Andrew Hoskins

811939260

CSCI 4730

This project is an implementation of an EXT2-like File System. Within this project I have implemented the file read, remove and link command along with the directory make, remove, and change command. To run my program enter in the command ./fs_sim disk.dat. From here you can issue any of the commands I made such as read, rm, ln, mkdir, rmdir, or cd.

The read command starts out with setting our variable of iNodeCounter to the results of the search_cur_dir with the given name. This allows us to find the specific inode by doing comparisons on what number our variable is. If our variable is negative that means the name is not found within the directory, but if it is positive that means the name is found. However, just because the name is found doesn't mean that it is a file so we use an if statement to check on the type of the name given. After this we need to check if the rest of the arguments are valid by using if statements to see if offset (starting position that can be 0) is more than the size of our file, if the user is entering in negative arguments, or the combination of offset and size go beyond the limits of the file. Once these checks are done, the code begins the read operation by setting a size to our new local variable with malloc and filling in the information using a while loop. Once the information is filled in it prints the results, cleans up trash, and calls gettimeofday to show that the user accessed the file.

The file remove command starts out the same way as our read command with setting up iNodeCounter. The checks on iNodeCounter are then performed to make sure it is a file and the name is available. After these checks are done, the for loops are used to adjust the directory entry array while incrementing the superblock's block and inode counter. Next up is the removal of one entry from the current directory. Lastly, we update the time of the last file access.

The link command begins with setting the variable up in a similar way to the previous commands. The code checks to see if the new file's name is taken, then checks if the source file is found. The next two if statements check if we can fit the file in the directory and if there are enough inodes to hard link. After these checks are done, we check begin the linking process by incrementing the link_count. The src and desc point to the same i-node with strncpy. Finally, it increments the numEntry of the current directory.

The mkdir command starts with a variable set to the get_free_block function and a variable of type Dentry. We do the same validation checks on the name given such as if the name is in use, the directory is full, and if there are not enough inodes. Once the checks are done, the code begins to initialize the values of the inode. The values of Dentry (the new directory) begin to be initialized following the previous initialization. We finally use stryncpy to get the directory added into the current directory.

The rmdir command uses an if statement in the beginning to make sure that the user cannot delete the "." or ".." directories. Then it checks to make sure the name is a directory and not anything else. Call the set_bit function along with incrementing the superblock's block and inode. The for loop copies the string of the current directory and its inode. Decrement the numEntry of the current directory since it has been removed and then call disk_write.

The cd command changes the directory by staring with same validation of the given name in the argument (checks to make sure it is a file). The curDirBlock is the / directory so we update it to the direct block of the inode given then call the disk_read function.

The description of my design is essentially just an average file system with many validations to cover any unwanted errors. My implementation correctly handles exception cases throughout the project. These handles are typically seen in the beginning of each of the commands I wrote. A good example showcasing this is primarily shown at the start of the read file command. It handles cases where the user would input negative numbers like -1 for the offset or size. Inode validation can be seen in the make command for the directory after iNodeCounter gets set to the get_free_inode function call. These are just a few examples of my code handling exception cases.