CS416 Project 4 Report

Ryan Lee (rsl82)
Project was tested to be fully functional and fulfilling all requirements on iLab machine cp.cs.rutgers.edu.

1. Detailed logic of how you implemented each API function and the logic.

- get_avail_ino()

First, read the inode bitmap from disk. get available bitmap with get_bitmap and set_bitmap for that location. Return the value.

- get_avail_blkno()

Same as the get_avail_ino function. But read data bitmap instead of inode bitmap.

- readi(uint16_t ino, struct inode *inode)

Calculate block number and offset first for the inode. Read the block. Copy the target inode to the *inode with offset.

- writei(uint16_t ino, struct inode *inode)

Similar to readi, calculate block number and offset. Read the block where the target inode is stored. Change the target inode's information with offset. write back to the block.

- dir_find(uint16_t ino, const char *fname, size_t name_len, struct dirent *dirent)

Get inode with ino. Traverse the direct pointer of the inode and find if there is the same name for the target. If found, copy dirent of the target to the *dirent and return 0. If not, return -ENOENT.

- dir_add(struct inode dir_inode, uint16_t f_ino, const char *fname, size_t name_len)

First, if there is already fname in the dir_inode's direct pointers, return -EEXIST. Find the space for the new dirent and update the data block with the new dirent. Update the dir_inode with writei.

- get_node_by_path(const char *path, uint16_t ino, struct inode *inode)

Break the path with strtok((char*)path,"/"). If the first token is NULL, it means we return the root directory. Use strtok and find inode until we get to the end. If there is no inode for the path, return -ENOENT. If found, copy the target inode to *inode.

- rufs_mkfs()

Initialize the diskfile. Initialize superblock. Initialize bitmaps. Make the root directory inode and dirents for it. Write them all to the disk.

- rufs_init(struct fuse_conn_info *conn)

If DISKFILE is already made, read superblock and bitmaps from disk. Else, run rufs_mkfs()

- rufs_destroy(void *userdata)

Free superblock and bitmaps.

- rufs_getattr(const char *path, struct stat *stbuf)

Get target inode with get_node_by_path. Copy the value of the inode to the stbuf.

- rufs_opendir(const char *path, struct fuse_file_info *fi)

get path_inode. If I could not find or was not directory, return -1. Update the atime for the inode. return 0.

- rufs_readdir(const char *path, void *buffer, fuse_fill_dir_t filler, off_t offset, struct fuse_file_info *fi)

get path_inode. If path is not found, return -ENOENT. Fill all the dirents in the path_inode and copy them with filler function.

- rufs_mkdir(const char *path, mode_t mode)

Separate dir_name and base_name. I copied the original path and used them because dirname() and basename() modified the original path when using it. Find dir_inode. Make a new inode for the new directory and make the dirents for it. add the directory to the parent directory's direct pointer. writei new inode and return.

- rufs_create(const char *path, mode_t mode, struct fuse_file_info *fi)

Same as rufs_mkdir, but we do not have dirents for the new file inode.

- rufs_open(const char *path, struct fuse_file_info *fi)

Same as rufs_opendir, but this function is for the files.

- rufs_read(const char *path, char *buffer, size_t size, off_t offset, struct fuse_file_info *fi)

Find the target inode. Copy the data for the file from the offset to the buffer. If copying size is larger than the size of the file, copy until the end of the file.

- rufs_write(const char *path, const char *buffer, size_t size, off_t offset, struct fuse_file_info *fi)

Find the target inode. Write the data to the data block and link them to inode's direct pointer. Update the size and modification time for the inode and writei.

2. Benchmark results of your file library with time and number of used blocks.

```
rsl82@cp:~/Downloads/p42/p4/benchmark$ ./test_case
TEST 1: File create Success
TEST 2: File write Success
TEST 3: File close Success
TEST 4: File read Success
TEST 5: Directory create success
TEST 7: Sub-directory create success
Benchmark completed
Time: 1.227000 ms
```

In test cases, I passed all of the tests for the program. I also added a time counter for the benchmark code, I got 1.23 ms for the execution.

```
Blocks used: 132
```

I could not pass the number of used blocks to the user-end, so I checked it through debugging. I added to print used blocks in the opendir since the opendir function was executed last in the benchmark code. And it says I used 132 blocks to execute the benchmark code.

3. Any challenges for the project.

I was new to the FUSE file system. So, I was confused with the structure of this system and the meaning for the dirent. After I got clarification from Piazza, there was not much hard stuff for it. The hardest part was writing code for offset for read and write but it was solved.