COMSC 322
Operating Systems
Programming Assignment 4: The UNIX inode
Due: December 9, 9:00 PM

Note: Please do not put this off to the end!

The goal of this lab is to simulate the UNIX inode based on our discussion in class. We will save the contents of a file on the disk using this simulated inode.

For this assignment we will assume that the inode has 14 pointers of the index block. The first 12 pointers point to direct blocks, i.e., they contain addresses of blocks that contain data of the file. The last two pointers point to the **indirect blocks**. The 13th pointer points to a **single indirect block**, which is an index block containing not data but the addresses of blocks that do contain data. The 14th pointer points to a **double indirect block**, which contains the address of a block that contains the addresses of blocks that contain pointers to the actual data blocks.

Simplifying assumption: In this assignment we will simulate each block in the input file as a separate file on our disk. The block pointers, in that case, will be the file names storing the corresponding block data.

Your program should take two input files from the command line: input_file, file_access_trace. "Input_file" will contain information about the file that needs to be stored in the simulated inode. This file has two columns: "block number," and "data in block". As the names imply, "block number" indicates the number of this block in the original file to be stored (numbering starts from 0). "Data in block" contains the data stored in the corresponding block number. A sample input_file file "input_file1.txt" is released with the assignment. Use this file for testing your code.

"File_access_trace" is a sequence of block requests from the input_file. It has requests for reading or writing data into the file. E.g. entries may look like:

R, 100 //read block number 100 from the file
W, 21, Hello //Write/overwrite the contents of block number 21 with the string "Hello"

Let us consider the contents of test_file.txt for example:
#block number ,data in block
0, hi
1, hello
2, test
3, msg

A read request to block 2 should return the string “test,” similarly a write request “W, 2, test123” should replace the contents of block 2 with “test123.” A subsequent call “R, 2” should now return “test123.”

A sample file_access_trace (“access_trace1.txt”) accompanies this assignment. Use this file along with “input_file1.txt” for testing. For final evaluation, I will use a different set of test files.

In this assignment we will simulate each block in the file as a separate file on our disk, e.g., a new file in a subdirectory inside the current working directory. For example, you can create a subdirectory called “input_file1_dir” to store all the “blocks” corresponding to the “input_file1.txt.” Let us again consider the contents of test_file.txt (mentioned above). This file has four blocks. The inode of this file will contain: file name, i.e., “test_file.txt,” and four pointer values. In our simulation these pointers are the names of the files which store these blocks. For instance, if we store block 0 (containing “hi”) in “zero.txt,” block 1 (containing “hello”) in “one.txt,” block 2 (containing “test”) in “two.txt,” block 3 (containing “msg”) in “three.txt,” then the contents of inode would be: “test_file.txt,” “zero.txt,” “one.txt,” “two.txt,” “three.txt.” You should save the inode in a special file named “super_block.txt” in the same directory as other files. You can choose any format for saving the inode contents.

For simplicity, in this assignment we will assume that, in case of indirect block access, each block can store 100 file pointers (or names in this case). Therefore, the 13th pointer value in our inode will point to a file which can contain, at max, 100 file block pointers (remember in our case these pointers are the file names). Similarly, the 14th pointer value will point to a file which contains (at max) 100 file names, each of these files in turn will contain (at max) 100 file names each of which will contain actual data blocks.

While serving a read or write request for the file your program should print all the intermediate files it reads. For example, on servicing the “R, 12” request, it should print something like: “Accessed pointer 12 of inode; next, read the 0th entry of the file ‘level1_indirection.txt’ .” (Here we are assuming the counting starts from 0). Similarly, when servicing double indirection blocks, you should print all the intermediate files read and the positions/pointers that were accessed.

Your submission will be graded for the following:

1. Correct functioning of Read operations:

A. Read operation in direct blocks. (5)
B. Read operation in **single indirect block.** (10)
C. Read operation in **double indirect block.** (10)

2. Correct functioning of Write operations:

A. Write operation in direct blocks. (5)
B. Write operation in **single indirect block.** (10)
C. Write operation in **double indirect block.** (10)

Please note that part of the correctness of your Read and Write operations will be determined by the intermediate print messages, as mentioned above. So please make sure these messages are accurate and readable.

3. If the "input_file" specifies a file greater than the maximum file size, your program should throw an error message: "input file greater than max supported file size!" (5)

4. If the trace file tries to access an invalid block number, your program should throw an error message: "Invalid block number!" (5)

Answer the following in your report:

5. What is the maximum number of blocks (in a file) that the above inode can support? (10)

6. Explain, succinctly, how does your program locate the address of a given block number in the inode? Answer this for the following three cases: (10)

A. Block number is from 0 to 11.
B. Block number is from 12 to 111.
C. Block number is >= 112.

7. What naming convention do you use for naming the files containing the input file blocks? (You should explain how do you name the files pointed by the direct block pointers in the inode? Similarly, how do you name the files pointed by the **single indirect block** and **double indirect block**?) (5)

8. Prepare a "README.txt" file for your submission. The file should contain the following: (10)

a) Names of all the group members.
b) Instructions for compiling and executing your program(s). Include an example command line.
c) If your implementation does not work, you should also document the problems in the README file, preferably with your explanation of why it does not work and how you would solve it if you had more time.
d) If you did not implement certain features, you should list them as well.

9. You should also comment your code well. The best way to go about it is to write comments while coding. (5)