시스템 프로그래밍 실습

[Assignmen2-1]

Class : **D 반(실습 2 금 56)**

Professor : **최상호 교수님**

Student ID : **2022202104**

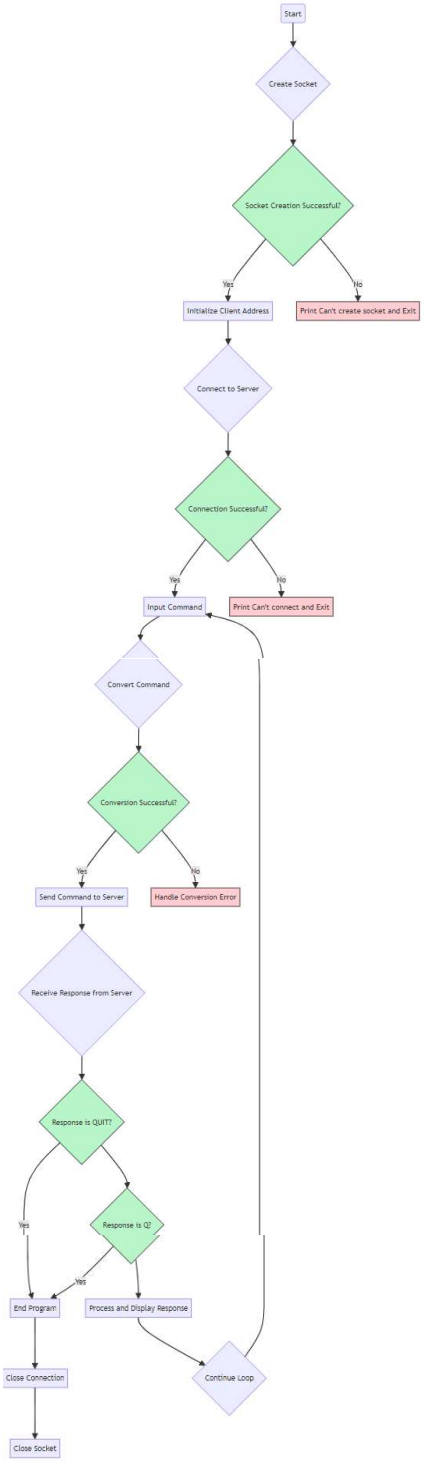
Name : **김유찬**

Introduction

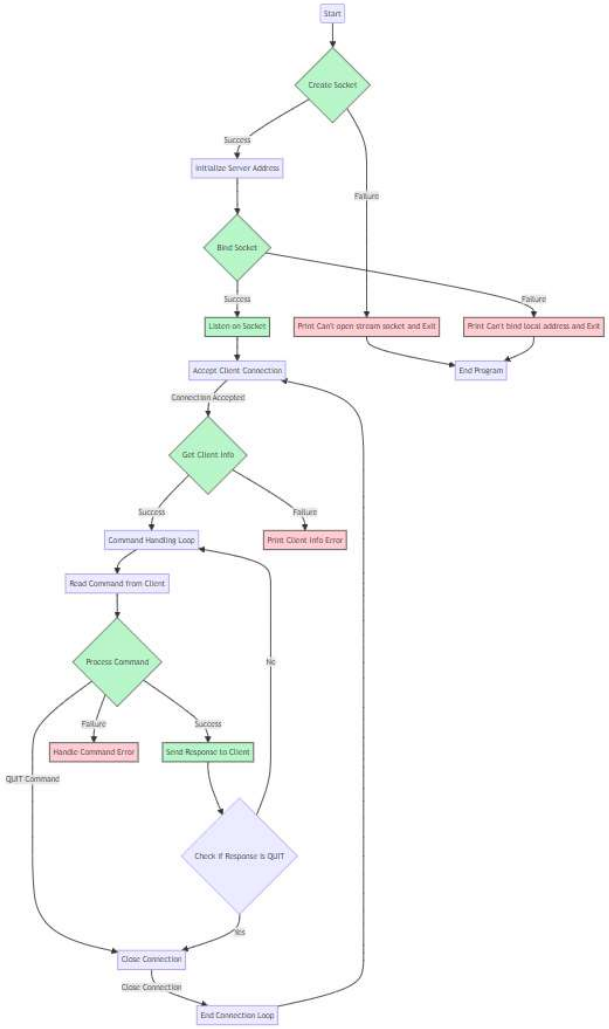
저번 과제에는 client와 server서로 주고 받는 것처럼 했지만 파이프라인으로 client에서 리눅스 명령어를 FTP 명령어로 바꾸고 server 에 입력을 넣어주어 server 에서 그 명령어의 결과를 출력하는 형식으로 했다. 즉 한 컴퓨터에서만 client, server 가 가능한 형태이다. 이번엔 socket 을 이용하여 서로 다른 두 컴퓨터끼리 네트워크 전송계층 프로토콜 중 하나인 TCP 를 통해 client 와 server 를 연결하고 client 에서 FTP 명령어 바꿔서 server 에 보내고 server 는 그 명령어를 수행하여 결과값을 다시 client 에 보내서 client 쪽에서 결과값이 출력되게 만들어볼 것이다.

Flow chart

1) cli 의 flow chart



2) srv 의 flow chart



Pseudo code

1) cli 의 pseudo code

////////////////////// Client-side Communication Setup //////////////////////

// Define maximum buffer sizes
int MAX_BUFF = 10000;
int RCV_BUFF = 10000;

// Function prototypes for command conversion and result processing
function int conv_cmd(char* buff, char* cmd_buff);
function void process_result(char* rcv_buff);

// Main function
int main(int argc, char** argv) {
    int sockfd;
    int n;
    int portno;

```
int str_len;
char buff[MAX_BUFF];
char cmd_buff[MAX_BUFF];
char rcv_buff[RCV_BUFF];
char message[100];
char* haddr;
struct sockaddr_in server_addr;

// Create a TCP socket and store the descriptor in sockfd
sockfd = socket(PF_INET, SOCK_STREAM, 0);
if (sockfd < 0) {
    Print "can't create socket";
    Exit -1;
} else {
    // Initialize rcv_buff and server_addr to zeroes
    bzero(rcv_buff, sizeof(rcv_buff));
    bzero(&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = inet_addr(argv[1]);
    server_addr.sin_port = htons(atoi(argv[2]));

    // Attempt to connect to the server using sockfd and server_addr
    if (connect(sockfd, &server_addr, sizeof(server_addr)) < 0) {
        Print "can't connect";
        Exit -1;
    } else {
        // Start an infinite loop to handle user commands
        Print prompt "> ";
        while (true) {
            bzero(buff, sizeof(buff));
            bzero(cmd_buff, sizeof(cmd_buff));
            Read user input into buff;

            // Convert Linux command in buff to FTP command in cmd_buff
            if (conv_cmd(buff, cmd_buff) < 0) {
                Print "conv_cmd() error!!";
                Exit 1;
            } else {
                // Calculate the length of cmd_buff
                n = strlen(cmd_buff);

                // Send the converted command to the server
                if (write(sockfd, cmd_buff, n) != n) {
                    Print "write() error!!";
                    Exit 1;
                } else {
                    // Receive a response from the server into rcv_buff
                    n = read(sockfd, rcv_buff, sizeof(rcv_buff));
                    if (n < 0) {
```

```
                    Print "read() error";
                    Exit 1;
                } else {
                    // Check for termination commands from the server
                    if (strcmp(rcv_buff, "QUIT") == 0) {
                        Print "Program quit!!";
                        Exit 1;
                    } else if (strcmp(rcv_buff, "Q") == 0) {
                        Exit 1;
                    } else {
                        // Process and display the server's response
                        process_result(rcv_buff);
                        Print prompt "> ";
                    }
                }
            }
        }
    }
    // Close the socket and end program
    close(sockfd);
    Return 0;
}

// Convert Linux commands to FTP commands
int conv_cmd(char* buff, char* cmd_buff) {
    char* ptr = strtok(buff, " ");
    if (strcmp(ptr, "ls") == 0) {
        strcpy(cmd_buff, "NLST");
        while ((ptr = strtok(NULL, " ")) != NULL) {
            strcat(cmd_buff, " ");
            strcat(cmd_buff, ptr);
        }
        return 1;
    } else if (strcmp(ptr, "quit") == 0) {
        strcpy(cmd_buff, "QUIT");
        return 1;
    }
    return -1;
}

// Display the results from the server
void process_result(char* rcv_buff) {
    for (int i = 0; i < RCV_BUFF; i += 100) {
        write(STDOUT_FILENO, rcv_buff + i, 100);
    }
}
```

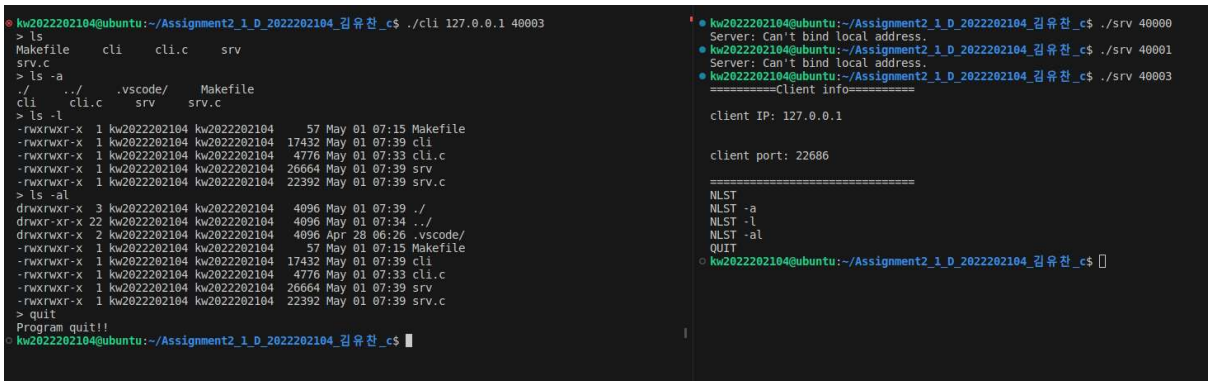
2) srv 의 pseudo code

```
////////////////////////////////////
// Server Application Pseudocode //
////////////////////////////////////

// Define Buffer Sizes and Constants
Define MAX_BUFF as 10000
Define SEND_BUFF as 10000
Define SEND_SIZE as 100

// Function Prototypes
Function int client_info(struct sockaddr_in* client)
Function int cmd_process(char* command, char* output)
Function void print_file_information(struct dirent* entry, struct stat* info, char* buffer, int buffer_index)

// Main Server Function
Function main(int argc, char** argv)
    Declare serverAddr, clientAddr as sockaddr_in
    Declare listenFD, connFD as integers
    Declare cliLen, readLength, portNo
    Declare buffers: buff, resultBuff, temp

    // Initialize server socket
    listenFD = socket(PF_INET, SOCK_STREAM, 0)
    if (listenFD is less than 0)
        Print "Server: Can't open stream socket."
        Return failure

    // Set server address and bind
    Initialize serverAddr
    serverAddr.sin_family = AF_INET
    serverAddr.sin_addr.s_addr = htonl(INADDR_ANY)
    serverAddr.sin_port = htons(portNo from argv[1])

    if (binding listenFD to serverAddr fails)
        Print "Server: Can't bind local address."
        Return failure

    // Start listening for connections
    Listen on listenFD with a queue limit of 5
    Loop Forever
        Accept a connection on listenFD and assign to connFD
        if (client_info(clientAddr) returns negative)
            Print "client_info() error!!"

        Loop
```

```
            Clear buffers: buff, resultBuff, temp
            readLength = read data from connFD into buff
            if (end of data or error)
                break the loop

            if (cmd_process(buff, resultBuff) returns negative)
                Send "Q" to client to signify termination
                break the loop

            Send processed command output back to client
            if (client sent "QUIT")
                Close connFD and exit loop

            Print received command for verification

    Close listenFD
    Return success

// Client Information Retrieval
Function client_info(struct sockaddr_in* client)
    if (client's family is not AF_INET)
        return negative

    Print "Client IP: ", inet_ntoa(client.sin_addr)
    Print "Client Port: ", ntohs(client.sin_port)
    return positive

// Command Processing
Function cmd_process(char* command, char* output)
    Initialize command arguments array argv with SEND_SIZE elements
    Parse command into argv using strtok

    if (first argument is "QUIT")
        if (too many arguments for QUIT)
            return negative
        Set output to "QUIT"
        return positive

    else if (first argument is "NLST")
        Handle options 'a' and 'l'
        Determine directory to list based on command arguments
        List directory contents using print_file_information
        return positive

    else
        return negative

// Print File Information
Function print_file_information(struct dirent* entry, struct stat* info, char* buffer, int buffer_index)
```

Retrieve owner and group information using getpwuid and getgrgid
Convert last modification time of info to readable format
Format file details into buffer starting at buffer_index
Append formatted string with file permissions, links, owner, group, size, and name
If entry is a directory, append '/' to its name
Copy formatted string to result buffer

결과화면



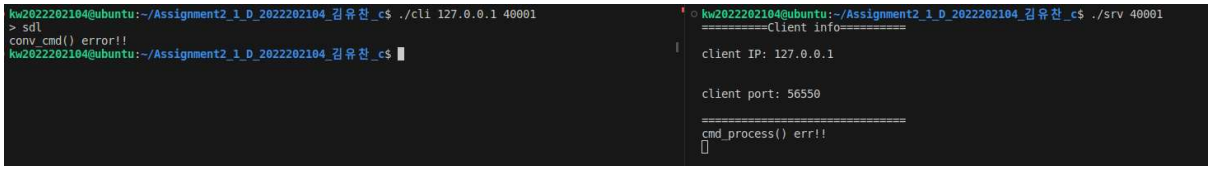
1. server 에서 쓸 수 없는 포트로 열면 에러가 뜬다. 만약 쓸 수 있는 포트를 열어두면 client 로부터 신호를 받을 준비를 할 수 있다.

2. Client 가 정상적으로 Server 에 연결이 되면 Client 의 IP 의 주소와 port 번호가 server 쪽에서 출력이 된다.

3. ls 명령어, ls -a, ls -l, ls -al 옵션을 client 에서 입력하면 client 쪽에서 FTP 명령어인 NLST, NLST -a, NLST -l, NLST -al 명령어로 바꿔서 server 에게 보내고 sever 쪽은 그 명령어를 출력하는 동시에 출력 결과를 client 에게 보내서 client 는 이 결과를 출력한다. 또한 클라이언트는 명령어를 받을 때가 언재인지 알려주기 위해 >를 터미널에 띄어 놓게 했다.

4. quit 를 명령어를 client 에서 입력하면 client 쪽에서 FTP 명령어인 QUIT 로 바꿔서 server 에서 보내고 sever 쪽은 이 명령어를 출력하고 똑같은 명령어를 client 에 다시 보내고 "Program quit!!"을 출력하면서 서버와 클라이언트가 모두 닫히게 만든다.

5. client 가 server 에 연결할 수 없으면 할 수 없다고 출력한다.



6. client 에서 잘못된 명령어를 보내면 server 는 계속 유지되지만 client 쪽에서는 접속이 끊어진다.

고찰

socket programming 으로 서로 다른 기기들끼리 정보를 주고받을 수 있다는 이론을 배웠던 적이 있는데 실제로 해볼 수 있는 과제라 지금까지 해본 것 중 제일 흥미로웠다. 하지만 그만큼 너무 생소했기 때문에 코딩을 하다가 error 가 많이 발생했다. 그래도 문제가 생길 때마다 각각에 상황에 대해 error 출력문을 만들었기 때문에 어디서 문제가 생긴지 알 수 있었다. 제일 많이 error 생긴 부분은 getopt 함수를 쓸 때 optind 라는 외부 전역함수를 초기화하지 않아 원하는 출력 값이 안 나온 부분과 데이터를 두 컴퓨터가 주고 받을 때 개행 문자까지 받거나 char 배열을 초기화하지 않아서 생기는 문제들이었다. 이런 문제들 때문에 segmentation fault (core dumped)와 원하지 않는 출력 값이 나왔다. 앞으로 socket programming 을 할 때는 이런 점을 잘 유의하여 코딩해야 한다.

Reference

시스템프로그래밍 실습 강의자료