Jason Dao (jnd88)

NOTE: I changed dirent so the size of it is 256 bytes!!!!!!!!!!!!!!!

Assumptions: I was confused on how many data blocks a directory would have(the instructions on the project said 16 but I think the professor said only 1), so for my implementation, a directory has up to 16 blocks to store dirents.

Multi thread: The sample benchmark(simple_test.c,) took 0.002551 seconds. Test_cases.c took 0.002928 seconds.

Single thread: The sample benchmark(simple_test.c,) took 0.002568 seconds. Test_cases.c took 0.009863 seconds.

From simple_test.c, 102 inode blocks were used and 175 data blocks were used at the end(Some blocks were reused after unlink and rmdir, I allocated 68 data blocks for all inodes, bitmaps, and superblock from the start so inodes and data blocks would not be mixed up). Test_cases.c also had 102 inode blocks used and 175 data blocks used (I excluded test case 9 and 10 as I did not do the extra credit).

Implementation:

- To implement get_avail_ino, I first read the inode bitmap from the disk. Next, I went through the indexes, and if there was a bit with a 0, I set it to 1 and returned that index.
- To implement get_avail_blkno, I first read the data block bitmap from the disk. Next, I went through the indexes, and if there was a bit with a 1, I set it to 0 and returned that index.
- To implement readi, first I got the offset by doing the modulus of the inode number by the number of inodes that can fit in one block. I got the block number of the inode by simply doing division of the inode number by the number of inodes that can fit in one block. Next, I read from that block and saved it to the inode.
- To implement writei, first I got the offset by doing the modulus of the inode number by the number of inodes that can fit in one block. I got the block number of the inode by simply doing division of the inode number by the number of inodes that can fit in one block. Next, I read from that block and saved it in memory where I wrote to the offset in memory and later I wrote it to the data block.
- To implement dir_find, using readi, I got the inode of the parent directory. Next going through the directory data blocks, if a block was valid and the name matched fname, it was written into dirent and exited. If I went through all the valid directory blocks and nothing was found, -1 was returned.
- To implement dir_add, going through all the data blocks of the directory, I searched for both an open dirent spot as well as if the name was already in there. If the name was found, -1 was returned. If there was no more open spots, I allocated another block (I checked if there was less than 16 to allocate first, if there weren't, I returned -1), and then I added the directory entry to the open slot.
- To implement dir_remove, going through all the data blocks of the directory, I searched for fname in the directory entries and if there was, I removed it from there. If not, I returned -1.
- To implement get_node_by_path, for every time in the path name there was a "/", I would do dirfind on that name to see if a directory entry existed (besides the root folder, which the entry would equal 0). If by the end of the string there was a directory entry for every folder in that path, I returned 1, if not -1.

Jason Dao (jnd88)

- To implement rufs_mkfs, I sent the first data block for a superblock, which I set to a default values in block.h. Next, I set the second block for the inode bitmap and the third for the data block bitmap. I set up the folder values for the root folder and put its inode in the fourth data block. I also set the next 64 blocks for the inodes so the inodes and the data block would not be jumbled up. I also set up the directory entries for the root folder and set each entry to invalid.
- For rufs_init, after getting a mutex, I opened the diskfile and checked if the magic number was correct in the superblock, if not, I would call rufs_mkfs. If it was, I loaded the bitmaps to memory.
- For rufs_destory, after getting a mutex, I cleared the memory of the superblock, and the bitmaps.
- For rufs_getattr, after getting a mutex, first I checked using getnodebypath to see if the file existed, if not I returned -ENOENT. Next I set the stat values to what was saved in its inode from getnodebypath.
- For rufs_opendir, after getting a mutex, I checked using getnodebypath to see if the directory existed. If it did I returned 0, if not -1.
- For rufs_readdir, after getting a mutex, I read the folder from its directory entry block and traversed the entry and added it to the buffer with its stat included as a parameter.
- For rufs_mkdir, after getting a mutex, I ran getnodebypath on its parent to see if it existed. If it did, I set the values in the struct of the inode to that of a folder (the correct mode, the size of a directory entry for . and ..). I also found a data block and inode block by running get_avail_ino and get_avail_blkno. I also set up the new directory block and set the entries in there to invalid.
- For rufs_rmdir, after getting a mutex, I checked if the path existed using getnodebypath, if it did, I also found its directory block and checked if there were any entries that existed that was not . or … If it did, I returned an error. Next I unset its bit in the inode and data block bitmaps.
- For create, after getting a mutex, I ran getnodebypath on its parent to see if it existed. If it did, I set the values in the struct of the inode to that of a file (the correct mode, size to 0.). I also found a data block and inode block by running get_avail_ino and get_avail_blkno. I also set up the new data and memset the chars in there to 0.
- For rufs_open, after getting a mutex, I checked using getnodebypath to see if the file existed. If it did I returned 0, if not -1.
- For rufs_read, I first checked if the file existed or if the offset or size exceeded 16 blocks, if it did, I returned -1. Next I found the data block and for every BLOCK_SIZE block, I memcpy it into the memory. Once the size was reached, I memcpy that to buffer.
- For rufs_write I first checked if the file existed or if the offset or size exceeded 16 blocks, if it did, I returned -1. Next I found the data block and for every BLOCK_SIZE block, I memcpy the buffer contents into the data block incrementally. Once the size was reached, I exited and updated the modified time and the size to offset+buffer for the stat attributes.
- For rufs_unlink, after getting a mutex, I checked if the path existed using getnodebypath, If it did not, I returned an error. Next I unset its bit in the inode and data block bitmaps.

No difficulties, I think I was able to resolve most issues, no additional steps.

Jason Dao (jnd88)