CS416 Project 4 FAQs: Tiny File System using FUSE Library

Just some possible FAQs. Please excuse the typos. This document will be updated when necessary and you will receive an email notification from Sakai.

Debugging

What is the -s flag in the command $ ./tfs -s /tmp//mountdir ?

-s

Run single-threaded instead of multi-threaded. This makes debugging vastly easier, since gdb doesn't handle multiple threads all that well. It also protects you from all sorts of race conditions. Unless you're trying to write a production filesystem and you're a parallelism expert, I recommend that you always use this switch.

More info: https://www.cs.hmc.edu/~geoff/classes/hmc.cs135.201109/homework/fuse/fuse_doc.html

open/create/initialize related questions

Do we need to handle flags for tfs_open()? Like prohibit writing of a file is opened for RDONLY

- No. In this project for tfs_open, you are only required to just check if the file exists.

Should we return -1 if a file does not exist but the user is trying to open the file

- Instead of returning -1 upon a file not being found, try including errno.h and return the ENOENT error code which is the error code that corresponds to "no such file or directory found" like the following code snipper. You can look at is more here. http://man7.org/linux/man-pages/man3/errno.3.html

```
#include <errno.h>
....

if(inode not found){
    return ENOENT;
}

// or

if(inode not found){
   return -ENOENT;
}
```

Disk, Read, and Write

Can we modify bio_read() and bio_write()

- No, you should be modifying segments of blocks in the individual functions that call bio_read/write. File systems (most of the time) operate in blocks instead of segments since blocking makes it easier to manage the disk and prevent external fragmentation due to multiple separated segments (and in turn also improving disk read performance since the read/write head doesn't have to wait for the disk to rotate to the sector it's looking for). Once you read the block and store it in memory, you should then manipulate the block since it will be much faster to do so.

- In this project, the file system is already (and fully) in memory (/tmp) so the performance impact of reading in sectors is minimal but this is obviously not the case for traditional file systems. The goal of the project is to create a file system that operates as if it is using a disk instead of a flat buffer in memory.

What does in-memory structure represent?

- In a file system, every disk structure like superblock, inode, dirent, bitmap, and any other structure stored in the disk must first get created in memory, modified as required, and only then written to disk. Assume you would like to modify something in the superblock structure. You can maintain a global superblock in the heap (memory), make the necessary modification, and then write the modified superblock to the disk. Note that the superblock does not change after the initialization.
- Similarly, let's assume you would like to create a new inode. You would first create an inode in the heap (using malloc), make the necessary changes to the inode, and then write the inode to the disk.
- The inode and superblock that you allocated represent the in-memory structures.

Can we store our superblocks, inode bitmap and data bitmap, and inode tables as global variables?

- You can keep some structures, such as superblock and bitmap, global and modify them directly in memory instead of reading from disk every time. BUT, after you make the change, you must DEFINITELY write them back to the disk. If you do not write them back after making a change, the file system will not be persistent. You must ensure that the in-memory structure is consistent with the on-disk format.

How to update superblock, inode bitmaps and data bitmaps?

- First regarding the superblock, when doing tfs_mkfs(), you will allocate space for a superblock and then we have to write a block to the disk for our superblock. Since the superblock will be the first block in the disk, anytime you update the superblock, you will need to use the bio_write() function to write our superblock to block 0.
- Same goes for updating inode bitmaps and data bitmaps, and inode blocks and data blocks. You would have to use bio_write().

The stat structure has a lot of possible fields to fill, and I was wondering which ones are necessary?

```
int(* fuse_operations::getattr)(const char *, struct stat *, struct fuse_file_info *fi)
```

- Get file attributes.
- Similar to stat(). The 'st_dev' and 'st_blksize' fields are ignored. The 'st_ino' field is ignored except if the 'use_ino' mount option is given. In that case it is passed to userspace, but libfuse and the kernel will still assign a different inode for internal use (called the "nodeid").

What is the units of Size? Is it the number of valid dirents in a director? The number of blocks used in direct_ptr?

- So the size member refers to the size of the actual file. You can use blocks or bytes, it's up to you. Just remember if you happen to use one or the other, you may have to convert it to fill in the following fields in struct stat in tfs_getattr():

```
off_t     st_size;        /* Total size, in bytes */
blksize_t st_blksize;     /* Block size for filesystem I/O */
```

What is the inode stat thing, and what is its purpose?

- We use struct stat to record the time related to an inode (the "vstat" field in struct inode in tfs.h). When a file is created or modified, we need to update this field in its inode. For more information about struct stat, please see "https://linux.die.net/man/2/stat". In this project, we ONLY use st_mtime and st_atime in struct stat. You could use function time() to set st_mtime and st_atime. For more information and example about time(), please see "https://linux.die.net/man/2/time".

What exactly is a link (i.e., link count) and what benefit does knowing how many there are have?

- This "link count" value is the number of different directory entries that all point to the inode of a file or subdirectory. In the case of a regular file, the link count is the number of hard links to that file (which is generally one). However, for a directory, even for the empty directory's inode, you will have a link count 2. In Linux or OSx you can find the number of links for a file or directory using the following command.

```
$ ls -a file1
$ mkdir emptydir1    //create empty directory.
$ ls -a emptydir1
```

Why 2 links for directories?

- Every directory also contains the "." link that points back to itself. So the minimum value of 2 links per directory. Every subdirectory has a "..." link that points back to its parent, incrementing the link count on the parent directory by one for each subdirectory created. So, when you created "emptydir1", the link count of the parent would have incremented by 1.

Where else to update the link values?

- See the stat structure carefully, which has a st_nlink member variable. This must be also updated. Return type of tfs_init The return value of this function is available to all file operations in the private_data field of fuse_context. It is also passed as a parameter to the destroy() method (See the link mentioned in Resources section 7 of the description). If you don't have anything to pass as a privata_data (or store some state to be used by all FS operations), you could just return a NULL (in tfs_init).

What is a direct_ptr in the inode struct, and the member, vstat?

- The direct_ptr(s) are a list of pointers to the data blocks of this particular file represented by this inode. By pointers, we mean they store the block numbers of the data blocks. vstat is simply a struct holding file info that you will have to keep for getattr. Look at the man page for struct stat here (http://man7.org/linux/man-pages/man2/stat.2.html)

If a direct_ptr has 16 elements (entries), should we assume every file/directory is going to take up 16 data blocks?

- No, 16 represents the maximum blocks that could be supported with direct pointers in this project. There could be files that take fewer than 16 data blocks.

Root directory

Does "/" represent the root directory? Do we need a dirent struct for root?

- Yes, the initial "/" is the root, you do not necessarily need a dirent struct for it but you may need an inode to represent it.

Will all paths start with '/'? Is there a chance to have something like "~/someDir/someDir2" or "../" or "."?

- While in real file systems this is absolutely possible, for this project you don't have to worry about the relative path or path w.r.t to current working directory. When we test, we will use full path strings.
- If you are curious about how path resolution works in a real file system, here are some details, http://man7.org/linux/man-pages/man7/path_resolution.7.html

What should the mode be for the root directory node?

- You can probably set it to something like

```
S_IFDIR | 0755;
```

General directory questions

What is the difference between data blocks and directory blocks?

- The data blocks of file inodes simply contain users data (i.e., contents of a file) and no dirents (directory entries are stored) The data blocks of directory inodes contain dirents structure.

How to represent and store directories?

- A directory is also an inode with direct pointers pointing to data blocks. In a directory's data blocks, you store instances of struct dirent. The maximum number of dirents stored in a directory is limited by the number of data blocks that can be used by directory inode, block size, and the size of each dirent. Because we are only using direct data block pointers (16 blocks in this project), you can easily calculate the number of dirents you could store in 16 data blocks.

Where are dirents stored?

- Data blocks

How do we tell if the data block corresponds to file content or a dirent?

- So let's assume a scenario where we have the dirent we're looking for already ("/dev" for example). The dirent will have the inode number and the name "dev" (or the full path, not completely sure on this one but most likely just the basename). Using this inode number we can find the inode corresponding to this directory. How do we know that this dirent/inode is a directory? When making the inode initially, there should already be a type field that indicates this (it might be easiest to setup a bit field and some macros to quickly determine this (sort of like S_IS_REG - you can't use this macro but you can make your own macros). So once we get the inode we want, check for the type.
- Once we have the type we can determine what the data blocks contain. If the type isn't a regular file (directory), then we know the data blocks only contain dirents. If it is a regular file, then we know the data blocks only contain file content. To get the file you want from a directory inode, just follow the same logic we just went through in this example.

- In regards to direct pointers (let's ignore indirect for now), you'll notice in tfs.h the inode struct has an array of direct pointers (it's an array of ints so think about what you're storing here, even though we call them "pointers"). Each direct pointer leads to a data block. You should fill each data block with dirents or file content before moving to the next one. So a directory inode will have each data block containing multiple dirents (not one per data block).
- If you want a more detailed explanation of all of this, you should read the textbook (chapter 39-40).

Iterating through the block: How do we know how many data blocks the current directory is using ?

- In order to know how many data blocks a current directory is using, you can just keep track of the size of the directory like you would a regular file as you add dirents and allocate more data blocks to hold those dirents.
- As for which direct ptrs are valid or not, you can set the direct_ptr to some default value when not in use, like 0, as we know block zero we reserved for the superblock.

Because there is no structure in the data block, should we make the desired data block "struct aligned"?

- Initially, when a new data block is allocated for a directory, it should be formatted so that all entries are set to invalid, that way the entires can be skipped when you have to iterate over the directory entries, like in dir_find. Then, when the first directory entry of a block is added, then only that one entry should be marked as valid.
  - As for how many dirents you will have to have per block is something you can figure out by finding the size of a struct dirent, and seeing how many dirents you can fit inside a single 4k block.

Can a user call rmdir on a non-empty directory? If so, would we be required to recursively remove the entries inside of the directory?

- If you do rmdir on a non-empty directory, Linux would return the following message. So, you would be returning a similar message.

```
"rmdir: failed to remove ‘temp’: Directory not empty"
```

Is there a reason for the size of the dirent to be a number as ugly as 214? May we increase the size of the name buffer to make the total size of the struct a round number like 256? This would make the math easier when reading/manipulating directories.

- It is okay to increase/pad the struct dirent to 256 bytes for this project. You will not lose points.
- Beyond this project, just because doing the math is complex, one cannot introduce internal fragmentation. Imagine a disk with 10-million directories, so that's around 400MB of disk space. While real FSes do try to align the structures in powers of 2, unfortunately, it is not always possible.

Do we need to support directories that contain a lot of files so that one data block doesn't hold all it's entries?

- If you are only supporting direct pointers and not doing the extra credit, a directory shoyuld be able to support as many directory entries as can fit in 16 data blocks

File read and writes

What is the purpose of offsets?

- The offset it simply the starting offset within the whole file. For example, let's say I have a 16kb file called "foo" that I want to read or write to. Since each block is 4k, this means foo's inode points to 4 data blocks. Now lets say you want to write a 5 byte string "hello" to file foo starting at offset 6144.

```
const char *str = "Hello";
tfs_read("foo", str, 5, 6144, asdasdas)
```

- So starting at offset 6144 within the file, I should start to copy the string hello. Since offset 6144 would lie in the middle of the second block (6144 - 4k in the first block = offset 2048 in the second block), that is where I would have to start writing. So I would read in the second block, copy the data, then write the second block back to disk.

Locking

Do we need to implement some kind of locking to FS functions?

- Yes. The project does not require making your locks fine-grained.

Destroy operations

In tfs_destroy the hint says "De-allocate in memory data structures". What exactly should we do here?

- Release all the malloc'd memory (if you did allocate).

Submission

Is okay for us to modify block.c/block.h?

- If you are adding helper functions, as long as you state it clearly in the report and upload them alongside with the required files, that's fine.

Is there a specific output that we have to be looking for when we run our code in the simple_test.c file?

- There are seven things that simple test.c (take a look at the simple test code carefully to see what it's doing) so you should so hopefully is correctly implemented, you should see success for each case like

```
Test 1: Success

Test 2: Success

etc...
```