Steven McVicker

Joseph Goergen

April 9th, 2018

TFTP Client Write-up

During our project, we ran into many problems that we had to work through. Our first major issue was finding out what classes were available and which ones we were going to use. We solved that by going through java documentation online and importing what we needed as we use it, for example the Datagram packet requires an InetAddress class so we imported that as well. Our next big issue came with receiving an acknowledgement packet and their port numbers. After Re-reading the RFC a few times we caught on to the pattern of the port destination and source, as well as the java functions that allowed us to grab those port numbers. We found out that the source port of the packet is our destination port. Also, after we sent the first read/write request, we would continually get an error message back. This error message was because we misplaced the 0 before our mode in the request byte array. We fixed this by making sure we placed the 0 in the correct spot. Once we got passed the initial send and receive and ack, it worked until we had to decide what was going to be the last packet. We were using the Datagram packet object and we did a “.data” which gave us an array of bytes including the 0’s, giving us a full packet length. Later we found that the object had a “.length” that did not include the extra 0’s at the end. This allowed us to be able to say if it was less than 512 bits then it was the last packet. One of the biggest issues we ran into for when we were sending a file was with the block numbers. Since there were two independent bytes for one block number, we had to create a function that took in a byte array and returned a byte array where it increments the value by one and if the rightmost byte reaches its max, then it adds one to the leftmost byte and sets the rightmost byte to zero. When we had to write back to the file we kept getting unknown chars in our file. The input stream object that we were using counted the opcode and block number (the first 4 bytes) and we did not account for that. We solved this by using the “.read” function from the input stream object and have it offset by 4 every time it reads.

There have been some parts of Java that made this task easy and difficult at the same time. It has to do with not being familiar with the byte array output stream and how to manipulate file structure with that object. With Java, there is so many ways to do the same thing, which can lead to confusion when trying to find a straight forward answer of something simple such as file manipulation.

We tested our client by setting up a TFTP server that we found online at <http://ww2.unime.it/flr/tftpserver/>. We have Macs so this server worked for both of us. We put up a test text file that we could run the program with. We used Wireshark for most of our debugging and testing. Wireshark helped tremendously and we could see all the information that we needed to. Our first tests were just seeing if we got a response back after our read/write request. We got this working, and our packet that we got back from the server was just an error packet which Wireshark allowed us to see. Later on, we tested again with our Acknowledgment packets and received the first block of data back. We then had to increment the blocks and finish properly reading the data packets into a file. Once we were finished, we could see the entire file being sent over packet by packet with all the correct order of data packets and Ack packets.

We installed the FindBugs plugin for java, and saw some minor issues with our code. FindBugs reported our first issue as class names should start with an uppercase letter. We fixed that with an easy name change. The next warning was that file may fail to close stream. We fixed this by putting a inputstream.close at the end of the method. Lastly, it gave us a warning about not having a default case for our switch statement. These were minor issues, however in some cases could cause issues.

import java.io.ByteArrayOutputStream;

import java.io.DataOutputStream;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

import java.net.SocketException;

import java.net.UnknownHostException;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.Scanner;

import com.sun.xml.internal.bind.v2.runtime.unmarshaller.XsiNilLoader.Array;

public class Tftpclient {

static byte opcode;

static String fileName = "testfile.txt";

static String serverAddress = "10.19.80.42";

static DatagramPacket packetToSend;

static InetAddress InetServerAddress;

static DatagramPacket packetToRecieve;

static DatagramSocket udpPacketSender;

static ByteArrayOutputStream byteStream;

static Boolean noError;

static String mode;

public static void main(String[] args) throws IOException {

noError = true;

mode = "octet";

InetServerAddress = InetAddress.getByName(serverAddress);

udpPacketSender = new DatagramSocket();

Scanner scan = new Scanner(System.in);

System.out.println("What would you like to do? \n 1) Read File \n 2) Write File");

int choice = scan.nextInt();

switch (choice) {

case 1:readFile();break;

case 2:writeFile();break;

default: break;

}

}

public static void recieveAck() throws IOException {

byte[] rpacket = new byte[516];

packetToRecieve = new DatagramPacket(rpacket, rpacket.length, InetServerAddress, udpPacketSender.getLocalPort());

udpPacketSender.receive(packetToRecieve);

}

public static void readFile() throws IOException {

opcode = 1;

byte[] packet =  requestPacketByteArray(opcode, fileName);

packetToSend = new DatagramPacket(packet, packet.length, InetServerAddress, 69);

udpPacketSender.send(packetToSend);

byteStream = recieveFile();

OutputStream outputStream = new FileOutputStream(fileName);

byteStream.writeTo(outputStream);

byteStream.close();

}

public static byte[] requestPacketByteArray(byte opcode, String fileName) {

byte[] result = new byte[2 + fileName.length() + 1 + mode.length() + 1];

int i = 0;

result[i] = 0;

i++;

result[i] = opcode;

i++;

for(int k = 0; k < fileName.length(); k++,i++) {

result[i] = (byte) fileName.charAt(k);

}

result[i] = 0;

i++;

for (int k = 0; k < mode.length(); k++,i++)

result[i] = (byte) mode.charAt(k);

result[i] = 0;

i++;

return result;

}

public static byte[] makeDataPacket(int dataSize, byte[] blockNum) {

byte[] result = new byte[dataSize];

blockNum = incrementBlock(blockNum);

int i = 0;

result[i] = 0;

i++;

result[i] = 3;

i++;

result[i] = blockNum[0];

i++;

result[i] = blockNum[1];

i++;

return result;

}

public static void sendAck(byte[] blockNum) throws IOException {

byte[] something = new byte[4];

something[0] = 0;

something[1] = 4;

something[2] = blockNum[2];

something[3] = blockNum[3];

packetToSend = new DatagramPacket(something, something.length, InetServerAddress, packetToRecieve.getPort());

udpPacketSender.send(packetToSend);

}

public static ByteArrayOutputStream recieveFile() throws IOException {

ByteArrayOutputStream result = new ByteArrayOutputStream();

do {

byte[] rpacket = new byte[516];

packetToRecieve = new DatagramPacket(rpacket, rpacket.length, InetServerAddress, udpPacketSender.getLocalPort());

udpPacketSender.receive(packetToRecieve);

ArrayList<Byte> conversionArray = new ArrayList<Byte> ();

int numOfConversion = 0;

for (int i = 0; i < packetToRecieve.getLength(); i++) {

byte[] tempArray = packetToRecieve.getData();

if (i > 3 && tempArray[i] == 10 && tempArray[i-1] != 13) {

conversionArray.add((byte)13);

numOfConversion++;

}

conversionArray.add(tempArray[i]);

}

byte[] arrayToWriteBack = new byte[conversionArray.size()];

for (int i = 0; i < conversionArray.size(); i++) {

arrayToWriteBack[i] = conversionArray.get(i);

}

if(packetToRecieve.getData()[1] == 5) {

//System.out.println("Error Code: " + packetToRecieve.getData()[3]);

showError();

noError = false;

}

else if(packetToRecieve.getData()[1] == 3){

DataOutputStream dataOutputStream = new DataOutputStream(result);

//dataOutputStream.write(packetToRecieve.getData());

dataOutputStream.write(arrayToWriteBack, 4, arrayToWriteBack.length - 4);

System.out.println(Arrays.toString(arrayToWriteBack));

System.out.println(Arrays.toString(packetToRecieve.getData()));

sendAck(packetToRecieve.getData());

}

}while (packetToRecieve.getLength() > 512 && noError);

return result;

}

public static void writeFile() throws IOException{

opcode = 2;

byte[] packet =  requestPacketByteArray(opcode, fileName);

packetToSend = new DatagramPacket(packet, packet.length, InetServerAddress, 69);

udpPacketSender.send(packetToSend);

InputStream inputStream = new FileInputStream(fileName);

System.out.println(inputStream.available());

while (inputStream.available() > 0 && noError) {

recieveAck();

opcode = 3;

byte[] tempByteArray = new byte[4 + Math.min(512, inputStream.available())];

tempByteArray =  makeDataPacket(tempByteArray.length, packetToRecieve.getData());

for (int i = 4; i < tempByteArray.length; i++) {

int temp = inputStream.read();

tempByteArray[i] = (byte) temp;

}

//inputStream.read(tempByteArray, 4, tempByteArray.length - 4);

packetToSend = new DatagramPacket(tempByteArray, tempByteArray.length, InetServerAddress, packetToRecieve.getPort());

udpPacketSender.send(packetToSend);

System.out.println(inputStream.available());

if (packetToRecieve.getData()[1] == 5)

{

showError();

noError = false;

}

}

inputStream.close();

}

public static byte[] incrementBlock(byte[] currentBlock) {

byte[] result = new byte[2];

currentBlock[3]++;

if(currentBlock[3] < 0) {

currentBlock[2]++;

currentBlock[3] = 0;

}

result[0] = currentBlock[2];

result[1] = currentBlock[3];

return result;

}

public static void showError() {

switch(packetToRecieve.getData()[3]) {

case 0:System.out.println("Not defined, see error message (if any).");break;

case 1:System.out.println("File not found.");break;

case 2:System.out.println("Access violation.");break;

case 3:System.out.println("Disk full or allocation exceeded.");break;

case 4:System.out.println("Illegal TFTP operation.");break;

case 5:System.out.println("Unknown transfer ID.");break;

case 6:System.out.println("File already exists.");break;

case 7:System.out.println("No such user.");break;

default: break;

}

}

}