polished each method after rigorous testing we picked up on patterns that were used in later sections. We once came across a “Software caused connection abort” error and handled this by going step by step in our logic to find where our software crashed. We found that dir_add, remove, and get_node_by_path methods were the most challenging because sometimes error messages would not help us so of course we resorted to print statements and breakpoints.