Sun Yul Lee

Whitley Pittman

Moon Hyun Kim

Laboratory Assignment 1 Report

Introduction

Our names are Sun Yul Lee, Whit Pittman, and Moon Hyun Kim, and we are Labs Group 15. For this assignment, we first developed a datagram UDP client and datagram UDP server to simulate the formatted client request and the calculating server that offers the operations specified in the requirements. We also did the same simulation through our development of a client and server using TCP sockets. The source code for the two clients and two servers that we created both compile and execute with components working as desired by specifications as we will further explain in this report.

Files

- UDP - ClientUDP.java, ServerUDP.c
- TCP - ClientTCP.c, ServerTCP.java

System Architecture

We will now specify the languages we used to write our source code. For our system, the UDP client was written in Java, and the UDP server was written in C. Furthermore, the TCP client was written in C, and the TCP server was written in Java. The distinct port number where our servers work is 10025.

- ClientUDP.java - First, this program gets the hostname and port number of server from the user. It creates a socket using DatagramSocket(), then shows the user a list of opcode and their operations. After that, the program asks for user opcode with these prompts: first operand, whether second operand is needed, and if yes, second operand. With these inputs, the client system creates the packet with DatagramPacket(). Then, the system sends the packet to the server using send(). If it fails to receive data from the server using receive() for certain amount of time, then the client system re-sends the packet to server. If it fails to receive data from the server for certain amount of tries, it gives up receiving data from server. If the system

succeeds to receive data, it gets data from received packet and prints contents of the packet and round time. Then, the system asks the user if there is another operation. If yes, the whole process is repeated; and if no, socket is closed by using close().

- ServerUDP.c - First, this program gets the port number of server from the user. Then, the server system creates the address information with the port number using getaddrinfo(). It creates a socket using socket() and binds the socket to port using bind(). If the system gets the packet using recvfrom(), it shows data in the packet and creates a structure with the data and does the calculation using calculation(). After the calculation, the result of the calculation is printed and necessary data is stored in a buffer of packet for client. Then, the system sends the packet to the client using sendto(). The server system socket is never closed. The program is ended by typing “Ctrl + C” in the command line.
- ClientTCP.c - First, the client system gets the hostname and port number from the user. Then, with the inputs, the system creates the address information using getaddrinfo(). The client system creates a socket using socket(). Then, it connects the socket to the server using connect(). After the connection, the system shows the user a list of opcode and their operations. The program asks for user opcode with these prompts: first operand, whether second operand is needed, and if yes, second operand. With these inputs, the client system creates the packet and sends it to the server using write(). Then, the system receives the packet from the server using recv(). If the system succeeds to receive data, it gets data from the received packet and prints contents of the packet and round time. Then, the system asks the user if there is another operation. If yes, the whole process is repeated; and if no, the socket is closed by using close().
- ServerTCP.java - First, this program gets the port number of the server from the user. Then, the server system creates a server socket with the port number using ServerSocket(). It accept a client using accept(). Then, using InputStream.read(), the system gets the packet from the client and shows the content of the packet. After that, the server system does the calculation using calculation() and prints the result

out. Then, it creates a buffer of the packet for the client and sends the packet to client by using OutputStream.write(). When the client socket is closed, the server system disconnects the client's socket. The server system socket is never closed. The program is ended by typing "Ctrl + C" in the command line.

System Description

- UDP

  A step-by-step walkthrough of how to compile and run the ClientUDP.java and ServerUDP.c are shown in Figure 1(a) and 1(b). We tested the programs by using a Tux machine and both programs compiled and ran successfully and correctly. Figure 1(a) shows the flow of the client system, and Figure 1(b) shows the flow of the server system.

```
wcp0012@tux059:~/lab1$ javac ClientUDP.java
wcp0012@tux059:~/lab1$ java ClientUDP tux059 10025
tux059
131.204.14.59
Welcome Client!
************************************
*   Opcode              operation   *
*     0                     +       *
*     1                     -       *
*     2                     |       *
*     3                     &       *
*     4                     >>      *
*     5                     <<      *
*     6                     ~       *
************************************

OpCode(1 byte): 3
Operand1(2 bytes): 1
Operand2 is needed? (yes/no) yes
Operand2(2 bytes): 5
Sending:
        Content: 08 00 03 02 00 01 00 05
        Length: 8
        TML: 08
        RID: 00
        OPCODE: 03
        NUMBER OF OPERAND: 02
        OPERAND1: 0001
        OPERAND2: 0005
        ServerAddress: 131.204.14.59
        Port Number: 10025
Packet is sent

Packet is received

Process time: 211 microsecond

Received: 07 00 00 00 00 00 01
ServerAddress: 131.204.14.59
Port Number: 10025

TML: 7
RID: 0
Error: 0
Result: 1
Round time: 432 microsecond

Is there another operation?(yes/no)
```

Figure1(a)

```
szl0093@tux059:~/lab1$ gcc ServerUDP.c -o ServerUDP
szl0093@tux059:~/lab1$ ./ServerUDP 10025
listener: waiting to recvfrom...
listener: got packet from 131.204.14.59

listener: packet is 8 bytes long
listener: packet contains: 08 00 03 02 00 01 00 05
listener: received:
TML: 08
RID: 00
OPCODE: 03
NUMBER OF OPERAND: 02
OPERAND1: 0001

OPERAND2: 0005

Calculation Result:
TML: 07
RID: 00
ERROR: 00
RESULT: 00000001

Sending: 07 00 00 00 00 00 01

listener: sent packet to 131.204.14.59

listener: waiting to recvfrom...
```

Figure1(b)

- TCP

A step-by-step walkthrough of how to compile and run the ClientTCP.c and ServerTCP.java are shown in Figure 2(a) and 2(b). We tested the programs by using a Tux machine and both programs compiled and ran successfully and correctly. Figure 2(a) shows the flow of the client system, and Figure 2(b) shows the flow of the server system.

```
wcp0012@tux059:~/lab1$ gcc ClientTCP.c -o ClientTCP
wcp0012@tux059:~/lab1$ ./ClientTCP tux059 10025
client: connecting to 131.204.14.59
Welcome client!
************************************
*   Opcode              operation   *
*     0                     +       *
*     1                     -       *
*     2                     |       *
*     3                     &       *
*     4                     >>      *
*     5                     <<      *
*     6                     ~       *
************************************
OpCode(1 byte): 0
Operand1(2 bytes): 9
Second operand?(yes/no): yes
Operand2(2 bytes): 123
Sending:
        Content: 08 00 00 02 00 09 00 7b
        TML: 08
        REQUEST ID: 00
        OPCODE: 00
        NUMBER OF OPERAND: 02
        OPERAND1: 0009
        OPERAND2: 007b
ServerAddress: 131.204.14.59
Port Number: 10025

client: received:
        Content: 07 00 00 00 00 00 84
        TML: 07
        REQUEST ID: 00
        Error: 00
        Result: ffffff84

Round time: 14288.0 microsecond
Do you have another calculation?(yes/no):
```

Figure2(a)

```
szl0093@tux059:~/lab1$ javac ServerTCP.java
szl0093@tux059:~/lab1$ java ServerTCP 10025
Waiting

Handling client at 131.204.14.59 on port 34806

Received
Content: 08 00 00 02 00 09 00 7B

TML: 08
REQUEST ID: 00
OPCODE: 00
NUMBER OF OPERAND: 02
OPERAND1: 0009
OPERAND2: 007b

Calculation Result:
TML: 07
REQUEST ID: 00
ERROR: 00
RESULT: 00000084

Sending: 07 00 00 00 00 00 84

write
Sending Complete
```

Figure2(b)

Experiments

- Assumption

We assume that the user enters valid values for inputs. For example, for a prompt that is asking user to put an opcode, we assume that user puts a 1 byte value. This assumption is for both clients of UDP and TCP.

- Error Condition

1. TML value and size of received data are different.

   EX) If TML is 8 and size of received data is 7, we set error value to 127.

2. Size of received data are not matched with number of operand.

a) If number of operand is 1 and size of received data is not 6, it is an error.

b) if number of operand is 2 and size of received data is not 8, it is an error.

3. Number of operand is not matched with opcode.

a) If number of operand is 1 and opcode is not 6, it is an error.

b) If number of operand is 2 and opcode is not 8, it is an error.

4. Invalid number of operand

EX) If number of operand is other than 1 or 2, it is an error.

5. Invalid opcode

EX) If opcode is not one of 0, 1, 2, 3, 4, 5, and 6, it is an error.

6. Invalid operands for bitwise shift operations

a) If either one of operands is negative, it is an error.

b) If second operand is greater than 31, it is an error.

- Testing examples

1) UDP: Operation: ~(3) ; Expected result(HEX): FFFFFFFC

Figure 3(a)

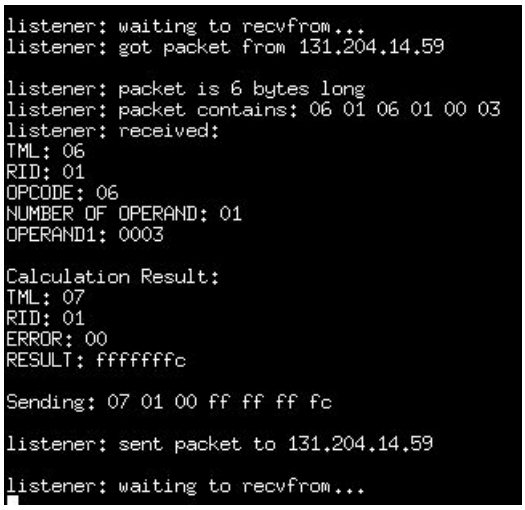

Figure 3(b)

2) UDP: Operation: 1 & 5 ; Expected result(HEX): 00000001

See Figure1(a) and Figure1(b) as a reference for this example.

3) TCP: Operation: ~(12) ; Expected result(HEX): FFFFFFF3

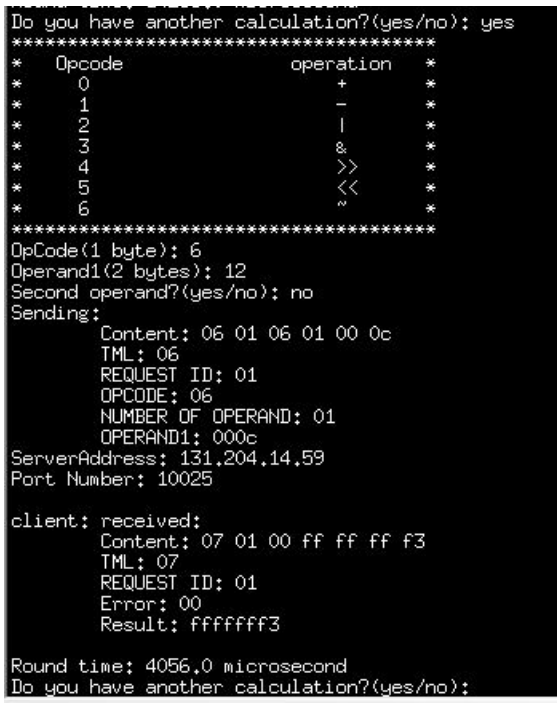

Figure 4(a)

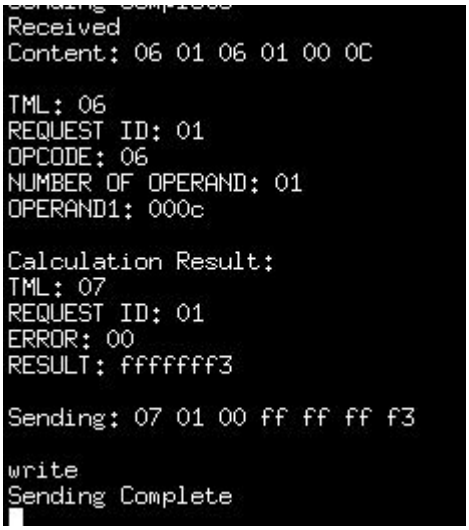

Figure 4(b)

4) TCP: Operation: 9 + 123 ; Expected result(HEX): 00000084

See Figure2(a) and Figure2(b) as a reference for this example.

- Error examples

1) UDP: Operation: 9 – (missing operand) ; Error = 127 is expected

Figure5(a)- see below

Figure5(b)- see below

```
Is there another operation?(yes/no) yes
************************************
*   Opcode              operation   *
*     0                     +       *
*     1                     -       *
*     2                     |       *
*     3                     &       *
*     4                     >>      *
*     5                     <<      *
*     6                     ^       *
************************************

OpCode(1 byte): 1
Operand1(2 bytes): 9
Operand2 is needed? (yes/no) no
Sending:
        Content: 06 02 01 01 00 09
        Length: 6
        TML: 06
        RID: 02
        OPCODE: 01
        NUMBER OF OPERAND: 01
        OPERAND1: 0009
        ServerAddress: 131.204.14.59
        Port Number: 10025
Packet is sent

Packet is received

Process time: 175 microsecond

Received: 07 02 7F 00 00 00 09
ServerAddress: 131.204.14.59
Port Number: 10025

TML: 7
RID: 2
Error: 127
Result: 9
Round time: 282 microsecond

Is there another operation?(yes/no) yes^H^H
Please type yes or no.
Is there another operation?(yes/no) no

Good bye
```

```
listener: waiting to recvfrom...
listener: got packet from 131.204.14.59

listener: packet is 6 bytes long
listener: packet contains: 06 02 01 01 00 09
listener: received:
TML: 06
RID: 02
OPCODE: 01
NUMBER OF OPERAND: 01
OPERAND1: 0009

Calculation Result:
TML: 07
RID: 02
ERROR: 7f
RESULT: 00000009

Sending: 07 02 7f 00 00 00 09

listener: sent packet to 131.204.14.59

listener: waiting to recvfrom...
```

2) TCP: Operation: 8 >> 32; Error = 127 is expected.

```
Do you have another calculation?(yes/no): yes
************************************
*   Opcode              operation   *
*     0                     +       *
*     1                     -       *
*     2                     |       *
*     3                     &       *
*     4                     >>      *
*     5                     <<      *
*     6                     ^       *
************************************
OpCode(1 byte): 4
Operand1(2 bytes): 8
Second operand?(yes/no): yes
Operand2(2 bytes): 32
Sending:
        Content: 08 02 04 02 00 08 00 20
        TML: 08
        REQUEST ID: 02
        OPCODE: 04
        NUMBER OF OPERAND: 02
        OPERAND1: 0008
        OPERAND2: 0020
ServerAddress: 131.204.14.59
Port Number: 10025

client: received:
        Content: 07 02 7f 00 00 00 08
        TML: 07
        REQUEST ID: 02
        Error: 7f
        Result: 00000008

Round time: 3530.0 microsecond
Do you have another calculation?(yes/no): no
Good bye
```

Figure6(a)

```
Received
Content: 08 02 04 02 00 08 00 20

TML: 08
REQUEST ID: 02
OPCODE: 04
NUMBER OF OPERAND: 02
OPERAND1: 0008
OPERAND2: 0020

Calculation Result:
TML: 07
REQUEST ID: 02
ERROR: 7f
RESULT: 00000008

Sending: 07 02 7f 00 00 00 08

write
Sending Complete
Waiting
```

Figure6(b)

Conclusion

In conclusion, after analyzing a testing our code as described above, we have a working communication between our client and calculating server for both TCP and UDP protocols in compliance with the lab specifications. Our programs work between two different tux machines as we tested many times. Also, we learned a lot about the components of datagram socket programming and the functionality of communications between clients and servers through this lab. We were able to simulate the structure of a TCP/UDP client and server and get hands on experience with what we learned in class from the diagrams of the structure of a TCP/UDP system.

References

The main resources we used to guide us in this lab were the assignment specifications, piazza, and the sample source code files that Dr. Biaz provided for us. More specifically, for our C files we expanded on the UDP-server.c and TCP-client.c sample files. For the Java files, we used the code samples from the module 2 powerpoint in Canvas to guide us. We also used the following website to define rules for bitwise shift operations: https://rules.sonarsource.com/c/RSPEC-874.