iLab Used: kill.cs.rutgers.edu
Single person group: Michael Zhang
NetID: mbz27

<u>Benchmark</u>

simple_test.c

Output:

```
TEST 1: File create Success
TEST 2: File write Success
TEST 3: File close Success
TEST 4: File read Success
TEST 5: File unlink success
TEST 6: Directory create success
TEST 7: Sub-directory create success
Benchmark completed
```

<u>The number of blocks used 107 (this does not include the super block, bitmaps, and inode structure). The runtime of this benchmark is 6.726 seconds.</u>

test_case.c

Output:

```
TEST 1: File create Success
TEST 2: File write Success
TEST 3: File close Success
TEST 4: File read Success
TEST 5: File unlink success
TEST 6: Directory create success
TEST 7: Directory remove success
TEST 8: Sub-directory create success
TEST 9: Large file write failure
```

<u>The number of blocks used 123 (this does not include the super block, bitmaps, and inode structure). The runtime of this benchmark is 2.834 seconds.</u>

<u>Implementation</u>

Notice: Most of the functions are implemented based on the steps in the comments (there's probably not a whole lot to explain other than the steps indicated in the tfs.c file)

<u>In memory Structures</u>

In my implementation I have three in memory structures that I use to keep track of the superblock, inode bitmap, and data bitmap (I allocate a block for each structure so I can easily write the entire structure to the disk as a whole block). In functions that I need to modify the bitmaps I simply modify the in memory bitmaps and then in order to update the bitmaps I just write the entire in memory structure to the disk.

Bitmap

get_avail_ino()

The function reads in the inode bitmap from the disk and traverses the bitmap until an available slot is found. When the available slot is found the inode's valid bit is set and the bitmap is written back to the disk. Otherwise it returns -1 (there are no more available inodes).

get_avail_blkno()

The function reads in the block bitmap from the disk and traverses the bitmap until an available slot is found. When the available slot is found the block's valid bit is set and the bitmap is written back to the disk. Otherwise it returns -1 (there are no more available data blocks).

Inode

readi()

The function calculates the block number that the given inode is in on the disk, and the offset the inode is located in that block. The function then reads in the block from the disk and returns a copy of the inode at the offset.

writei()

The function calculates the block number that the given inode is in on the disk, and the offset the inode is located in that block. The function then reads in the block from the disk and copies the given (parameter) inode to the inode at the offset in the block then writes the block back to the disk.

Directory and Namei

dir_find()

The function reads in the inode at the given inode number and checks if the inode is valid. The function then iterates through all the data blocks in the direct ptr array in the given inode. For each data block the function will iterate through all of the dirents checking if any of the dirents has the given name fname. If the function finds a valid dirent with the name fname then it will copy the dirent to the dirent parameter and return 1. Otherwise it will return -1 (dirent does not exist).

dir_add()

The function checks if the given inode type is a dirent then it will iterate through all of the data blocks pointed by the inode to check if there is a duplicate dirent with same name fname. If there are no duplicate dirents then the function will iterate through the data blocks to find an available dirent or will allocate an additional data block if there are not available dirents in the given data blocks (it will return -1 if there are no more available data blocks). Then it will add a new dirent to the data block with the given parameters and writes the block to the disk. It will update the directory inode which will update the size, and time the directory inode was accessed and write the updated inode to the disk.

dir_remove()

The function will check if the given inode type is a dirent then it will iterate through all of the data blocks pointed by the inode and tries to find the dirent that has the same name fname. If the function finds the dirent it will set the valid bit to 0 and writes the data block to the disk and updates the directory inode which will update the size, and the time the directory inode was accessed and write the updated inode to the disk.

get_node_by_path()
The function will iterate through the directories in the specified path using dir_find() to check if the directories are all valid and copy the terminating inode to the parameter inode.

Tiny File System

tfs_mkfs()
The function initializes the diskfile, the superblock, the inode and data bitmaps, and the root directory with its stats. It also allocates a data block to add the . and .. directories to the root directory (the root directory's parent points back to itself).

tfs_init()
The function will call tfs_mkfs() if the diskfile has not been initialized. But if the diskfile has already been initialized then it will read in the superblock and the bitmaps to the in memory data structures.

tfs_destroy()
The function simply frees all in memory data structures and closes the diskfile.

tfs_getattr()
The function will check if the given path is valid and if it is then it will return back the stat structure. Otherwise it will return -ENOENT.

tfs_opendir()
The function will check if the given path is valid and if it is then it will return 0 otherwise it returns -1.

tfs_readdir()
The function will check if the given path is valid then it will iterate through the data blocks of the path's inode. Then it will use the filler function to fill the buffer with allo the valid dirents in the inode.

tfs_mkdir()
The function will separate out the parent directory path and the target directory name then it will get the parent directory inode. It will then allocate a new inode for the target directory (also increments the number of hard links) and calls dir_add() to add a new directory to the parent directory. Then the function will initialize the inode of the target directory and stat structure of the target directory's inode. Also it will add the . and .. directory where the .. directory points back to the parent directory.

tfs_rmdir()
The function will separate out the parent directory path and the target directory name then it will get the inode of the target directory. It will check if the inode is a directory and if there are exactly two directory entries which should be the . and .. directories (since we can only remove a directory if the directory is empty). Then it will set the valid bits in the data bitmap to 0 of the data blocks that the target inode pointed to and writes the data bitmap to the disk. It will then get the parent directory (also decrements the number of hard links) and calls dir_remove() on the target directory to remove the directory entry from the parent directory.

tfs_create()
The function will separate out the parent directory path and the target directory name then it will get the parent inode directory and allocate a new inode number for the target file. It will then call dir_add() to add the target file to the parent directory. It will also initialize the inode structure and stat structure and write the inode to the inode structure on the disk.

tfs_open()
The function will simply check if the given path is valid and if it valid it will return 0 otherwise it will return -1.

tfs_read()
The function will get the inode of the target file and checks if the inode is a file. Then it will calculate which data block to start at and the index in that block to start copying the data. It will then iterate through the data blocks pointed by the inode and copy the data byte by byte into the buffer starting from the offset.

tfs_write()
The function will get the inode of the target file and checks if the inode is a file. Then it will allocate more data blocks to the inode target file if the number of blocks allocated to the inode is insufficient to hold the data (total size needed to hold the data is offset+size). After that it will then iterate through the data blocks and copy the data byte by byte into the data block starting from the offset.

tfs_unlink()
The function will separate out the parent directory path and the target directory name then it will get the inode of the target file. It will iterate through all of the block numbers and set all of the valid bits to 0 in the data bitmap and write the data bitmap back to the disk. It also set the valid bit to 0 in the inode for the target file and wrote the inode bitmap back to the disk. Next the function gets the parent directory and calls dir_remove() which will remove the target file from the parent directory.

Compile and Run
I used the math library for this project so I added the -lm flag to both the project makefile. I also added the pthread library for mutexes.