John Quinn

jjq23

Report on RUFS

Implementation

1. For getting the next available inode and block numbers, I called bio_read on the areas of the file corresponding to the bitmaps for both inode and data blocks respectively. If one was available for either, they are sent back.
2. For both reading from and writing to the inodes, I translated the inode values into a corresponding block number within the inode blocks. After this, I call bio_read to read the data from the block into the global buffer. In readi, I write this data to the inode struct, and for writei, I take the data from the inode struct and write it back into the buffer, before calling bio_write to put the data back into the disk.
3. The dir_find, dir_add, and dir_remove functions were implemented as so:
    a. Dir_find first makes a new inode and calls readi with it and the passed in ino to get an inode for the current directory. Afterwards, in a for loop, I iterate throughout the directory's direct_ptrs to see if there's a link to a dirent block. If there is, I iterate through the dirents to see if a dirent for fname exists, and if so, I call readi on the dirent structure and return 0. If not, I return -1.
    b. For dir_add, I first call dir_find to see if the passed in file already exists, and if so, I return -1. Otherwise, I iterate through the current directory's direct_ptrs to find either an empty block that needs to be initialized or a block with an available dirent. If the former, I allocate a new data block, populate it with a new dirent for the new file, call writei, and return. If the latter, I initialize the empty dirent with the file data, call writei and return.
    c. Dir_remove was simple. Once again I iterate through the direct_ptrs, this time looking for the block with the dirent that will be removed. Once the dirent is found, it is made invalid, and freed.
4. Get_node_by_path simply involved calling strtok_r with the file path and iterating through the resulting inodes by calling dir_find.
5. Initializing rufs was done as such:
    a. Rufs_mkfs calls dev_init to make the diskfile, initializes the superblock, initializes the bitmaps, and initializes the root inode. A global "opened" flag is set upon first call.
    b. Rufs_init checks if the "opened" flag is set. If it isn't, rufs_mkfs is called. If it is, the superblock and bitmaps are loaded.
    c. Rufs_destroy simply involved freeing the superblock and bitmaps.
6. Getattr first calls get_node_by_path to see if the inode at path exists, and if not, returns -ENOENT, if the inode is retrieved, then the stats are retrieved. First getuid() and getgid() are called for the st_uid and st_gid respectively. Then st_ino, st_nlink, and st_size are retrieved directly from the inode. St_blksize is set to

BLOCK_SIZE, and st_mode is determined based on the type field of the inode. St_atime and st_mtime are both filled in with a call to time(NULL).

7. The rufs directory functions went as so:
    a. Opendir simply involved calling get_node_by_path to retrieve the inode that corresponded with the path.
    b. Readdir first fills the buffer with the listings "." and ".." to correspond to the current and parent directory. Afterwards get_node_by_path is called to retrieve the relevant inode. After a check of the inode's type, the directory's contents are listed out. I iterate through the relevant data blocks so long as they are valid, and iterate through the dirents stored inside. Once all of them are read, the function returns.
    c. Mkdir first extracts the dirname and basename by calling dirname and basename respectively. Get_node_by_path is called on dirname to retrieve the directory's inode. Once done, the subdirectory's inode is initialized and the underlying contents are filled. Finally, dir_add is called to officially add the new subdirectory to the current directory.
    d. Rmdir works similar to this. First the method checks if the path is "/" because that means it is trying to remove root, and if it is, then the function returns -EPERM. Otherwise, the procedure follows similar to mkdir, with the dirname and basename retrieved and inodes being supplied. Once retrieved, the subdirectory is iterated through to clear out all dirents within the valid data blocks. Once cleared out, the bit for the directory on both bitmaps are unset. Dir_remove is called to remove the dirent for the subdirectory, and the function returns.
8. The rufs file functions were implemented in a similar way.
    a. Create works near identical to mkdir, except with the inode values being initialized for a file as opposed to being initialized for a dir.
    b. Open is identical to opendir. Get_node_by_path is called to find the inode for the file.
    c. Read is more involved than readdir. First the inode for the file is retrieved. After this, the file's direct_ptrs are iterated through to retrieve the data. They are transferred to the internal buffer. If more than BLOCK_SIZE bytes have to be read, the next direct_ptr is called. If all the direct data blocks have been read, then the focus is shifted towards the indirect_ptrs. If there is an indirect_ptr stored, then the data within is accessed, dereferenced, and iterated through. Once everything has been read and no more data needs to be accessed, the method returns the number of bytes read.
    d. Write is also similarly involved. The file's inode is retrieved from get_node_by_path, and then the ptrs are iterated through the same way as in read. Once the data is written into all the direct_ptrs and indirect_ptrs, the inode size is increased and the inode struct is amended.

e. Unlink is also similar to rmdir. First the exact inode to be deleted is found, then the direct_ptrs and indirect_ptrs are completely cleared, with their corresponding bits on the bitmaps also unset. Then dir_remove is called to remove the file's inode from the directory. Finally, the corresponding bit on the inode bitmap is cleared.

Problems

Unfortunately, the benchmarks failed, and I was unable to determine the exact errors that caused them. In both benchmarks, the everything works as intended until /files/dir0 is created. Here, the subdirectory itself was created, but afterwards, getattr is called, but then fails to find the subdirectory. I have an idea as to what might be causing it, as the direct_ptr in the subdirectory's inode is abnormally high, and I suspect the unlink function might be the source, but I was unable to verify this nor fix the error within time.

What I can say about the benchmarks however is that while they were working, both of them allocated up to 24 data blocks. The total runtime however, I was unable to determine.