Pedro Torres (pt397) and Ivan Zheng (iz60)

Project 3 Writeup

Code Implementation

`get_avail_ino`

We implemented it by first reading the inode bitmap from the disk. Once we have the bitmap, we iterated through from inode 0 to the max inode number and searched for the first bit that was set to 0. We returned this inode number

`get_avail_blkno`

We implemented it by first reading the block bitmap from the disk. Once we have the bitmap, we iterated through from block 0 to the max block number and searched for the first bit that was set to 0. We returned this block number + the start position of data blocks in the drive.

`readi`

We first found the block the inode belonged to (ino/ INO_FACTOR). The INO_FACTOR was derived by the sizeof(inode) / BLOCK_SIZE. We also found the inode offset in the block. We used bio_read to grab the inode from the disk, then stored the inode in the pointer given to us from the parameters of the function.

`writei`

Similar to readi, we first found the block and offset the inode belonged to using the INO_FACTOR. We called bio_read to get the block data, grabbed the inode pointer from the offset, copied the new inode to the pointer, then wrote back to the block.

`dir_find`

To implement dir_find, we first called readi to get the inode of the current directory. We then get the first data block that the inode points to. We then iterate through all the data blocks pointed to by the directory inode and for each data block, we iterate through all the dirents to check if the target file exists. We copy this dirent to the dirent given to us in the function parameters.

`dir_add`

To implement dir_add, we first call dir_find to see if the file we are trying to add already exists. We then check if the parent directory has space to accommodate a new dirent. If it does not, we add a new data block. We then iterate through the directory's data blocks and find a dirent that is not populated. We fill out this dirent for the new directory we are adding.

get_node_by_path

To implement get_node_by_path, we first iteratively loops through the path, calling dir_find at each level and returning once we finish, we save the inode from path to function parameter inode pointer and return 0, and a negative error otherwise. Each level is separated by a slash in the path string.

rufs_mkfs

In implementing rufs_mkfs, initiate the disk with the diskfile_path, and initiate all data structures we need for rufs. This includes the superblock, inode and data bitmaps, and the root inode for the root path "/". We write any relevant changes back into the disk.

rufs_init

In implementing rufs_init, we allocate the memory for the bitmaps and superblocks, and we read from memory to fetch data needed to populate the bitmaps and superblocks.

rufs_destroy

We free the malloced bitmaps and superblocks and call dev_close.

rufs_getattr

We call get_node_by_path to get the inode from the path, then we save the inode's vstat member into stbuf.

rufs_opendir

We call get_node_by_path to get the inode from the path, then return 0 if it exists, other returns -1.

rufs_readdir

In implementing rufs_readdir, we call get_node_by_path to get the directory inode, We then iterate through the inodes data blocks and therefore its dirents to call the filler function to populate with the directories file entries.

rufs_mkdir

To implement rufs_mkdir, we call get_node_by_path to get the parent directory from path. We then get the next available inode to create the new directory's inode. We then save this inode into the datablock of the parent directory using dir_add. We use writei to write the new inode back into the block.

rufs_create

To implement rufs_create, we call get_node_by_path to get the inode of the parent directory. We then get the next available inode to create the new file's inode. We then save this inode into the data block of the parent directory using dir_add. We use writei to write the new inode back into the block.

rufs_open

To implement rufs_open, we call get_node_by_path and if it exists we return 0, else -1.

rufs_read

To implement rufs_read, we call get_node_by_path to get the file's inode. We then calculate the start and end block from the offset and size requested. We then iterate through and use memcpy to copy from the inode's data blocks into the buffer we are given.

rufs_write

To implement rufs_write, we call get_node_by_path to get the file's inode. We then calculate the start and end block from the offset and size requested. If we need to add more data blocks to accommodate, we do. We then iterate through and use memcpy to copy from the buffer to the inode's data blocks.