Tiny File System Report

Peter Tilton and Nikita Volodin

3 May 2021

1 Benchmark results

Total number of blocks used:

- simple_test: 2 bitmap blocks, 102 inode blocks, and 107 data blocks used
- test_cases: 2 bitmap blocks, 103 inode blocks, 123 data blocks used

Time taken:

- simple_test single-thread total: 3.815 ms
- simple_test multi-thread total: 4.326 ms
- test_cases single-thread total: 4.784 ms (tests 1-8 only)
- test_cases multi-thread total: 4.600 ms (tests 1-8 only)

2 Compilation instructions

- -lpthread included in Makefile rules to allow using c thread library.

3 Implementation details

get_avail_ino()

- 1 Read the inode bitmap into a buffer.
- 2 Scan through this buffer for a free bit.
- 3 Set this bit.
- 4 Return the index of this bit.

get_avail_blkno()
As get_avail_ino(), but using the data bitmap instead of the inode one.

readi()

- 1 From ino, determine which block the inode is stored in, as well as its offset within this block.
- 2 Read the corresponding block to a buffer.
- 3 Copy the data at the offset into the inode pointer passed as an argument.

writei()

- 1 From ino, determine which block the inode is stored in, as well as its offset within this block.
- 2 Read the corresponding block to a buffer.
- 3 Copy the data pointed to by the inode argument into the buffer at the offset.
- 4 Write the buffer back to disk.

dir_find()

- 1 Read the inode corresponding to the ino argument.
- 2 Scan through this inode's 16 direct pointers.
- 2.1 Load the block pointed to by this direct pointer into a buffer.
- 2.2 Scan through this buffer to find the file name given as an argument.
- 2.2.1 If a match is found, copy the corresponding buffer region into the given dirent pointer.

dir_add()

- 1 Read the inode corresponding to the ino argument.
- 2 Scan through this inode's 16 direct pointers.
- 2.1 Load the block pointed to by this direct pointer into a buffer.
- 2.2 Scan through this buffer to check if the given name already is used.
- 2.3 If the name is already used, return.
- 3 Scan through the inode's 16 direct pointers again.
- 3.1 Load the block pointed to by this direct pointer into a buffer.
- 3.2 Scan through this buffer to check if there is empty space in this block.

- 3.3 If there is empty space, add a dirent there and return.
- 4 Create a new block.
- 5 Add a dirent in the new block and return.

dir_remove()

- 1 Read the inode corresponding to the ino argument.
- 2 Scan through this inode's 16 direct pointers.
- 2.1 Load the block pointed to by this direct pointer into a buffer.
- 2.2 Scan through this buffer to check if the given name exists.
- 2.3 If the name exists, delete the dirent corresponding to it.
- 3 If the name was not found, return.

get_node_by_path()

- 1 Pull out the first name in the given path.
- 2 Read the inode corresponding to the ino argument.
- 3 Scan through this inode's 16 direct pointers
- 3.1 Load the block pointed to by this direct pointer into a buffer.
- 3.2 Scan through this buffer to check if the first name exists.
- 3.3 If the name exists, get the inode associated with the dirent with this name (inode2).
- 3.3.1 If the first name in the path is also the last name of the path, copy the inode2 into the inode pointer function argument and return.
- 3.3.2 Call get_node_by_path() again, but with the path after the first name and the ino of inode2.

tfs_mkfs()

- 1 Initialize inode bitmap, data bitmap, inode blocks, data blocks, superblock, and mutex.
- 2 Create the inode associated with the root directory, write it into the start of the inode blocks.

- 3 Create the dirents associated with "." and ".." in the root directory, write them into the start of the data blocks.

tfs_init()

- 1 If disk file doesn't exist, call tfs_mkfs().
- 2 If disk file does exist, read superblock and load root directory information.

tfs_destroy()

- 1 Free global data structures.
- 2 Call dev_close().

tfs_getattr()

- 1 Get an inode from the given path.
- 2 Update the stat buffer with inode stat data.
- 3 Update stat buffer with time data.

tfs_opendir()
Checks to see if a directory exists by calling get_node_by_path().

tfs_readdir()

- 1 Get inode associated with path.
- 2 Scan through this inode's 16 direct pointers.
- 2.1 Load the block pointed to by this direct pointer into a buffer.
- 2.2 Scan through this buffer to check if the dir name exists.
- 2.3 If the name exists, call filler() using it.

tfs_mkdir()

- 1 Split path name into parent and target directory.
- 2 Get inode of parent directory.
- 3 Get the next available inode number.

- 4 Make a target inode with this ino.
- 5 Create dirents associated with "." and ".." in the target directory.
- 6 Save target inode to inode blocks and target dirents to data blocks.

tfs_rmdir()

- 1 Split path name into parent and target directory.
- 2 Get inode of target directory.
- 3 If this inode's link number is greater than 2, return.
- 4 Clear the data/inode bitmap/data blocks associated with the target directory.
- 5 Get the inode of parent directory.
- 6 Remove the link to the target directory within the parent directory.

tfs_create()
As with tfs_mkdir(), but creating a file instead of a directory.

tfs_open()
As with tfs_opendir(), but checking a file instead of a directory.

tfs_read()

- 1 Get inode associated with target file.
- 2 Scan through this inode's 16 direct pointers.
- 2.1 Determine which data blocks to read by size and offset.
- 2.2 Load disk into a corresponding part of the given buffer.

tfs_write()

- 1 Calculate which block to start writing in and using the relative offset within that block using the offset parameter and BLOCK_SIZE.
- 2 Read the required blocks from the disk and create a single buffer that we will modify.
- 3 Copy the buffer parameter's data into our new buffer, starting at the offset we calculated earlier, and write the data blocks back to disk.

- 4 Update the inode accces/modify time and size, and write the inode back to disk.

tfs_unlink()

- 1 Separate the terminal point from the rest of the file path using basename() and dirname().
- 2 Use get_node_by_path() on the full file path to get the inode of the target file.
- 3 Loop through the target-inode's direct pointers, find what data blocks it's using and free the corresponding indices in the data block bitmap.
- 4 Using the inode number of the target inode, clear the corresponding bit in the inode bitmap.
- 5 Write both bitmaps back to disk.
- 6 Call get_node_by_path() on the parent directory of the target.
- 7 Call dir_remove() to remove the directory entries for target in the parent's data blocks.

4 Reflection

Overall we were able to complete the basic functionality of TFS, meeting all of the required test cases. All functionality seems to work with block sizes as low as 512 bytes when testing manually, however it seems the benchmarks need to be configured specifically for varying block sizes, which made it difficult to find out if it would still work.

When running the large file benchmark tests, it appears that there may potentially be subtly subversive behavior when the file system reaches its space capacity for a direct pointer-only inode, **though this does not seem to cause a crash during the benchmark**. This would require further investigation and action addressing this edge case.