Linnéuniversitetet
Kalmar Växjö

Report

Assignment 3

1DV701

Author: Marteinn Hjaltason
Bryan Besong
Semester: Spring 2024
Email: mh226eh@student.lnu.se
bb222qj@student.lnu.se

Contents

1 Problem 1

1.1 Preface

We carried out all the testing using the python code given to us for the tests. This means we have no screenshots. We hope this will suffice to adequately report our thought process.

1.2 Discussion

Our receiveFrom method was implemented such that it gets the socket address of the the client connecting to it as this address is crucial to writing and reading data in the server

Using buffer indexing and format specification(0xff), we get the 2 first bytes and combine them with “|” to give us the opcode of the message from the client. From the 3rd byte, we loop through all the bytes until we get to the one whose value is 0, implying the end of the filename, such that all of the already visited bytes make up the filename. In the event of an opcode we don't recognise, we set the opcode the the predefined OP_ERR

d. We create the response packet by reading the target file and dividing it into blocks of max 512 bytes and then send them one by one in a loop for as many blocks as there are using our sendBlockAndWaitForAck that takes the block to be sent and a value for the timeout. The datagram packet is prepared in the method and then we set an acknowledgment flag to false such that the flag is not true(while we have not gotten an ACK back), the datagram packet should keep being sent. A snippet of the method is below.

```
   private boolean sendBlockAndWaitForAck(DatagramSocket sendSocket, InetSocketAddress clientAddress, byte[] dataBlock, int blockNumber, int timeout) throws IOException {
       byte[] sendData = new byte[dataBlock.length + 4]; // Plus 4 for the opcode and block number
       sendData[0] = 0;
       sendData[1] = OP_DAT; // DATA opcode
       sendData[2] = (byte) ((blockNumber >> 8) & 0xff); // High byte of block number
       sendData[3] = (byte) (blockNumber & 0xff); // Low byte of block number
       System.arraycopy(dataBlock, 0, sendData, 4, dataBlock.length);

       byte[] ackBuf = new byte[4]; // Buffer for receiving ACKs
       DatagramPacket ackPacket = new DatagramPacket(ackBuf, ackBuf.length);

       sendSocket.setSoTimeout(timeout);

       boolean ackReceived = false;
```

```
        while (!ackReceived) {
            try {
                // Sending packet
                System.out.println("Sending packet for block number " +
blockNumber);
                DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, clientAddress);
                sendSocket.send(sendPacket);

                // Attempting to receive ACK
                sendSocket.receive(ackPacket);

                // Check if the received packet is an ACK for the correct block
                if (ackBuf[1] == OP_ACK && ((ackBuf[2] & 0xff) << 8 |
(ackBuf[3] & 0xff)) == blockNumber) {
                    System.out.println("ACK received for block number " +
blockNumber);
                    ackReceived = true;
                }
            } catch (java.net.SocketTimeoutException e) {
                // Handle the timeout specifically
                System.out.println("Timeout waiting for ACK for block number "
+ blockNumber + ", resending packet.");
                // The packet will be resent on the next loop iteration
            }
        }

        return ackReceived;
    }
```

Doing it this way by looping through the blocks of the file meant that it would handle less than 512 bytes and more 512 bytes consistently as all it meant was just a one block in the case of the former and several with the last one between 0 and 511 bytes in the case of the latter

For writing the data from the client, the aim was to reverse the above process,get the blocks and pack them up, send ACKs for all of them and then write them into a file block by block. So we made use of 2 helper methods, the sendAck that takes the socket address and the block number of the block it is acknowledging and a writeDataToFile that takes an array of bytes blocks and filename string and writes them to the file. Below is the code for these methods.

```
    private void sendAck(DatagramSocket socket, InetSocketAddress
clientAddress, int blockNumber) throws IOException {
        byte[] ackPacket = new byte[4];
        ackPacket[0] = 0;
        ackPacket[1] = 4; // Opcode for ACK
        ackPacket[2] = (byte) (blockNumber >> 8);
        ackPacket[3] = (byte) (blockNumber);

        DatagramPacket packet = new DatagramPacket(ackPacket, ackPacket.length,
clientAddress.getAddress(),
                clientAddress.getPort());
        socket.send(packet);
    }
```

```
    public void writeDataToFile(List<byte[]> receivedData, String
requestedFile) throws IOException {
        try (FileOutputStream fos = new FileOutputStream(requestedFile)) {
            for (byte[] dataBlock : receivedData) {
                fos.write(dataBlock);
                System.out.println("writing to: " + requestedFile);
            }
        }
        catch (IOException e) {
            e.printStackTrace();
        }
    }
```

From our observation, The purpose of **socket** is to wait for initial contact from clients. Once a request is received, the server determines the type of request (read or write) and proceeds accordingly.
The **sendSocket** is dedicated to handling the communication with the client for the duration of the file transfer. It is used to send data packets to the client for read requests or receive data packets from the client for write requests. From this, it seems that using a separate sendSocket for each request allows the server to handle multiple clients concurrently without blocking the main listening socket.
.

2 Problem 2

2.1 Discussion

The timeouts and retransmissions were tested using the last 2 tests in the python tests provided to us. We used print statements to visualize the failed transmissions due to timeouts as well as the successful transmission on the arrival of the correct acknowledgement. This is the snippet of the timeout and retransmission logic.

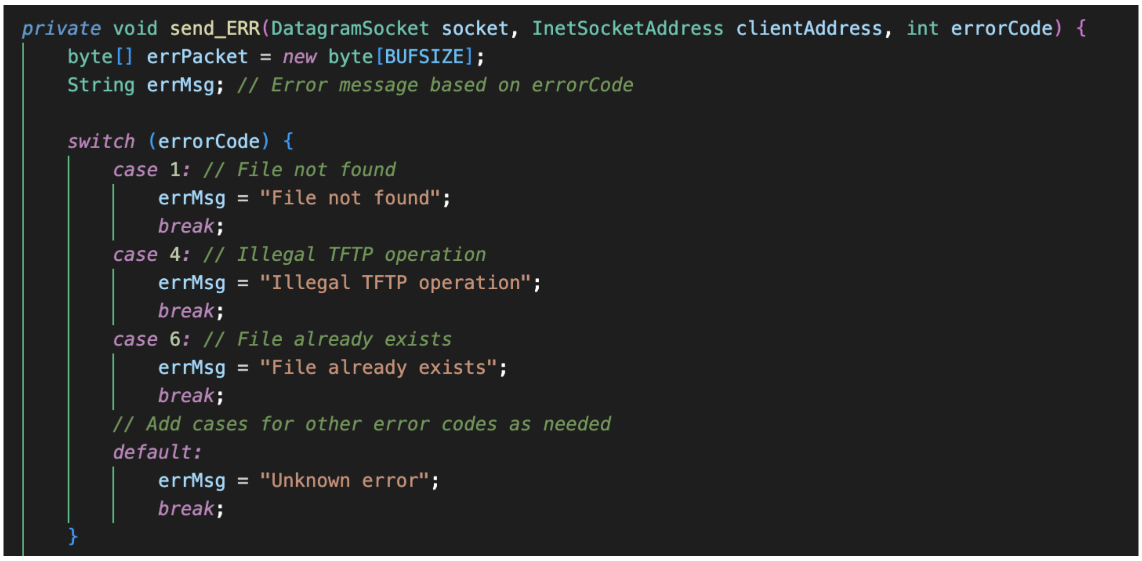
```
while (!ackReceived) {
            try {
                // Sending packet
```

```
                    System.out.println("Sending packet for block number " +
blockNumber);
                    DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, clientAddress);
                    sendSocket.send(sendPacket);

                    // Attempting to receive ACK
                    sendSocket.receive(ackPacket);

                    // Check if the received packet is an ACK for the correct
block
                    if (ackBuf[1] == OP_ACK && ((ackBuf[2] & 0xff) << 8 |
(ackBuf[3] & 0xff)) == blockNumber) {
                        System.out.println("ACK received for block number " +
blockNumber);
                        ackReceived = true;
                    }
                } catch (java.net.SocketTimeoutException e) {
                    // Handle the timeout specifically
                    System.out.println("Timeout waiting for ACK for block
number " + blockNumber + ", resending packet.");
                    // The packet will be resent on the next loop iteration
                }
```

3 Problem 4



```
private void send_ERR(DatagramSocket socket, InetSocketAddress clientAddress, int errorCode) {
    byte[] errPacket = new byte[BUFSIZE];
    String errMsg; // Error message based on errorCode

    switch (errorCode) {
        case 1: // File not found
            errMsg = "File not found";
            break;
        case 4: // Illegal TFTP operation
            errMsg = "Illegal TFTP operation";
            break;
        case 6: // File already exists
            errMsg = "File already exists";
            break;
        // Add cases for other error codes as needed
        default:
            errMsg = "Unknown error";
            break;
    }
```

3.1 Discussion

We implemented the TFTP error handling for error codes 0, 1, 2 and 6 by using switch cases to set the error message in the event of a transmission or connection termination. We later set the error packet as shown in the snippet below and send

```
        // Error opcode
        errPacket[0] = 0;
        errPacket[1] = OP_ERR;
        // Error code
        errPacket[2] = (byte) ((errorCode >> 8) & 0xFF);
        errPacket[3] = (byte) (errorCode & 0xFF);
        // Error message
        byte[] errMsgBytes =
errMsg.getBytes(java.nio.charset.StandardCharsets.UTF_8);
        System.arraycopy(errMsgBytes, 0, errPacket, 4, errMsgBytes.length);
        errPacket[4 + errMsgBytes.length] = 0; // Null terminator for the
string

        try {
            DatagramPacket packet = new DatagramPacket(errPacket, 4 +
errMsgBytes.length + 1, clientAddress.getAddress(), clientAddress.getPort());
            socket.send(packet);
        } catch (IOException e) {
            System.err.println("Error sending the error packet: " +
e.getMessage());
            e.printStackTrace();
        }
    }
```

Workload:
Marteinn Hjaltason: 40 %
Bryan Besong: 60 %