CS105 – Computer Systems Fall 2024

Assignment 9: File System Lab

Due: Tuesday, November 26, 2024 at 11:59pm

For this assignment, your goal is to port some files from a user-space filesystem to the native file system provided by the OS. As usual, you should complete this assignment with a partner; you may choose your partner for this assignment.

The starter code for this assignment is available both on the course website and on the course VM. You may complete it either on your local machine or on the course VM (the file path is `/cs105/starters/fslab.tar`.

On the server, unpacking the file with "`tar xvf /cs105/starters/fslab.tar`" will create a subdirectory named `fs-handout` containing the starter code `fs.c`, a `Makefile`, and a subdirectory `disk` containing the individual blocks of the user-level file system you will be porting the files from.

1 Background

The user-level file system that you will be porting files from uses an indexed allocation with the following details:

- The disk is comprised of 2048 256-byte blocks.
- Block 0 is the superblock, which contains the file system parameters described here.
- Block 1 is a bitmap that stores information about which inodes are currently in use.
- Block 2 is a bitmap that stores information aboout which data blocks are currently in use.
- Blocks 3 - 34 are inode blocks; each inode block stores an array of inodes.
- Blocks 35 - 2048 are data blocks; each allocated data block stores the contents of some file (either a normal file or a directory file).

Inodes in this file system have the following components:

- A 16-byte filename.
- An 8-byte filesize (i.e., a long).
- 8 direct pointers, each of which is a 4-byte value (i.e., an int) that stores a block number or 0.
- 1 indirect pointer, a 4-byte value that stores a block number or 0.
- 1 doubly-indirect pointer, a 4-byte value that stores a block number or 0.

Any indirect (and doubly-indirect) blocks simply store an array of 4-byte block numbers (i.e., ints).

Directory entries in this filesystem are 32-byte values comprised of a 16-byte filename followed by a 4-byte inode number plus some padding. Directories are simply files where the contents of the file are an array of directory entries.

2 Starter Code

I've provided starter code that implements a generic block interface (two functions `read_block` and `write_block`). I've also provided some helper functions for copying inodes to/from memory and for parsing filepaths.

3 Your Tasks

Your assignment is to port some files that are currently stored in this file system to the native file system provided by the operating system. This will be done as a series of 5 tasks:

Task 1: Your first task is to implement the ability to find the inode number of a file given an absolute file path. For now, we'll make two simplifying assumptions (1) you will only ever need to look for files that are stored in the root directory, and (2) all files are small enough that they can be stored using only the direct pointers (i.e., both the indirect pointer and the doubly indirect pointer will always be 0).

This task should be completed in two steps:

First, implement the function `search_dir_data_bloc`. This function takes three arguments: a block number, a filename, and an integer `max_entries`. This function should assume that the block specified by `data_block_num` is part of a directory and thus contains an array of directory entries. It should search the first `max_entries` directory entries in that block for one that matches the specified file name; if it finds the directory entry for that filename it should return the corresponding inode number for that file, otherwise it should return -1.

Second, implement the function `search_dir`. This function takes the inumber associated with some directory and a filename, and it should search the contents of that directory (all the datablocks in that directory) to try to find a data entry that matches the specified filename. If it finds such a directory entry it should return the corresponding inode number for that file, otherwise it should return -1. Remember that for now you can assume that all files (including this directory) use only direct pointers!

You'll see that I've provided in initial implementation of a function `search_path` that parses a filepath (assuming it has the form `/filename.ext`) and calls your `search_dir` function with inumber 2 (the number of the inode for the root directory) and filename `filename.ext`. So as long as we make the assumption that you will only ever need to look for files that are stored in the root directory, `search_path` should return the inode number of the file with the specified path. If you try running the first test case, your code should now return the correct inode number for the file `/test.txt`.

Task 2: Your next task is to implement a simple version of the function `port_from_105` that continues to make the two simplifying assumptions from Task 1. This function takes two arguments: the first argument `filepath` is an absolute path in the user level filesystem. This function should read that file from the user-level file system and save a copy to the native file system using the name `new_filename`. You should now be able to successfully port the file `/test.txt` from the user-level file system and open (and read) it.

Task 3: We will now start eliminating our simplfying assumptions. For your third task, you should no longer assume that you will only ever need to look for files that are stored in the root directory; you now

want to support files that are stored in subdirectories. To complete this task, you will need to modify the implementation of `search_path`. If you try running the third test case, your code should now return the correct inode number for the file `/example_dir/test2.txt`; you should also be able to successfuly port this file (test 4).

Task 4 (Optional): You can now eliminating the second assumption. To support large directories, implement the functions `search_dir_indirect` and `search_dir_doubly_indirect` and modify your implementation of `search_dir`.

Task 5: Finally, you need to add support for large files. Modify your implementation of `port_from_105` to support large files. You should now be able to successfully port the file `/example_dir/image.jpg` from the user-level file system and open it (you may want to download it to your local machine to open the image).

Hint: If you are not getting a correct file, you may find it helpful to run your code against the autograder on Gradescope. That will let you know where and how your file is going wrong.

Feedback

Create a file called `feedback.txt` that answers the following questions:

1. How long did each of you spend on this assignment?
2. Any comments on this assignment?

As always, how you answer these questions **will not affect your grade**, but whether you answer them will.

Submission

Use the course submission site to submit your competed program `fs.c`, copies of the three files you ported (`test1.txt`, `test2.txt`, `image.jpg`), and your feedback file. Make sure that you tag your partner as a collaborator when you submit. Also, be sure the names of all team members are *clearly* and *prominently* documented in the comments at the top of your `fs.c`.