HO CHI MINH UNIVERSITY OF TECHNOLOGY



**EMBEDDED SYSTEM**

**REPORT LAB 2**

Lecturer: PhD Pham Hoang Anh

Student:

Le Huy Hoang: 1652212

**Contents**

[1. Introduction 1](#_Toc2269325)

[2. Design 1](#_Toc2269326)

[2.1. UDP 1](#_Toc2269327)

[2.1.1. Server side 1](#_Toc2269328)

[2.1.2. Client side 3](#_Toc2269329)

[2.2. TCP 4](#_Toc2269330)

[2.2.1. Server side 4](#_Toc2269331)

[2.2.2. Client 5](#_Toc2269332)

[3. Result 6](#_Toc2269333)

[3.1. UDP server – client 6](#_Toc2269334)

[3.2. TCP single server – client 6](#_Toc2269335)

[3.3. TCP multi server – client 7](#_Toc2269336)

[4. Reference 7](#_Toc2269337)

**Figures**

[Figure 1. Initiation code for UDP server 2](#_Toc2269230)

[Figure 2. Main part of server side 2](#_Toc2269231)

[Figure 3. Initiation of UDP echo client side 3](#_Toc2269232)

[Figure 4. Send and receive from server 3](#_Toc2269233)

[Figure 5. TCP Echo server's init section 4](#_Toc2269234)

[Figure 6. Infinite section of TCP echo server 5](#_Toc2269235)

[Figure 7. Initiation part of TCP echo client 5](#_Toc2269236)

[Figure 8. Main part of TCP echo client 6](#_Toc2269237)

[Figure 9. Screen capture for UDP echo server - client demo, the left terminal is of server, the right one of client 6](#_Toc2269238)

[Figure 10. Screen capture for TCP echo server - client demo, left one is server, right one is client 7](#_Toc2269239)

[Figure 11. Screen capture for TCP echo server - client demo, left one is server, two right ones are clients 7](#_Toc2269240)

**Keynotes**

Times New Roman, font size 12: normal text

Courier New: code

# 1. Introduction

The Intel Edison is a computer-on-module that was offered by Intel as a development system for wearable devices and Internet of Things devices. It runs a small Linux version.

Secure Shell (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network. The best known example application is for remote login to computer systems by users. SSH provides a secure channel over an unsecured network in a client-server architecture, connecting an SSH client application with an SSH server. Common applications include remote command-line login and remote command execution, but any network service can be secured with SSH.

The Spanning Tree Protocol (STP) is a network protocol that builds a loop-free logical topology for Ethernet networks. The basic function of STP is to prevent bridge loops and the broadcast radiation that results from them. Spanning tree also allows a network design to include backup links to provide fault tolerance if an active link fails.

In this report, I use SSH to control Intel Edison and STP to transfer source code from my machine (Linux machine) to Intel Edison as well as enrich knowledge about UNIX network programming. The programs I make are simple version of echo server – client. The only difference between the two is one uses TCP and another uses UDP.

# 2. Design

In the jargon, the design of two program are similar in server side and client side. However, UDP-used program needs a set of built-in functions to implement rather than using directly system calls like TCP-used program.

## 2.1. UDP

### 2.1.1. Server side

Server side of UDP program does not require to handle many incoming connections at the same time.

As other server side programs, mine includes 2 main part:

* Initiation: set up things for server such as socket address, socket port…
* Infinite loop: to handle clients

#### 2.1.1.1. Initiation

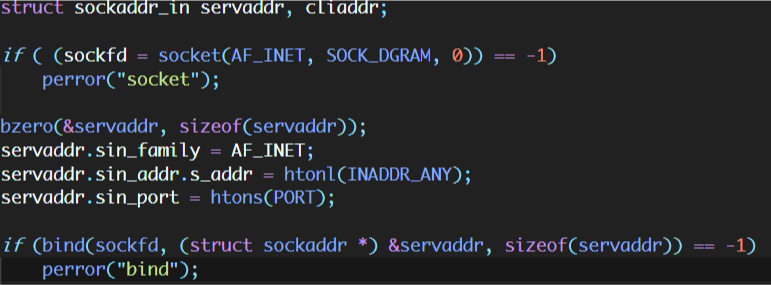


Figure 1. Initiation code for UDP server

Detail of the given snippet code is as follows:

SOCK\_DGRAM: use UDP protocol  
sockaddr\_in: standard struct of socket. Not only server but also client do use this struct for socket.  
AF\_INET: use IPv4  
htonl: convert number in host’s order byte to network’s order byte. Network order byte is little endian while for some host, that can be big endian.  
INADDR\_ANY (= 0): means computer can take any arbitrary address for server since this is localhost server.  
htons: their function is the same as that of htonl but with smaller convert number range  
PORT: this is an given port set by programmer.

#### 2.1.1.2. Infinite loop

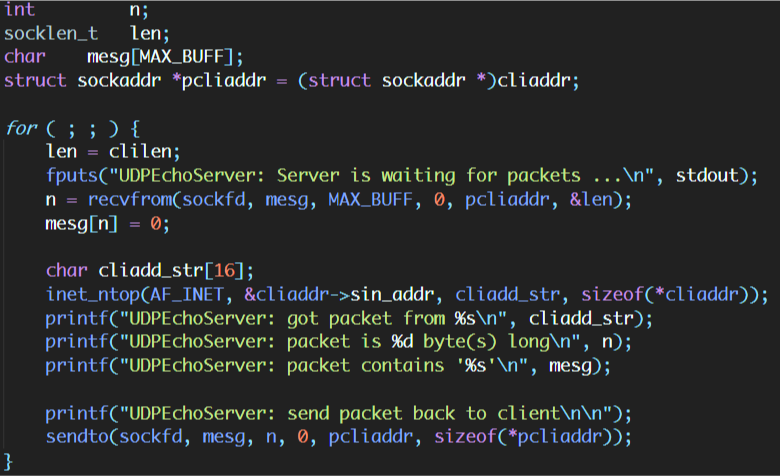


Figure 2. Main part of server side

As I mentioned before, UDP server code and TCP server code are different in some. The first one is UDP server has no **accept** function.

inet\_ntop() function is used to convert IP from numeric format which is saved in struct cliaddr to presentation format which is later stored in string cliadd\_str. This string is declared 16 chars long for the reason that the maximum length of presentation IPv4 IP address is: 255.255.255.255, plus **‘\0’** character will be exactly 16 characters long. The declaration is to ensure this issue.

The second difference between UDP server and TCP server is UDP server code uses recvfrom() and sendto() functions are dedicated for UDP protocol.

### 2.1.2. Client side

#### 2.1.2.1. Initiation

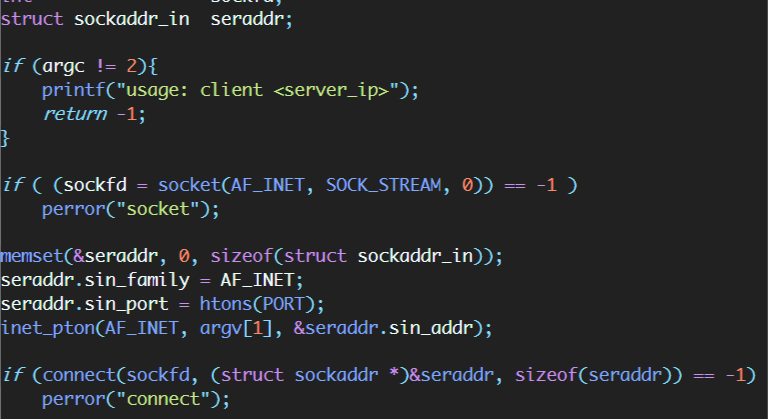


Figure 3. Initiation of UDP echo client side

Basically, the skeleton of client side is not much different with that of the server.

#### 2.1.2.2. Sending to and receiving from server

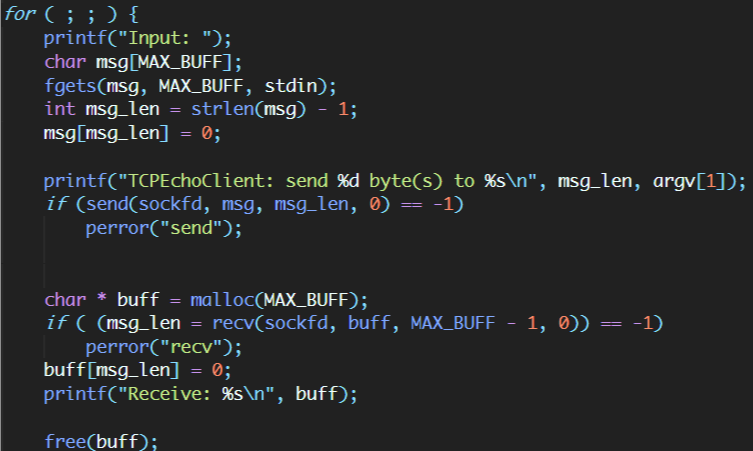


Figure 4. Send and receive from server

Although there exists an infinite for loop, this loop and the loop of server is totally irrelevant in terms of core function. In particular, the loop in client serves the constant read-send-receive-show cycles. Client first reads from **STDIN** (fgets() function, this function reads until reaching **‘\n’** character (Enter button in keyboard)), send to Echo Server via send(). It then receives message by recv(). The client only reads no more than MAX\_BUFF bytes and later shows onto **STDOUT** (in this case is the screen).

For loop of server, on the other hand, is to serve constantly request sent by clients.

## 2.2. TCP

Normally, TCP send-receive functions can utilize system calls (write() and read() respectively), but in this program, I do use built-in functions.

Besides, I will only present the design of the server version that permits many clients connect at the same time. In the summit code, however, I do attach single-request server version.

### 2.2.1. Server side

#### 2.2.1.1. Initiation

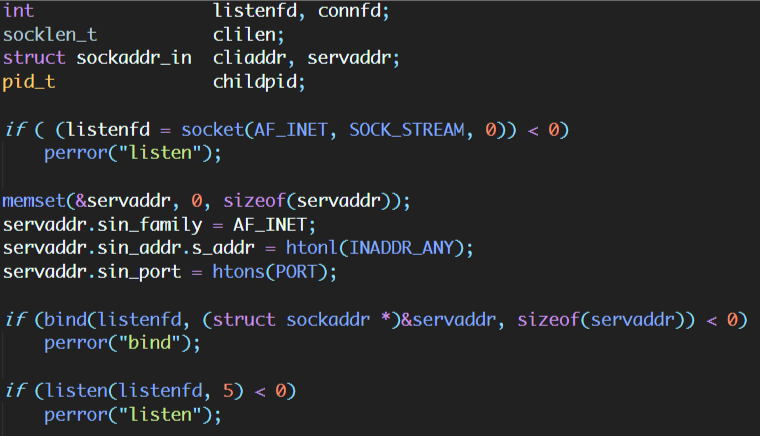


Figure 5. TCP Echo server's init section

It is somewhat similar with UDP server.

SOCK\_STREAM in socket() implies that this server uses TCP protocol.

However, TCP server does have listen() function. The parameter 5 implies the backlog for this socket is 5.

#### 2.2.1.2. Infinite loop

Fig. 6 shows the code of TCP multi-client server. The server, in fact, has 2 different parts serving 2 separated jobs:

1. Accepting incoming connection
2. Serving connection

The first one consists of six first lines of the code. This section is responsible for receiving request from new connection.

The second one is embedded in fork() sub-section. After the above section has received the request, it then creates a new process by fork() and let the newly created process to deal with it. The first section later waits for new request.

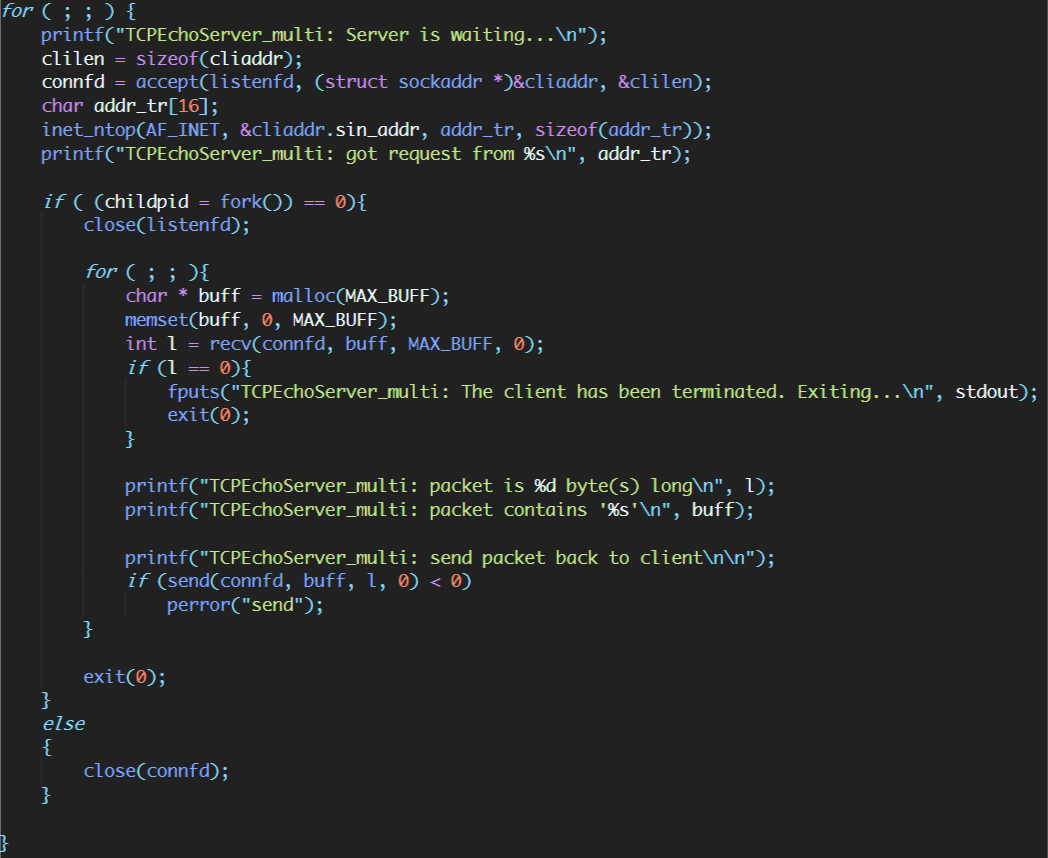


Figure 6. Infinite section of TCP echo server

## 2.2.2. Client

TCP client is similar with UDP client.



Figure 7. Initiation part of TCP echo client

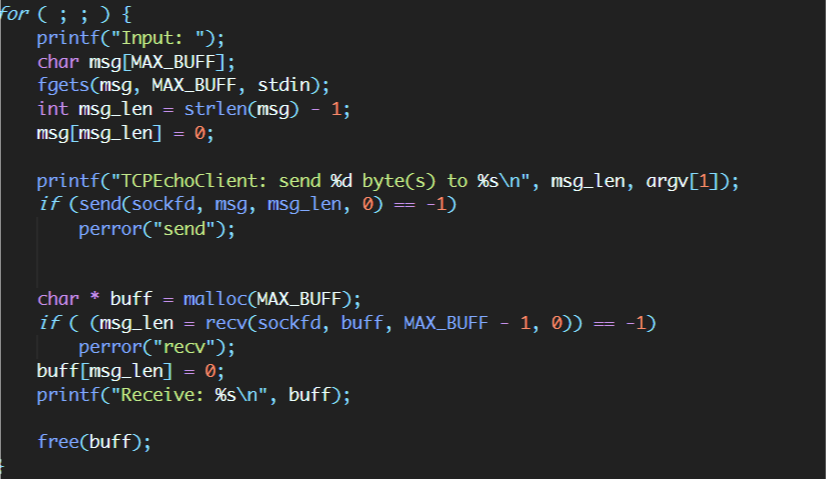


Figure 8. Main part of TCP echo client

# 3. Result

## 3.1. UDP server – client

I created two SSH to Intel Edison, one for server, another for client. Client connects to 127.0.0.1.

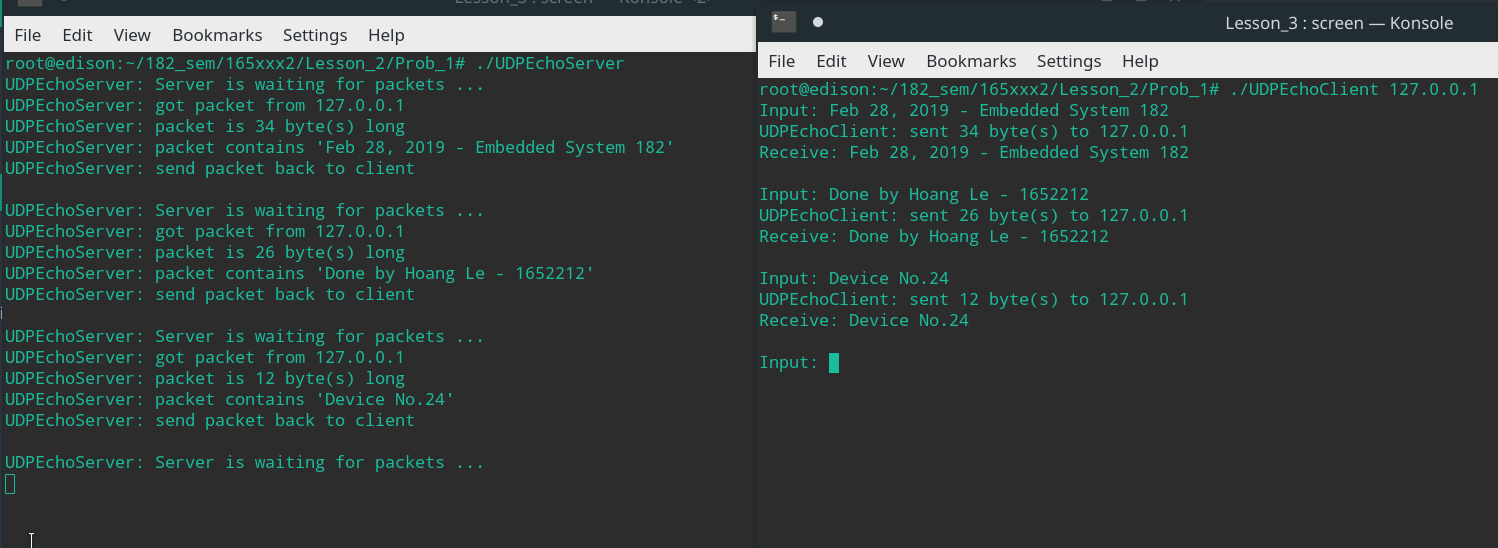


Figure 9. Screen capture for UDP echo server - client demo, the left terminal is of server, the right one of client

## 3.2. TCP single server – client

I did similar with UDP.

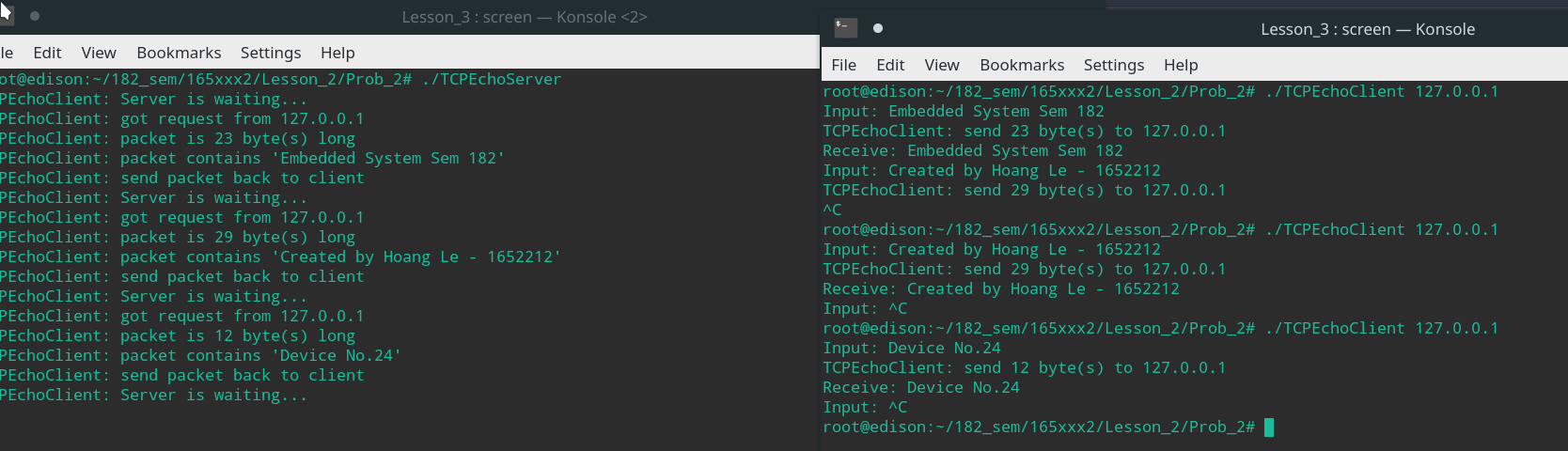


Figure 10. Screen capture for TCP echo server - client demo, left one is server, right one is client

## 3.3. TCP multi server – client

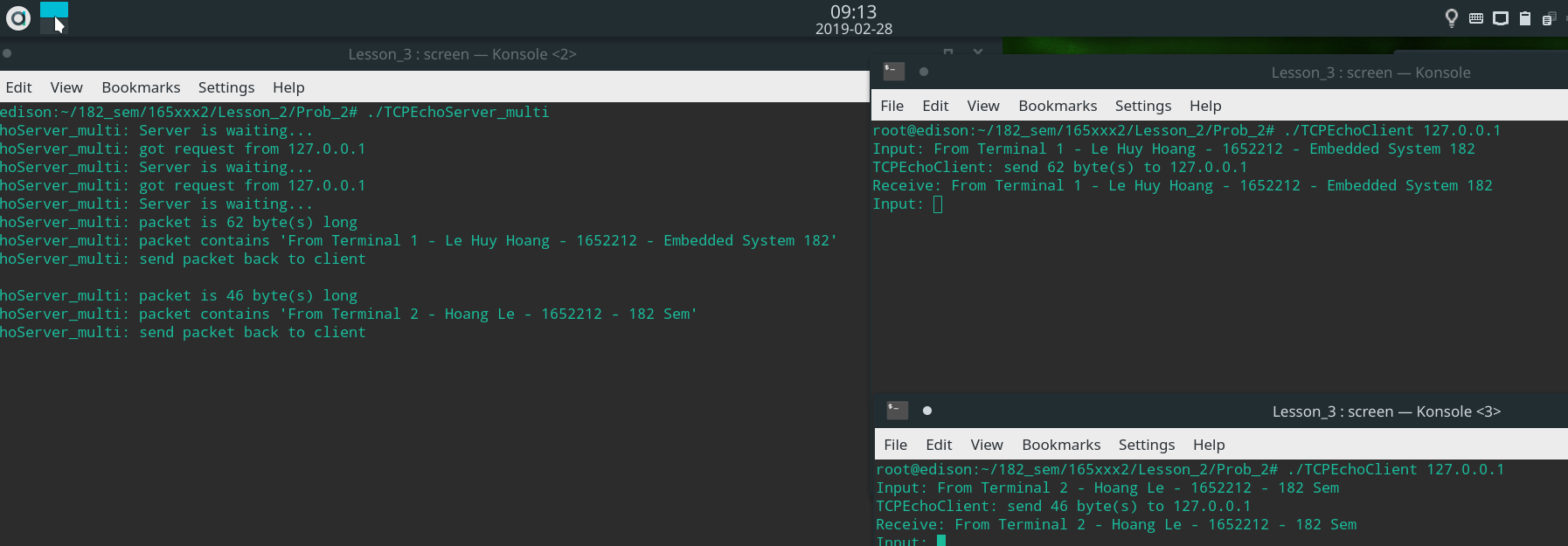


Figure 11. Screen capture for TCP echo server - client demo, left one is server, two right ones are clients

# 4. Reference

Steven - UNIX Network Programming, volume 1