Implementation

rufs_init() just calls rufs_mkfs() if it hasn't been initialized yet.

rufs_mkfs() initializes the superblock to be at the start. Then performs calculations determining where the start of the superblock, inode bitmap, data bitmap, inode blocks, and data blocks are. Then creates the superblock, and allocates one inode for the root directory. rufs_mkfs() also stores the locations of all the important information in memory, because these things are unchanging, and that would require less disk reads.

rufs_destroy() just closes the file using dev_close(). No dynamic memory needs to be free because no dynamic memory was allocated.

get_node_by_path() takes the path and writes inode data to the given pointer if it finds an inode at the requested filepath. Uses dirfind() to find the requested directory entry in the current directory.

rufs_getattr() calls get_node_by_path() to determine if the path requested exists. If it does, fill stbuf with the requested data.

rufs_mkdir() uses the path to create a directory. Calls dir_add() to find an open directory entry to store the inode number and name. Initializes all the indices of direct_ptr[16] to 0.

rufs_create() uses the path to create a file. Calls dir_add() to find an open directory entry to store the inode number and name. Initializes all the indices of direct_ptr[16] to 0.

rufs_read() reads the requested bytes from the file, at the offset requested. Returns count which tracks how many bytes have been read.

rufs_write() writes the requested bytes to the file. If the file's direct_ptrs aren't already assigned, assign the necessary amount until the requested offset. Go to the requested offset and start writing bytes. If offset + count exceeds the file size, that means we have written new bytes into the file. In this case, set the file size to offset + count. Then, return the amount of bytes written.

rufs_readdir()
Uses get_node_by_path to get the requested file inode, and then iterate through the

directories and fill them into the buffer using the filler() command. Start by filling the two required directory entries "." and "..".

The main difficulty I had was completing rufs_mkdir, towards the beginning of the problem. I was able to create multiple directories, but for some reason, any subdirectories I created were shared among all of the directories I created. The way I resolved this problem was by ensuring I wasn't giving each file duplicate data block pointers. Another issue I had was with get_node_by_path(). I had to add null terminators to the char array I passed to get_node_by_path, or the C string functions would not work properly.