Christopher Brant
EEL 5737
Due 10/18/19

Homework 4 Design and Tests

Design and Implementation

First of all, I would like to say that I added something to my InodeNumberLayer.py such that it would not link a file to a new directory such that the directory would end up containing two duplicate names. The code snippet that was added is pasted following the description page, and above the implementation of client_stub.py and server_stub.py. Next, my design and implementation was relatively straightforward and I also was able to optimize the way that I wrote my code such that both the client_stub.py and server_stub.py files are just about as lightweight as they can be. I did this by overloading the return instruction line such that it would it would un-marshal inputs, call its necessary function, and marshal the output or return data from functions before returning. Therefore, it will show that my client_stub.py and server_stub.py implementations seem rather lightweight, and that was on purpose. The only real difference between the two stubs is that my client_stub.py file has all of its server calls within try-catch statements so that it will catch errors/faults and report them accordingly without causing a total program fault.

Tests and Comparison to Local Filesystem

Once I had fully written and implemented my code in the client_stub.py and server_stub.py files, I tested my code the same way that I tested my code for HW3 running the very same series of operations from the Filesystem.py python script and comparing the differences or issues between the two. Other than a few issues dealing with my need to add the inability for a link to create duplicate file names within a directory, there were no issues that I was able to find in my testing, except I had accidentally tabbed too far on a conditional else statement in my last implementation of FileNameLayer.py, so I fixed that as well, and hopefully that did not cause any major issues in my HW3 grading. Once I had verified functionality was working correctly, or at the very least, appeared to be working as expected, I added timing checks in both my local Filesystem.py implementation and the server/client implementation for this assignment. I tested each version 10 times, and of those 10 for each, 5 were including the filesystem initialization in the timing statistics, and 5 excluded that timing. I took the average of the 5 runs for each version and test type, and I found that Local with initialization included had an average timing of 2.03690434 seconds and without initialization had an average timing of 0.02059855 seconds. The server/client version with initialization had an average timing of 2.68415875 seconds, and without initialization had an average timing of 0.66032896 seconds. This shows that the initialization generally takes about the same amount of time in both cases, but that the server/client version takes about 33x longer to fully finish than the local implementation version. When there is a failure in the network, my implementation handles this using try-catch statements in the client stub, and if an error in the network comes about, then the catch statement will catch it and print out the error messages to the terminal.

Christopher Brant
EEL 5737
Due 10/18/19

```
# This is what was added to link() in InodeNumberLayer.py
# Add link to directory in new location
                    if not hardlink_name in hardlink_parent_inode.directory.keys():
                              hardlink_parent_inode.directory[hardlink_name] = file_inode_number
                              # Increment file_inode ref count
                              file_inode.links += 1
                    else:
                              print "\nError: Attempt to link two files with the same name in a single
directory."
                              return -1

        # BELOW HERE IS THE CODE FOR client_stub.py AND server_stub.py RESPECTIVELY #
# client_stub.py
# Christopher Brant
# University of Florida
# Fall 2019
# EEL 5737 PoCSD
# HW4 Part B
import xmlrpclib, config, pickle, time

class client_stub():

          def __init__(self):
                    self.proxy = xmlrpclib.ServerProxy("http://localhost:8000/")

          # CLIENT REQUEST TO INITIALIZE THE MEMORY SYSTEM
          def Initialize(self):
                    try:
                              self.proxy.Initialize()
                    except Exception as err:
                              # print error message
                                        print "Error in re-initializing the filesystem."
                                        quit()


          #REQUEST TO FETCH THE INODE FROM INODE NUMBER FROM SERVER
          def inode_number_to_inode(self, inode_number):
                    # Return the correct data
                    try:
                              return
pickle.loads(self.proxy.inode_number_to_inode(pickle.dumps(inode_number)))
                    except xmlrpclib.Error as err:
                              print "A fault occurred in client_stub.inode_number_to_inode()"
```

```
                        print "Fault code: %d" % err.faultCode
                        print "Fault string: %s" % err.faultString
                        quit()


        #REQUEST THE DATA FROM THE SERVER
        def get_data_block(self, block_number):
                # Return the correct data
                try:
                        return
pickle.loads(self.proxy.get_data_block(pickle.dumps(block_number)))
                except xmlrpclib.Error as err:
                        print "A fault occurred in client_stub.get_data_block()"
                        print "Fault code: %d" % err.faultCode
                        print "Fault string: %s" % err.faultString
                        quit()


        #REQUESTS THE VALID BLOCK NUMBER FROM THE SERVER
        def get_valid_data_block(self):
                # Return the correct data
                try:
                        return pickle.loads(self.proxy.get_valid_data_block())
                except xmlrpclib.Error as err:
                        print "A fault occurred in client_stub.get_valid_data_block()"
                        print "Fault code: %d" % err.faultCode
                        print "Fault string: %s" % err.faultString
                        quit()


        #REQUEST TO MAKE BLOCKS RESUABLE AGAIN FROM SERVER
        def free_data_block(self, block_number):
          # Return the possible error, if no error, set retErr to 0
                try:
                        return
pickle.loads(self.proxy.free_data_block(pickle.dumps(block_number)))
                except xmlrpclib.Error as err:
                        print "A fault occurred in client_stub.free_data_block()"
                        print "Fault code: %d" % err.faultCode
                        print "Fault string: %s" % err.faultString
                        quit()
```

```
	#REQUEST TO WRITE DATA ON THE THE SERVER
	def update_data_block(self, block_number, block_data):
		# Return the possible error, if no error, set retErr to 0
		try:
			return
pickle.loads(self.proxy.update_data_block(pickle.dumps(block_number),
pickle.dumps(block_data)))
		except xmlrpclib.Error as err:
			print "A fault occurred in client_stub.update_data_block()"
			print "Fault code: %d" % err.faultCode
			print "Fault string: %s" % err.faultString
			quit()


	#REQUEST TO UPDATE THE UPDATED INODE IN THE INODE TABLE FROM SERVER
	def update_inode_table(self, inode, inode_number):
		# Return the possible error, if no error, set retErr to 0
		try:
			return pickle.loads(self.proxy.update_inode_table(pickle.dumps(inode),
pickle.dumps(inode_number)))
		except xmlrpclib.Error as err:
			print "A fault occurred in client_stub.update_inode_table()"
			print "Fault code: %d" % err.faultCode
			print "Fault string: %s" % err.faultString
			quit()


	#REQUEST FOR THE STATUS OF FILE SYSTEM FROM SERVER
	def status(self):
		# Return the status string after marshalling the data
		try:
			return pickle.loads(self.proxy.status())
		except xmlrpclib.Error as err:
			print "A fault occurred in client_stub.status()"
			print "Fault code: %d" % err.faultCode
			print "Fault string: %s" % err.faultString
			quit()


# server_stub.py
# Christopher Brant
# University of Florida
# Fall 2019
```

Christopher Brant
EEL 5737
Due 10/18/19

```
# EEL 5737 PoCSD
# HW4 Part B
import xmlrpclib
from SimpleXMLRPCServer import SimpleXMLRPCServer

import time, Memory, pickle , InodeOps, config

filesystem = Memory.Operations()

# FUNCTION DEFINITIONS
# INITIALIZE THE FILESYSTEM
def Initialize():
        # Marshal the response of Memory.Initialize()
        retVal = Memory.Initialize()
        retVal = pickle.dumps(retVal)
        return retVal


#GIVES ADDRESS OF INODE TABLE
# Not so sure this needs to be implemented in this specific version
def addr_inode_table():
        return pickle.dumps(sblock.ADDR_INODE_BLOCKS)


#RETURNS THE DATA OF THE BLOCK
def get_data_block(block_number):
        # Unmarshal the data for block_number
        return pickle.dumps(filesystem.get_data_block(pickle.loads(block_number)))


#RETURNS THE BLOCK NUMBER OF AVAIALBLE DATA BLOCK
def get_valid_data_block():
        # Unmarshal the data and call the Memory Operations function
        return pickle.dumps(filesystem.get_valid_data_block())


#REMOVES THE INVALID DATA BLOCK TO MAKE IT REUSABLE
def free_data_block(block_number):
        # Marshal the input and call filesystem
        return pickle.dumps(filesystem.free_data_block(pickle.loads(block_number)))


#WRITES TO THE DATA BLOCK
```

```
def update_data_block(block_number, block_data):
	# Marshal the input and call filesystem
	return pickle.dumps(filesystem.update_data_block(pickle.loads(block_number),
pickle.loads(block_data)))


#UPDATES INODE TABLE WITH UPDATED INODE
def update_inode_table(inode, inode_number):
	# Marshal the input and call filesystem
	return pickle.dumps(filesystem.update_inode_table(pickle.loads(inode),
pickle.loads(inode_number)))


#RETURNS THE INODE FROM INODE NUMBER
def inode_number_to_inode(inode_number):
	# Marshal the input and call filesystem
	return pickle.dumps(filesystem.inode_number_to_inode(pickle.loads(inode_number)))


#SHOWS THE STATUS OF DISK LAYOUT IN MEMORY
def status():
	# Marshal the input and call filesystem
	return pickle.dumps(filesystem.status())


# Begin server listening
server = SimpleXMLRPCServer(("",8000))
print ("Listening on port 8000...")

# Registering all server functions below
server.register_multicall_functions()
server.register_function(Initialize,			"Initialize")
server.register_function(addr_inode_table,		"addr_inode_table")
server.register_function(get_data_block,		"get_data_block")
server.register_function(get_valid_data_block,	"get_valid_data_block")
server.register_function(free_data_block,		"free_data_block")
server.register_function(update_data_block,		"update_data_block")
server.register_function(update_inode_table,	"update_inode_table")
server.register_function(inode_number_to_inode, "inode_number_to_inode")
server.register_function(status,				"status")

# Run the server
server.serve_forever()
```