Assignment 2

Computer Networks

Introduction

The server starts with it's socket bound to port 9000 instead of the TFTP Port 69 because ports below 1025 throw an exception. It's run method then waits for requests from clients. In the UDP version the client starts with its socket bound to any port between 1025-65,535 complying with the TFTP RFC. The client's run method starts a menu where it can write files to the server by selecting option 1 and read files from the server by selecting option 2 with an unlimited file size. The menu is executed until the user selects option 3 and quits. This protocol has only been tested with text files. Type file1.rtf to store a file < 512 bytes, file2.rtf to store a file > 512 bytes, file3.rtf to retrieve a file < 512 bytes or file 4.rtf to retrieve a file > 512 bytes in the UDP version. In the TCP version type file5.rtf to store a file < 512 bytes, file6.rtf to store a file > 512 bytes, file7.rtf to retrieve a file < 512 bytes or file8.rtf to retrieve a file > 512 bytes.

TFTP-UDP

Sending Request Packets- Client

In the client's run method a request option is chosen followed by the filename which is stored and it calls the sendRequest method defined at line 118 which uses a ByteArrayOutputStream to write the opcode as a byte array, the filename which is converted to bytes using UTF-8 and the mode converted to bytes as defined in the RFC. The ByteArrayOutputStream is then converted to a byte array, encapsulated within a UDP packet and sent to the server with a timer of 10 seconds.

Receiving Request Packets - Server

The Server receives packets at line 76 using a ByteArrayInputStream to read the opcode and extract the port and IP Address between lines 79-85. When the server receives either a WRQ or RRQ it calls extractFileName defined at line 293 where it uses a ByteArrayInputStream to read each filename byte and writes each filename byte to a ByteArrayOutputStream until it reaches a 0 separator byte. The ByteArrayOutputStream is then converted to a string representing the filename.

Sending ACK Packets - Server

When the Server receives a WRQ packet it initialises the block number of 00 and proceeds to line 96 calling sendACK defined at line 257 where a ByteArrayOutputStream is used to write the opcode and the current block number. The ByteArrayOutputStream is then converted into a byte array, encapsulated within a UDP packet and sent to the client with a timer of 10 seconds. The server then creates a FileOutputStream at line 98 to write file data using the extracted filename.

Receiving ACK Packets - Client

The Client receives ACK packets using receiveACK defined at line 207 where a loop starts as the client attempts to receive the packet, if it is not received within the 10 second timer a SocketTimeoutException is thrown where it retransmits the last sent packet with a timer of 10 seconds and increases the number of retries, if the current number of retries is equal to MAX_RETRIES of 5 then it will give up and not retransmit.

Sending Data Packets - Client

Once the Client receives the ACK for a WRQ it calls sendToServer defined at line 149 where it attempts to create a FileInputStream using the supplied filename, if the file isn't found it throws an exception and an error is printed at line 197. If the file is found then it reads each byte of data using the FileInputStream and writes each byte into a ByteArrayOutputStream and increments totalBytesRead. if totalBytesRead reaches 512 then it converts the ByteArrayOutputStream into a byte array, increments the block number using incrementBlockNumber defined at line 260 which increments infinitely but rounds the 2nd byte when it reaches 9 e.g. 09 -> 10, 19 -> 20.

It then sends the data packet using sendDataPacket defined at line 241 where a ByteArrayOutputStream is used to write the opcode, the current block number and the file data. The ByteArrayOutputStream is then converted into a byte array, encapsulated within a UDP packet and sent with a timer of 10 seconds set. It then calls receiveACK and once it's received, it resets the totalBytesRead and the ByteArrayOutputStream and carries on reading the remaining bytes where eventually the loop at line 154 terminates indicating totalBytesRead was less than 512 bytes in which case the last data packet is sent in the same way, then the client waits to receive the last ACK and returns to it's menu to process further requests.

Receiving Data Packets - Server

When the server has received a data packet it proceeds to line 105 where it uses the ByteArrayInputStream to read the block number of the packet and then reads each byte of data. The bytes of data are then sequentially written to the file using the FileOutputStream then incrementing totalBytesRead. Once totalBytesRead reaches 512 the loop terminates and it proceeds to line 127 where it calls sendACK, however if byteRead was 0 at line 114 then totalBytesRead must be less than 512 bytes as it reached a padded 0, in which case it breaks out of the loop and proceeds to lines 133-135 where it calls sendLastACK which sends an ACK packet with no timer because it is not expecting a reply, then closes the input and output streams and returns to wait for more request packets.

Sending Data and Error Packets & Receiving ACK Packets - Server

When the server receives a RRQ packet it proceeds to line 143 where it initialises the block number of 00 then attempts to create a FileInputStream using the supplied filename at line 151, if the file isn't found it proceeds to line 237 where a FileNotFoundException is caught and sendErrorPacket is called which is defined at line 397 where a ByteArrayOutputStream is used to write the opcode of an error packet, error code and error message which is converted into bytes using UTF-8. The ByteArrayOutputStream is then converted to a byte array, encapsulated into a UDP Packet and sent to the client with no timer because it's not expecting a reply. If the file was found it sends data packets in the same way as the client using sendDataPacket defined at line 324 and receives ACK packets in the same way as the client.

Receiving Data and Error Packets & Sending ACK Packets - Client

When the client sends a RRQ packet it calls writeToFile defined at line 280 which receives data packets in the same way as the server. If it receives an error packet it proceeds to line 306 where it closes the ByteArrayInputStream, prints an error message and returns to the menu, however if it's a data packet it proceeds to line 314 and checks if the FileOutputStream hasn't been created, if not then it creates it, and writes the data to it in the same way as the server, calls sendACK defined at line 383 in the same way as the server and finally calls sendLastACK defined at line 403 in the same way as the server and returns to the menu.

TFTP-TCP

Initiating The Connection

When the client sends requests it creates it's socket by setting it's IP Address and binds it to the TFTP Port by calling createSocket defined at line 198, however this time it creates an output stream for sending data to the server called outToServer and an input stream for receiving data from the server called inFromServer both attached to it's socket. When the server is started it also creates a ServerSocket bound to the TFTP Port where it proceeds to line 57 and calls it's accept method which waits to receive requests from the client.

The Client types option 1 or option 2 for a request with the filename and proceeds to the sendRequest method defined at line 96 where the output stream outToServer writes the contents of a TFTP request to the servers input stream. When the server accepts the request it creates a connection socket called slaveSocket and creates an input stream for receiving data from the client called inFromClient and an output stream called outToClient for sending data to the client. This creates a pipe between the client and the server providing an in order reliable byte stream service where the client and server send bytes through these streams with guaranteed delivery, therefore ignoring the need to set timers, send acknowledgements and retransmit bytes.

Sending Data - Client

If the Client sent a WRQ request then it calls sendToServer defined at line 115 which performs in the same way as the UDP version, however this time if the file is found it reads the entire file data from the FileInputStream, writes it to a ByteArrayOutputStream, then it's contents is converted into a byte array and outToServer writes this byte array to the server's input stream. If the file is not found then a FileNotFoundException is thrown and an error message is printed at line 143.

Receiving Requests - Server

The Server proceeds to extract the opcode at line 64 helping to identify if it's a WRQ or a RRQ by reading the first 2 bytes from it's input stream, it then extracts the filename by calling extractFileName defined at line 144. It then calls extractMode defined at line 169 where it reads the remaining number of bytes into a byte array representing the mode but does nothing with it, this is just to comply with the RFC.

Receiving Data - Server

If the Client sent a WRQ then it proceeds to line 77 onwards where it creates a FileOutputStream using the extracted filename and creates a byte array to store the fileData with a size of the remaining number of bytes available and then reads the remaining number of bytes (file data) from inFromClient into the byte array fileData. Finally it writes fileData to the file using the FileOutputStream.

Sending Data and Error Bytes - Server

If the Client sent a RRQ it proceeds to line 95 onwards where it attempts to create a FileInputStream using the extracted filename and if the file is found it uses a ByteArrayOutputStream to write 2 bytes representing the opcode for data then the FileInputStream reads the entire file data into the ByteArrayOutputStream which is converted to a byte array and outToClient writes the byte array to the client's input stream. If the file is not found it proceeds to line 118 where a FileNotFoundException is caught and it uses a ByteArrayOutputStream to write the opcode, error code and error message of an error packet and converts the ByteArrayOutputStream to a byte array and uses outToClient to write this byte array to the client's input stream.

Receiving Data and Errors - Client

If the Client sent a RRQ request it calls writeToFile defined at line 151 where it's input stream reads the first 2 bytes indicating the opcode.

If the client received an error indicating the file was not found on the server then it proceeds to line 161 where it gets the remaining number of bytes available to read, creates a byte array of this size to hold the errorInfo and reads the remaining bytes from it's input stream into the errorInfo byte array but does nothing with it, this is only to comply with the RFC, then an error message is printed stating the file was not found.

If the client received data then it proceeds to line 174 where it creates a FileOutputStream for the file using the supplied filename, gets the remaining number of bytes available to read from it's input stream, creates a byte array of this size to hold the fileData and reads the remaining bytes into the byte array. It then uses the FileOutputStream to write the fileData byte array to the file, prints a message indicating the file has been stored and closes the it's socket and returns to the menu.

References

[1] TFTP RFC, [Online], Available from https://www.ietf.org/rfc/rfc1350.txt [accessed 5th May 2016]

[2] Kurose, J.F and Ross K.W, Computer Networking: A Top Down Approach, 5th Edition