



EE 533: Network Processor Design and Programming

Instructor: Young Cho, Ph.D.

Zhongqi Zhao: 7864803206

0. Github Link

<https://github.com/ZzqXAUT/EE533.git>

1. Objectives

The purpose of this lab is to implement a basic TCP client-server application using the Linux Socket API. The program should:

- (1) Create a TCP server that listens on a specified port.
- (2) Allow a TCP client to connect to the server using hostname and port.
- (3) Enable the client to send messages to the server repeatedly.
- (4) Enable the server to receive client messages, print them to the terminal, and send back a confirmation response.
- (5) Support multiple clients by using fork() so that each client connection is handled by an independent child process while the parent process continues accepting new connections.

2. Environment

Language: C

Platform: Linux

Compiler: GCC

Protocol: TCP (SOCK_STREAM)

Address Family: IPv4 (AF_INET)

3. System Design Overview

This program contains two main components: server design and client design.

3.1 Server Design

The server follows the standard TCP server workflow:

- (1) Create a socket using socket()
- (2) Bind it to a port using bind()
- (3) Start listening using listen()
- (4) Accept client connections using accept()
- (5) For each new connection, call fork():

The child process handles communication with the connected client

The parent process continues calling accept() to wait for more clients

3.2 Client Design

The client performs the following:

- (1) Create a socket using socket()
- (2) Resolve server hostname using gethostbyname()
- (3) Connect to server using connect()
- (4) Repeatedly:

- Read input from the user
- Send it to the server using write()
- Receive a response using read()
- (5) Exit when the user types "exit"

4. Concurrency with fork()

To support multiple clients, the server uses: pid_t pid = fork(). This creates a child process so that each client is handled independently. In child process, the return value of fork() is 0 and in parent process, this value is greater than 0.

The child process does not need the listening socket sockfd. It only communicates through newsockfd. Closing sockfd prevents unnecessary resource usage.

The parent does not communicate with this client. It continues to accept new connections. Closing newsockfd ensures the correct socket lifecycle and avoids descriptor leaks.

5. Results

Client send message to server:

```
zzq@zzq:~/EE533_client$ ./client 192.168.160.129 8080
Please enter the message: hi
I got your message
Please enter the message: hello
I got your message
```

Client terminate connection:

```
Please enter the message: exit
zzq@zzq:~/EE533_client$
```

Server's reply:

```
zzq@zzq:~/EE533/lab1_server$ ./server 8080
Client says: hi

Client says: hello

Client disconnected.
```

Client send message to server:

```
zzq@zzq:~/EE533_client$ ./client 192.168.160.129 8080
Please enter the message: hi
I got your message
Please enter the message: what your name?
I got your message
Please enter the message: exit
zzq@zzq:~/EE533_client$
```

Server's reply:

```
zzq@zzq:~/EE533/lab1_server$ ./server 8080
Client says: hi

Client says: hello

Client disconnected.
```

```
Client says: hi

Client says: what your name?

Client disconnected.
```

Concurrent connections by 2 clients:

```
zzq@zzq:~/EE533_client$ ./client 192.168.160.129 8080
Please enter the message: 123
I got your message
Please enter the message: 
```

```
zzq@zzq:~/EE533_client$ ./client 192.168.160.129 8080
Please enter the message: 456
I got your message
Please enter the message:
```

```
Client says: 123

Client says: 456
```