John Hodson
Principles of Computer System Design
Homework 4 Design

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()
```