**Test and describe the behavior of your system when there is a failure of the network.**

I have included error handling to the system when there is a failure of the network. When the network fails, there is a message that is printed to the console informing the user of the client software that the server has failed. The client program is then terminated.

**Experiment and compare the performance of the client/service approach vs. the local memory approach for a test case of your choosing.**

I timed my “happy path” test – included in my submission. This test took 1.06220817566 seconds for the client/service implementation and 0.153266906738 seconds for the local memory implementation.

**Describe the design of your implementation and tests you have conducted to check the functionality of your code.**

No more features were added to the file system in this homework – only changing where the data is stored. Thus, I was able to test with the same testing platform and test cases that I used for the last homework. For the last homework, I enumerated several top layer procedures that I believed could induce edge cases. These test cases included creating files/directories, reading/writing to files, moving files/directories sequenced in different orders. This test platform was ran against the new implementation of the file system and worked flawlessly. I also tested the behavior of the client when the server faults – this was done by stopping the client and cutting the server and then running the client again.

client\_stub.py

*# SKELETON CODE FOR CLIENT STUB HW4*

import xmlrpclib, config, pickle

*class* client\_stub():

*def* \_\_init\_\_(*self*):

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

*def* inode\_number\_to\_inode(*self*, *inode\_number*):

        inode\_number = pickle.dumps(inode\_number)

        try:

            respVal = *self*.proxy.inode\_number\_to\_inode(inode\_number)

            respVal = pickle.loads(respVal)

        except *Exception*:

            print("Server Error - terminating program.")

            quit()

        return respVal

*def* get\_data\_block(*self*, *block\_number*):

        block\_number = pickle.dumps(block\_number)

        try:

            respVal = *self*.proxy.get\_data\_block(block\_number)

            respVal = pickle.loads(respVal)

        except *Exception*:

            print("Server Error - terminating program.")

            quit()

        return respVal

*def* get\_valid\_data\_block(*self*):

        try:

            respVal = *self*.proxy.get\_valid\_data\_block()

            respVal = pickle.loads(respVal)

        except *Exception*:

            print("Server Error - terminating program.")

            quit()

        return respVal

*def* free\_data\_block(*self*, *block\_number*):

        block\_number = pickle.dumps(block\_number)

        try:

            respVal = *self*.proxy.free\_data\_block(block\_number)

            respVal = pickle.loads(respVal)

        except *Exception*:

            print("Server Error - terminating program.")

            quit()

        return respVal

*def* update\_data\_block(*self*, *block\_number*, *block\_data*):

        block\_number = pickle.dumps(block\_number)

        block\_data = pickle.dumps(block\_data)

        try:

            respVal = *self*.proxy.update\_data\_block(block\_number, block\_data)

            respVal = pickle.loads(respVal)

        except *Exception*:

            print("Server Error - terminating program.")

            quit()

        return respVal

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

        inode = pickle.dumps(inode)

        inode\_number = pickle.dumps(inode\_number)

        try:

            respVal = *self*.proxy.update\_inode\_table(inode, inode\_number)

            respVal = pickle.loads(respVal)

        except *Exception*:

            print("Server Error - terminating program.")

            quit()

        return respVal

*def* status(*self*):

        try:

            respVal = *self*.proxy.status()

            respVal = pickle.loads(respVal)

        except *Exception*:

            print("Server Error - terminating program.")

            quit()

        return respVal

*# example provided for initialize*

*def* Initialize(*self*):

        try :

*self*.proxy.Initialize()

        except *Exception*:

            print("Server Error - terminating program.")

            quit()

server\_stub.py

*# SKELETON CODE FOR SERVER STUB HW4*

import xmlrpclib

from SimpleXMLRPCServer import SimpleXMLRPCServer

import time, Memory, pickle , InodeOps, config, DiskLayout

filesystem = Memory.Operations()

*# FUNCTION DEFINITIONS*

*def* Initialize():

    print("Client request received - Initialize.")

    retVal = Memory.Initialize()

    retVal = pickle.dumps(retVal)

    print("Memory Initialized!")

    return retVal

*#FETCH THE INODE FROM INODE NUMBER*

*def* inode\_number\_to\_inode(*inode\_number*):

    print("Client request received - inode\_number\_to\_inode.")

    try:

        inode\_number = pickle.loads(inode\_number)

    except PickleError:

        print("Unable to unmarshal data from client.")

        return pickle.dumps(-1)

    retVal = filesystem.inode\_number\_to\_inode(inode\_number)

    retVal = pickle.dumps(retVal)

    return retVal

*#REQUEST THE DATA*

*def* get\_data\_block(*block\_number*):

    print("Client request received - get\_data\_block")

    try:

        block\_number = pickle.loads(block\_number)

    except PickleError:

        print("Unable to unmarshal data from client.")

        return pickle.dumps(-1)

    retVal = ''.join(filesystem.get\_data\_block(block\_number))

    retVal = pickle.dumps(retVal)

    return retVal

*#REQUESTS THE VALID BLOCK NUMBER*

*def* get\_valid\_data\_block():

    print("Client request received - get\_valid\_data\_block")

    retVal = ( filesystem.get\_valid\_data\_block() )

    retVal = pickle.dumps(retVal)

    return retVal

*#REQUEST TO MAKE BLOCKS RESUABLE AGAIN*

*def* free\_data\_block(*block\_number*):

    print("Client request received - free\_data\_block")

    try:

        block\_number = pickle.loads(block\_number)

    except PickleError:

        print("Unable to unmarshal data from client.")

        return pickle.dumps(-1)

    retVal = filesystem.free\_data\_block((block\_number))

    retVal = pickle.dumps(retVal)

    return retVal

*#REQUEST TO WRITE DATA*

*def* update\_data\_block(*block\_number*, *block\_data*):

    print("Client request received - update\_data\_block")

    try:

        block\_number = pickle.loads(block\_number)

        block\_data = pickle.loads(block\_data)

    except PickleError:

        print("Unable to unmarshal data from client.")

        return pickle.dumps(-1)

    retVal = filesystem.update\_data\_block(block\_number, block\_data)

    retVal = pickle.dumps(retVal)

    return retVal

*#REQUEST TO UPDATE THE UPDATED INODE IN THE INODE TABLE*

*def* update\_inode\_table(*inode*, *inode\_number*):

    print("Client request received - update\_inode\_table")

    try:

        inode = pickle.loads(inode)

        inode\_number = pickle.loads(inode\_number)

    except PickleError:

        print("Unable to unmarshal data from client.")

        return pickle.dumps(-1)

    retVal = filesystem.update\_inode\_table(inode, inode\_number)

    retVal = pickle.dumps(retVal)

    return retVal

*#REQUEST FOR THE STATUS OF FILE SYSTEM*

*def* status():

    print("Client request received - status")

    retVal = filesystem.status()

    retVal = pickle.dumps(retVal)

    return retVal

server = SimpleXMLRPCServer(("",8000))

print ("Listening on port 8000...")

*# REGISTER FUNCTIONS*

server.register\_function(Initialize, "Initialize")

server.register\_function(inode\_number\_to\_inode, "inode\_number\_to\_inode")

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(status, "status")

*# run the server*

server.serve\_forever()