Christopher Brant
EEL 5737 HW2design
Due 9/16/19

HW2 Design and Testing Description

Design

write() – When designing and implementing the write function, I started off with a few nested conditional statements first for error checking, as was done for the read and copy functions as well. After the checks for inode type and offset argument validation, my function proceeds with the logical progression of two major sections. Those are a conditional statement that has instructions for when the function is writing to an existing block, and another for when it is writing to a newly allocated block. When writing at all I also kept a check of the current file size vs the maximum file size to know when to break out of the writing loop. When writing to an existing block, I needed to make sure I preserved the preceding data in the file, and then begin to write data to a list of values to update to a block once the list was able to fill a block. When writing to a newly allocated block, I did not have to check for old data, so that part did not exist in that section of code. Once the length of the data to write was exhausted, or the max file size was met, the loop would break and then the last block would be updated and the inode metadata would be updated and the inode returned.

read() – When designing and implementing the read function, it was much more straightforward, as all I needed to accomplish was checking for the same initial errors as in the write function, and then reading bytes from blocks until the length to read was exhausted or the end of the file was met. I implemented a loop that will read until the length to read is exhausted, but would break if the "cursor" at which I was reading got to the end of the file, and then the inode would be updated and returned along with the list of returned data. The loop for reading was simple, except the check to make sure I began to read from the next block caused me a slight issue as I was initially updating the block contents incorrectly at that time. Once that was fixed, the function worked flawlessly, I believe.

copy() – When designing and implementing the copy function, this was the simplest of the three, all it contains is a for loop for the length of the given inode's block number list, and makes a deep copy by allocating a new block and filling it with the given inode's data for each block it contains, then updating each block and updating the new inode's block number list. Before returning, I update the new inode's file size, and the existing inode's access time and then return the new copy of the inode.

Description of Testing

When testing the functionality of my implementation, I wrote a short script for testing that would create some inodes and do some basic changes and operations and then print out the information for each inode for observation and verification. I began with a few base cases and then tested the corner cases I expected as well. I tested the copy function with a few simple cases and did not know of any corner cases to test there. The write function was tested with a write in the middle of a block that doesn't fill the block, a write that does fill a block starting from the middle and beginning, a write that truncates when it hits the maximum file size and a write that begins in the middle of a block and then finishes in a new block. The read function was tested by reading from each of those writes that were tested to ensure it worked correctly.