Homework 3 Design and Testing

**Design and Implementation**

The design of my code was based in modularity and what would be the best way to go about making the layers do what they each should be doing and not handling something a lower layer should be handling. That being said, most of my code in FileNameLayer.py, as you can see pasted in the following pages, just collects the inode number information needed so that it can pass those values down to the methods in InodeNumberLayer.py which will take care of the rest of the work that is required. However, I have implemented an extra caveat in the link method in FileNameLayer.py that allows for files to be renamed if the path given ends with a filename and not a directory name (or just a name that isn’t a directory), it will be moved and renamed to that file name. For link in the InodeNumberLayer, I implemented link in the most straightforward way by grabbing the inodes via their inode numbers, checking if they are valid, them updating them as expected for the new link. For unlink in the InodeNumberLayer, the method was slightly more complex, as it has to check if the inode is a directory inode or a file, and if it is a directory it cannot free the blocks and the inode unless the directory is empty. So that required a bit more code. Then lastly the read and write methods in InodeNumberLayer simply grab the inode objects and pass them to the InodeLayer read and write methods with some added error checks. Overall very modular and this was done intentionally.

**Symlink Method Design (Not Implemented)**

The way that I would have gone about implementing a symlink method would be to allocate a new inode for that symbolic link, instantiate the inode with a reference count of 1, an inode type of 2 for symbolic link, and I would save the name of the path to link to be the only entry in the inode.directory dictionary with the (key, value) binding being (path, “symlink”). Then once this is all saved in its own inode, the parent inode would be updated to contain this inode’s (name, inode) binding and the inode table is updated to reflect these new changes. This allows for the path string to be saved and easily accessed for any other methods that need to symlink.

**Testing Process**

For my testing process I was rather straightforward, as after re-testing my code from HW2 that was fixed and copied into the InodeLayer.py using the HW2 testbench examples, I utilized the example code options in the FileSystem.py file that was provided to us so that I could do an overall litmus test on the system. Once I had tested all of the given example commands, I changed the commands so that I could test the functionality of the link, unlink, and move functions in the FileNameLayer by testing all the different ways of creating, moving, and removing directories and files. Specifically, I tested creating as many directories/files in the root as possible, then creating multiple subdirectories and creating files within subdirectories, then moving subdirectories around (does not rename them, however I do have the capability of renaming files using the move command) and files around, then removing directories/files as well. For the read and write functions in FileNameLayer, there was little testing I did specifically for this layer, since all they do is call the lower layer functions after gathering the necessary inode numbers to pass down to the lower layer. Down at the lower layer, the InodeNameLayer, I tested the link and unlink methods by scrutinizing and printing out critical values throughout all the main branching steps of those functions when I would create or move directories/files in the FileSystem.py script. Similarly, the read and write methods in the InodeNameLayer were tested the same way by printing out the critical values at every branching point, although these methods mainly call the InodeLayer read and write after checking if the given inode number is valid and within the given parent inode number’s directory.

# BELOW IS THE CODE FOR THE INODENUMBERLAYER.PY AND FILENAMELAYER.PY FILES #

'''

THIS MODULE ACTS AS A INODE NUMBER LAYER. NOT ONLY IT SHARES DATA WITH INODE LAYER, BUT ALSO IT CONNECTS WITH MEMORY INTERFACE FOR INODE TABLE

UPDATES. THE INODE TABLE AND INODE NUMBER IS UPDATED IN THE FILE SYSTEM USING THIS LAYER

'''

import InodeLayer, config, MemoryInterface, datetime, InodeOps, MemoryInterface

#HANDLE OF INODE LAYER

interface = InodeLayer.InodeLayer()

class InodeNumberLayer():

#PLEASE DO NOT MODIFY

#ASKS FOR INODE FROM INODE NUMBER FROM MemoryInterface.(BLOCK LAYER HAS NOTHING TO DO WITH INODES SO SEPERTAE HANDLE)

def INODE\_NUMBER\_TO\_INODE(self, inode\_number):

array\_inode = MemoryInterface.inode\_number\_to\_inode(inode\_number)

inode = InodeOps.InodeOperations().convert\_array\_to\_table(array\_inode)

if inode: inode.time\_accessed = datetime.datetime.now() #TIME OF ACCESS

return inode

#PLEASE DO NOT MODIFY

#RETURNS DATA BLOCK FROM INODE NUMBER

def INODE\_NUMBER\_TO\_BLOCK(self, inode\_number, offset, length):

inode = self.INODE\_NUMBER\_TO\_INODE(inode\_number)

if not inode:

print("Error InodeNumberLayer: Wrong Inode Number! \n")

return -1

return interface.read(inode, offset, length)

#PLEASE DO NOT MODIFY

#UPDATES THE INODE TO THE INODE TABLE

def update\_inode\_table(self, table\_inode, inode\_number):

if table\_inode: table\_inode.time\_modified = datetime.datetime.now() #TIME OF MODIFICATION

array\_inode = InodeOps.InodeOperations().convert\_table\_to\_array(table\_inode)

MemoryInterface.update\_inode\_table(array\_inode, inode\_number)

#PLEASE DO NOT MODIFY

#FINDS NEW INODE INODE NUMBER FROM FILESYSTEM

def new\_inode\_number(self, type, parent\_inode\_number, name):

if parent\_inode\_number != -1:

parent\_inode = self.INODE\_NUMBER\_TO\_INODE(parent\_inode\_number)

if not parent\_inode:

print("Error InodeNumberLayer: Incorrect Parent Inode")

return -1

entry\_size = config.MAX\_FILE\_NAME\_SIZE + len(str(config.MAX\_NUM\_INODES))

max\_entries = (config.INODE\_SIZE - 79 ) / entry\_size

if len(parent\_inode.directory) == max\_entries:

print("Error InodeNumberLayer: Maximum inodes allowed per directory reached!")

return -1

for i in range(0, config.MAX\_NUM\_INODES):

if self.INODE\_NUMBER\_TO\_INODE(i) == False: #FALSE INDICTES UNOCCUPIED INODE ENTRY HENCE, FREEUMBER

inode = interface.new\_inode(type)

inode.name = name

self.update\_inode\_table(inode, i)

return i

print("Error InodeNumberLayer: All inode Numbers are occupied!\n")

#LINKS THE INODE

def link(self, file\_inode\_number, hardlink\_name, hardlink\_parent\_inode\_number):

# Lookup location to add hardlink\_name to

file\_inode = self.INODE\_NUMBER\_TO\_INODE(file\_inode\_number)

hardlink\_parent\_inode = self.INODE\_NUMBER\_TO\_INODE(hardlink\_parent\_inode\_number)

# Ensure the inodes are valid before using them

if (hardlink\_parent\_inode) == False or (file\_inode == False):

print "\nError: Parent inode or file inode number supplied is invalid.\n"

return -1

# Add link to directory in new location

hardlink\_parent\_inode.directory[hardlink\_name] = file\_inode\_number

# Increment file\_inode ref count

file\_inode.links += 1

# Update the inode table with the new values necessary

self.update\_inode\_table(hardlink\_parent\_inode, hardlink\_parent\_inode\_number)

self.update\_inode\_table(file\_inode, file\_inode\_number)

# Return if we get here

return

#REMOVES THE INODE ENTRY FROM INODE TABLE

def unlink(self, inode\_number, parent\_inode\_number, filename):

# Retrieve inode to unlink filename from

inode = self.INODE\_NUMBER\_TO\_INODE(inode\_number)

parent\_inode = self.INODE\_NUMBER\_TO\_INODE(parent\_inode\_number)

# Ensure the inodes are valid before using them

if (parent\_inode) == False or (inode == False):

print "\nError: Parent inode or file inode number supplied is invalid.\n"

return -1

# Check if we need to free the inode and do so if necessary

if inode.type == 1: # If inode is a directory

if (inode.links - 1) == 1:

# Check if directory is empty

if len(inode.directory) == 0:

# Remove the filename from the parent\_inode

del parent\_inode.directory[filename]

# if empty, free all blocks in inode, and free the inode

interface.free\_data\_block(inode, 0)

inode = False

else: # If not free return error with message of non-empty directory unlink attempt

print "\nError: Attempt to remove a the last link to a non-empty directory."

return -1

else:

# Remove the filename from the parent\_inode

del parent\_inode.directory[filename]

# Decrement reference count for inode

inode.links -= 1

elif inode.type == 0: # If inode is a file

if (inode.links - 1) == 0:

# Remove the filename from the parent\_inode

del parent\_inode.directory[filename]

# Free all blocks and free the inode

interface.free\_data\_block(inode, 0)

inode = False

else:

# Remove the filename from the parent\_inode

del parent\_inode.directory[filename]

# Decrement reference count for inode

inode.links -= 1

else: # If inode is not a file or directory (for now) return error

print "\nGiven inode is of a type: ", inode.type, "and is not acceptable in this system."

return -1

# Update parent inode and the file inode in the inode table

self.update\_inode\_table(parent\_inode, parent\_inode\_number)

self.update\_inode\_table(inode, inode\_number)

# Return if it gets this far

return True

#IMPLEMENTS WRITE FUNCTIONALITY

def write(self, inode\_number, offset, data, parent\_inode\_number):

# Retrieve parent inode and file inode

parent\_inode = self.INODE\_NUMBER\_TO\_INODE(parent\_inode\_number)

inode = self.INODE\_NUMBER\_TO\_INODE(inode\_number)

# Ensure the inodes are valid before using them

if (parent\_inode) == False or (inode == False):

print "\nError: Parent inode or file inode number supplied is invalid.\n"

return -1

# Check if the inode exists in the parent directory

if inode\_number in parent\_inode.directory.values():

# Check inode type

if inode.type != 0:

print "\nError: inode is not of type: file.\n"

return -1

else:

# Call the InodeLayer write function

inode = interface.write(inode, offset, data)

# Check for errors

if inode == -1:

print "\nError in InodeLayer write to inode\_number: ", inode\_number, "\n"

return -1

else:

# Update the inode table

self.update\_inode\_table(inode, inode\_number)

# Return

return True

else:

# if the inode does not exist in the parent directory return error

print "\nError: Given inode number does not have a binding in the given parent inode number's context.\n"

return -1

#IMPLEMENTS READ FUNCTIONALITY

def read(self, inode\_number, offset, length, parent\_inode\_number):

# Retrieve parent inode and file inode

parent\_inode = self.INODE\_NUMBER\_TO\_INODE(parent\_inode\_number)

inode = self.INODE\_NUMBER\_TO\_INODE(inode\_number)

# Ensure the inodes are valid before using them

if (parent\_inode) == False or (inode == False):

print "\nError: Parent inode or file inode number supplied is invalid.\n"

return -1

# Check if the inode exists in the parent directory

if inode\_number in parent\_inode.directory.values():

# Check inode type

if inode.type != 0:

print "\nError: inode is not of type: file.\n"

return -1

else:

# Call the InodeLayer read function

inode, retData = interface.read(inode, offset, length)

if inode == -1:

print "\nError in InodeLayer read from inode\_number: ", inode\_number, "\n"

return -1

else:

# Update the inode table

self.update\_inode\_table(inode, inode\_number)

# Return the data read

return retData

else:

# if the

print "\nError: Given inode number does not have a binding in the given parent inode number's context.\n"

return -1

'''

THIS MODULE ACTS LIKE FILE NAME LAYER AND PATH NAME LAYER (BOTH) ABOVE INODE LAYER.

IT RECIEVES INPUT AS PATH (WITHOUT INITIAL '/'). THE LAYER IMPLEMENTS LOOKUP TO FIND INODE NUMBER OF THE REQUIRED DIRECTORY.

PARENTS INODE NUMBER IS FIRST EXTRACTED BY LOOKUP AND THEN CHILD INODE NUMBER BY RESPECTED FUNCTION AND BOTH OF THEM ARE UPDATED

'''

import InodeNumberLayer, os

#HANDLE OF INODE NUMBER LAYER

interface = InodeNumberLayer.InodeNumberLayer()

class FileNameLayer():

#PLEASE DO NOT MODIFY

#RETURNS THE CHILD INODE NUMBER FROM THE PARENTS INODE NUMBER

def CHILD\_INODE\_NUMBER\_FROM\_PARENT\_INODE\_NUMBER(self, childname, inode\_number\_of\_parent):

inode = interface.INODE\_NUMBER\_TO\_INODE(inode\_number\_of\_parent)

if not inode:

print("Error FileNameLayer: Lookup Failure!")

return -1

if inode.type == 0:

print("Error FileNameLayer: Invalid Directory!")

return -1

if childname in inode.directory: return inode.directory[childname]

print("Error FileNameLayer: Lookup Failure!")

return -1

#PLEASE DO NOT MODIFY

#RETUNS THE PARENT INODE NUMBER FROM THE PATH GIVEN FOR A FILE/DIRECTORY

def LOOKUP(self, path, inode\_number\_cwd):

name\_array = path.split('/')

if len(name\_array) == 1: return inode\_number\_cwd

else:

child\_inode\_number = self.CHILD\_INODE\_NUMBER\_FROM\_PARENT\_INODE\_NUMBER(name\_array[0], inode\_number\_cwd)

if child\_inode\_number == -1: return -1

return self.LOOKUP("/".join(name\_array[1:]), child\_inode\_number)

#PLEASE DO NOT MODIFY

#MAKES NEW ENTRY OF INODE

def new\_entry(self, path, inode\_number\_cwd, type):

if path == '/': #SPECIAL CASE OF INITIALIZING FILE SYSTEM

interface.new\_inode\_number(type, inode\_number\_cwd, "root")

return True

parent\_inode\_number = self.LOOKUP(path, inode\_number\_cwd)

parent\_inode = interface.INODE\_NUMBER\_TO\_INODE(parent\_inode\_number)

childname = path.split('/')[-1]

if not parent\_inode: return -1

if childname in parent\_inode.directory:

print("Error FileNameLayer: File already exists!")

return -1

child\_inode\_number = interface.new\_inode\_number(type, parent\_inode\_number, childname) #make new child

if child\_inode\_number != -1:

parent\_inode.directory[childname] = child\_inode\_number

interface.update\_inode\_table(parent\_inode, parent\_inode\_number)

#IMPLEMENTS READ

def read(self, path, inode\_number\_cwd, offset, length):

# Split filename and filepath

filepath, filename = os.path.split(path)

# Find the file's parent's inode number

parent\_inode\_number = self.LOOKUP(path, inode\_number\_cwd)

# Find the file's inode number

file\_inode\_number = self.CHILD\_INODE\_NUMBER\_FROM\_PARENT\_INODE\_NUMBER(filename, parent\_inode\_number)

# If parent\_inode\_number or file\_inode\_number are bad, return error

if (parent\_inode\_number == -1) or (file\_inode\_number == False):

print "\nError: FileNameLayer LOOKUP failed to find file to read from.\n"

return -1

else:

# Call the InodeNumberLayer read function

retData = interface.read(file\_inode\_number, offset, length, parent\_inode\_number)

if retData == -1:

print "\nError: issue in reading data from file at inode\_number ", file\_inode\_number, "\n"

return -1

else:

return retData

#IMPLEMENTS WRITE

def write(self, path, inode\_number\_cwd, offset, data):

# Split filename and filepath

filepath, filename = os.path.split(path)

# Find the file's parent's inode number

parent\_inode\_number = self.LOOKUP(path, inode\_number\_cwd)

# Find the file's inode number

file\_inode\_number = self.CHILD\_INODE\_NUMBER\_FROM\_PARENT\_INODE\_NUMBER(filename, parent\_inode\_number)

# If parent\_inode\_number or file\_inode\_number are bad, return error

if (parent\_inode\_number == -1) or (file\_inode\_number == False):

print "\nError: FileNameLayer LOOKUP failed to find file to write to.\n"

return -1

else:

# Call the InodeNumberLayer read function

retErr = interface.write(file\_inode\_number, offset, data, parent\_inode\_number)

if retErr == -1:

print "\nError: issue in writing data to file at inode\_number ", file\_inode\_number, "\n"

return -1

else:

return True

#HARDLINK

def link(self, old\_path, new\_path, inode\_number\_cwd):

# Split paths and names for usage

new\_link\_path, new\_link\_name = os.path.split(new\_path)

child\_path, child\_name = os.path.split(old\_path)

# Find the parent to the child inode number

parent\_inode\_number = self.LOOKUP(old\_path, inode\_number\_cwd)

new\_link\_grandparent\_inode\_number = self.LOOKUP(new\_path, inode\_number\_cwd)

if (parent\_inode\_number == -1) or (new\_link\_grandparent\_inode\_number == -1):

print "\nError: One or more parent inode numbers are invalid for linking in FileNameLayer.\n"

return -1

# Get inode number at next location

new\_link\_parent\_inode\_number = self.CHILD\_INODE\_NUMBER\_FROM\_PARENT\_INODE\_NUMBER(new\_link\_name, new\_link\_grandparent\_inode\_number)

# Find the child inode number

child\_inode\_number = self.CHILD\_INODE\_NUMBER\_FROM\_PARENT\_INODE\_NUMBER(child\_name, parent\_inode\_number)

child\_inode = interface.INODE\_NUMBER\_TO\_INODE(child\_inode\_number)

if (child\_inode\_number == False) or (new\_link\_parent\_inode\_number == False):

print "\nError: issue in finding child inode number to create new hard link for.\n"

return -1

else:

# Call InodeNumberLayer link with parent inode number for new path

new\_link\_parent\_inode = interface.INODE\_NUMBER\_TO\_INODE(new\_link\_parent\_inode\_number)

child\_inode = interface.INODE\_NUMBER\_TO\_INODE(child\_inode\_number)

# This is here for the ambiguity in how somebody may give arguments to the move function

if new\_link\_parent\_inode == False:

new\_link\_parent\_inode\_number = new\_link\_grandparent\_inode\_number

if child\_inode.type != 1 and new\_link\_name != "":

child\_name = new\_link\_name

else:

print "\nWarning: Attempted to rename a directory upon moving it."

print "Our system does not allow this as we save directory names in the inode."

print "And this becomes a complex issue of how to know when to rename the inode within"

print "the framework we are working in.\n"

if child\_name == "":

print "\nError: Invalid name to link to. [blank name]"

return -1

linkErr = interface.link(child\_inode\_number, child\_name, new\_link\_parent\_inode\_number)

if linkErr == -1:

print "\nError: issue in linking " + new\_path + " to " + old\_path + "\n"

return -1

else:

return True

#REMOVES THE FILE/DIRECTORY

def unlink(self, path, inode\_number\_cwd):

if path == "":

print("Error FileNameLayer: Cannot delete root directory!")

return -1

# Find the childpath and childname by splitting the path string

# MAKE SURE IMPORTING THE OS LIBRARY IS OKAY

childpath, childname = os.path.split(path)

# Get the parent\_inode\_number

parent\_inode\_number = self.LOOKUP(path, inode\_number\_cwd)

if parent\_inode\_number == -1:

print "\nError: FileNameLayer LOOKUP failed to find file to unlink.\n"

return -1

# Get child inode number from the inode\_number\_cwd

child\_inode\_number = self.CHILD\_INODE\_NUMBER\_FROM\_PARENT\_INODE\_NUMBER(childname, parent\_inode\_number)

# If file\_inode\_number is bad, return error

if child\_inode\_number == False:

print "\nError: FileNameLayer unlink failed to find the child inode number.\n"

return -1

else:

# Call the InodeNumberLayer unlink

unlinkErr = interface.unlink(child\_inode\_number, parent\_inode\_number, childname)

# Return something?

if unlinkErr == -1:

print "\nError: issue in unlinking " + path + "\n"

return -1

else:

return True

#MOVE

def mv(self, old\_path, new\_path, inode\_number\_cwd):

# Link to the new path

linkErr = self.link(old\_path, new\_path, inode\_number\_cwd)

if linkErr == -1:

print "\nError: issue in moving file from " + old\_path + " to " + new\_path + ".\n"

return -1

else:

# Delete link at the old path

unlinkErr = self.unlink(old\_path, inode\_number\_cwd)

if unlinkErr == -1:

print "\nError: issue in moving file from " + old\_path + " to " + new\_path + ".\n"

return -1

else:

return True