EEL-4736/EEL-5737 Principles of Computer System Design

Homework #3

Assigned: 9/16/2019; Due on 10/2/2019 – To be done individually

Part A

a) Problem set 1 (Bigger files)
b) Problem set 4 (EZPark)
c) Problem set 9 (Ben's Kernel)

Part B

In previous assignments, you implemented the raw block layer and the inode layer that allowed you to split the contents of a file across blocks of fixed size. However, that design did not provide the ability to name and organize files in directories. As discussed in class, practical UNIX-like file systems support a hierarchical namespace to organize the files into directories.

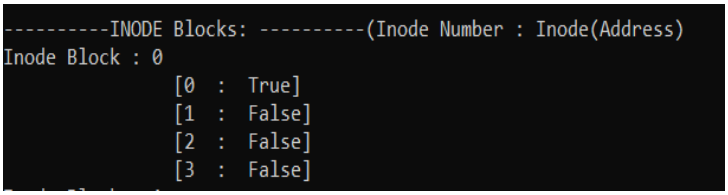
In this assignment, you will complete the implementation of the in-memory file system. Specifically, your goal is to implement the following methods of the file name and inode number layers discussed in class: unlink(), link(), read(), write() in **InodeNumberLayer.py** and unlink(), link(), read(), write(), mv() in **FileNameLayer.py**.

Skeleton code for the **FileNameLayer.py** and **InodeNumberLayer.py** files have been provided to you with this assignment. In addition, we are providing you with the full code that implements the two layers that sit above it – **AbsolutePathNameLayer.py** and **FileSystem.py**.

InodeNumberLayer.py

This layer maintains the inode table, which is a mapping of inode number and the respective Inode structure in memory.

new_inode_number – In previous assignments, we were just making an Inode structure and performing operations on them. Now, this function makes a new_inode_number with its respective Inode, and saves the inode in memory using **update_inode_table.** You can check it by running the "status" function of FileSystem. When your file system is initialized, the inode table's first entry is set to True.



In assignment 2, you used **INODE_TO_BLOCK** to fetch the data. In this assignment, you will use **INODE_NUMBER_TO_BLOCK** to fetch data, as the file system layer builds up layer by layer.

You have to implement these methods in **InodeNumberLayer.py** for this assignment:

- **unlink(inode_number, parent_inode_number, filename)** – removes a link in the file system; if it is the last link to be removed for the inode_number, free all blocks associated with the

inode (if it is a file inode), and free the inode. Note: an inode for a non-empty directory cannot be removed.

- **link(file_inode_number, hardlink_name, hardlink_parent_inode_number)** – creates a hard link with name "new_path" to the object resolved by "old_path
- **read(self, inode_number, offset, length, parent_inode_number)** – reads "length" bytes from a file, starting at "offset
- **write(self, inode_number, offset, data, parent_inode_number)** – writes "data" to a file, starting at "offset".

In addition to **implementing** the link, unlink, read, write methods, please **explain** how you would go about implementing a symlink method (you don't have to implement it – just describe in your own words)

FileNameLayer.py

This layer sits atop the inode number layer, and supports the use of user-friendly file names. The following are important methods that are provided to you.

LOOKUP – This is a core function used by the other functions. It recursively looks up file system object with name "path" in the context of the given directory. This function takes path and inode number (note: **inode_number_cwd** refers to the inode number of the directory used as the starting context to resolve path names) and returns the parent node number (or -1, if the file does not exist). You must use this function to retrieve the parent inode number of any file or directory which already exists.

CHILD_INODE_NUMBER_FROM_PARENT_INODE_NUMBER – This function gives you the child inode number from its parent inode number and childname. You have to use this function to get child inode number before making the call to InodeNumber Layer.

new_entry() – This function makes a new file/directory using the filename.

You have to implement below methods on this layer, which parse the whole path and passes inode_number of the file/directory to InodeNumberLayer:

- **unlink(path, inode_number_cwd)**
- **link(self, old_path, new_path, inode_number_cwd)**
- **read(self, self, path, inode_number_cwd, offset, length)**
- **write(self, path, inode_number_cwd, offset, data)**
- **mv(self, old_path, new_path, inode_number_cwd)**

Code that is provided to you - for your reference:

InodeOps.py

In a previous assignment your used Table_Inode structure of the inode. This time, you will also be using this structure to operate on inode in the file system, but when you are storing the inode in the memory, this structure is first converted into Array_Inode and, while retrieving, it is

converted back to table. Therefore, to update the inode table with inode we use a helper function **convert_table_to_array** to convert Table_Inode to array. Similarly, when we are fetching the Inode we are converting it in the tabular form for ease of operations with the help of **convert_array_to_table.**

AbsolutePathNameLayer.py

This module implements the absolute path name layer of the file system. You are suggested to not make any changes in this module. Please read the code carefully to see which function of FileNameLayer is called for which functionality to get an idea of layering.

FileSytem.py

You are provided with FileSystem.py module which implements the basic operations of file system. Use these functions to perform the operations on your file system. Please follow the examples and make sure that you provide the absolute path for e.g. "/A/B…". File system is initialized by "Initialize_My_FileSystem()" when you run "FileSystem.py" with root directory (inode number = 0). You can check the "status" function when you run the "FileSystem.py". Your "status" function will also show a glimpse of hierarchy that Unix-type file system implements. Utilize this function to test your code and read the comments in the code to figure out their respective functionality.

Basic operations are -
mkdir – creates the directory
create – creates the file.
write – writes into file
read – reads the file
rm – removes the file(removes file only)
mv – move from one folder to another

Example –

When you will run create("/A/B/1.txt") in FileSystem.py, assuming A and B directory exists, this simply passes this path with root inode number to AbsolutePathNameLayer.py where the layer checks and removes '/' in the path and then it is passed to FileNameLayer.py which parses the path and performs lookup to till the parent. Then it asks to create new InodeNumber for 1.txt in InodeLayer.py. Next, It will update the Inode table so that when next time you perform lookup to "write" this file, you will have inode number in the memory to update it.

Note -

- **There are some changes in Memory.py, config.py, InodeOps.py. Please use the latest files that are attached with the zip archive posted with the assignment.**
- **You have to copy your read() and write() functionalities of HW2 (InodeLayer.py) in the new InodeLayer.py of this assignment.**

Bottom line – what code you must implement:

InodeNumberLayer.py: unlink(), link(), read(), write()

FileNameLayer.py: unlink(), link(), read(), write(), mv()

Submission guidelines:

Turn in through Canvas following four files (attach individually):

1. homework3partA.pdf – Solution to the questions in Part A
2. InodeNumberLayer.py and FileNameLayer.py – Your Python code.
3. homework3design.pdf – PDF describing the design of your implementation and tests you have conducted to check the functionality of your code. Also included in this file should be your discussion of how you would approach the design of a symlink method (again, you don't need to implement it).

Make sure your Python code is well commented and tested before submission. Assignment will be graded on the basis of design and functionality. There might be several ways to implement this assignment, choose one which is most feasible, concise and modular allowing you to have minimum changes in the existing code when you extend its functionality. Copy and the paste the python code of all the layers (InodeNumberLayer, FileNameLayer) at the end of your homework3design.pdf file.