시스템 프로그래밍 실습

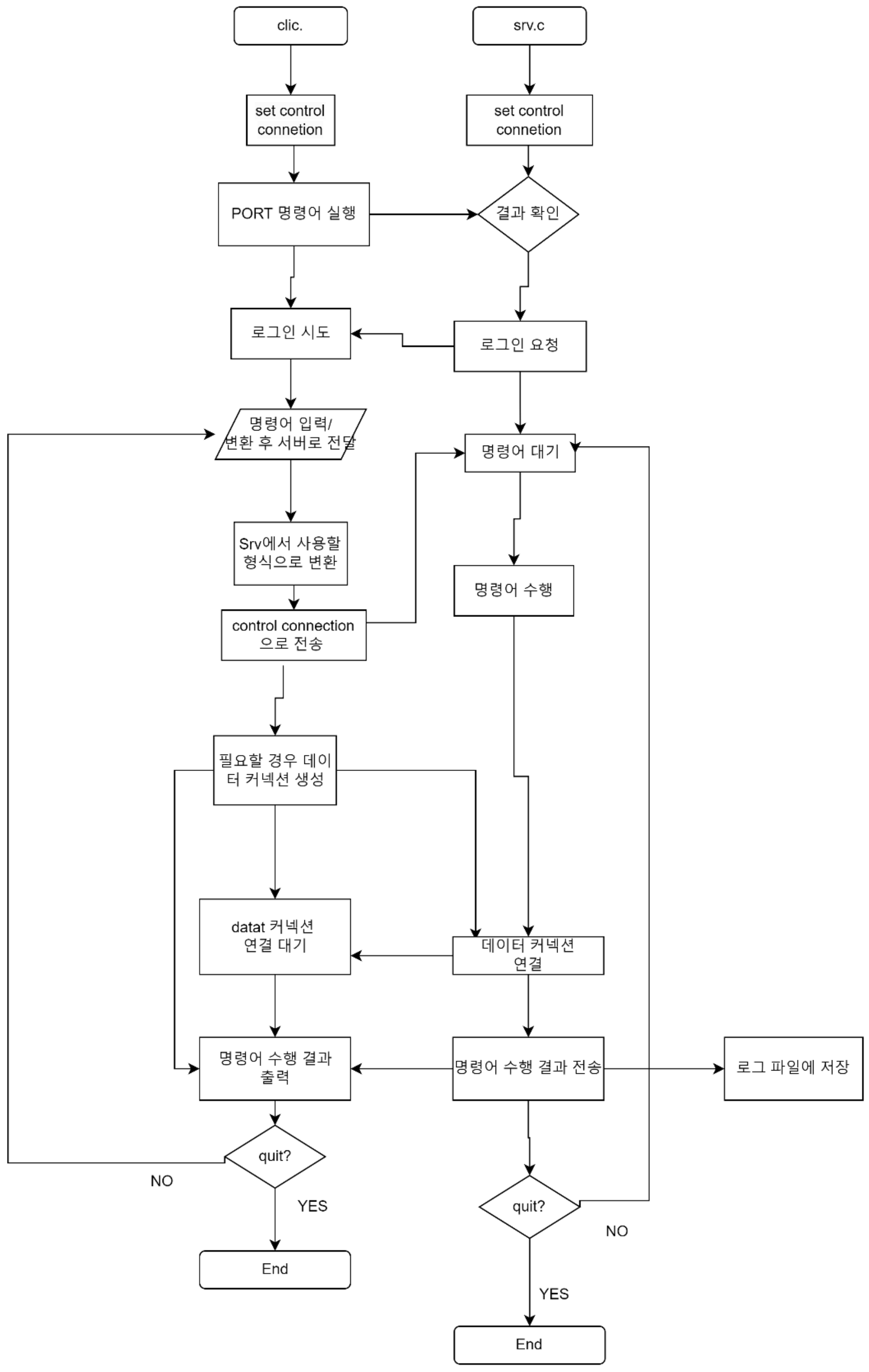
Assignment3 - 3

Class	:	D
Professor	:	최상호 교수님
Student ID	:	2020202037
Name	:	염정호

Introduction

본 프로젝트에서는 이전에 했던 FTP 과제를 통합하여 FTP 서버를 구현하는 것을 목표로 한다. 이전에 구현한 FTP 명령어에 추가로 USER PASS get put 명령어를 구현해야 하며 로그인과 서버/ 클라이언트 간에 파일 전송에 이용된다. 서버를 오픈하고 클라이언트가 들어오면 해당 클라이언트의 ip 주소를 확인하고 승인된 ip 의 경우 로그인 시도를 요청한다. 로그인 후 FTP 명령어를 입력하고 해당 결과를 logfile 을 통해 확인 가능하다. 사용자가 quit 또는 ctrl c 입력시 해당 프로세서의 연결을 종료한다.

Flow chart


clic.
srv.c
set control connetion
set control connetion
PORT 명령어 실행
결과 확인
로그인 시도
로그인 요청
명령어 입력/ 변환 후 서버로 전달
명령어 대기
Srv에서 사용할 형식으로 변환
명령어 수행
control connection 으로 전송
필요할 경우 데이터 커넥션 생성
datat 커넥션 연결 대기
데이터 커넥션 연결
명령어 수행 결과 출력
명령어 수행 결과 전송
로그 파일에 저장
quit?
NO
YES
End
quit?
NO
YES
End

Pseudo code

```
void main(int argc, char **argv)
{
    signal(SIGINT 발생시 함수); // Set up signal handler for SIGINT (Ctrl+C)
    int n, k, p_pid; // Socket file descriptor and other variables
    int data_cli_fd, data_srv_fd;
    struct sockaddr_in control_srv, data_cli, data_srv;
    char buff[256];
    char *hostport;
    char user_input[256];
    char *command[] = {"ls", "dir", "pwd", "cd", "mkdir", "delete", "rmdir", "rename",
"quit", "bin", "ascii", "get", "put", NULL};          // List of commands
    char *convert[] = {"NLST", "LIST", "PWD", "CWD", "MKD", "DELE", "RMD",
"RNFR&RNTO", "QUIT", "TYPE I", "TYPE A", "RETR", "STOR", NULL}; // Corresponding
commands to be sent to the server
    char*divide_input_string;    // Pointer for dividing input string
    char *to_srvNULL;            // String to be sent to the server
    char recv_from_srv[4000];               // Buffer to store server response
    size_t len_to_srv;
    int index_num = 0, cmd_num; // Index counter
    char *index[20] = {NULL};    // Array to store separated token
    srand(time(0)); // Seed the random number generator
    // Create a control socket
    control_sockfd = (socket 컨트롤 소켓 생성);

    // Set up the server address struct
    memset(서버 주소 설정));
    control_srv.sin_family = AF_INET;
    control_srv.sin_addr.s_addr = inet_addr(argv[1]);
    control_srv.sin_port = htons(atoi(argv[2]));

    connect(서버에 연결 요청);
```

```
log_in(control_sockfd); // Log in to the server

//////////////////////////Client connection completed/////////////

while (1)
{
    memset(사용자로부터 입력 받을 데이터 초기화); // Reset buffer
    divide_input_string = "₩0";
    fflush(stdout);
    write(STDOUT_FILENO, "> ", 2);                          // Print prompt
    ssize_t bytes_read = read(사용자로부터 입력받음)); // Read user input
    user_input[strlen(user_input) - 1] = '₩0';  //널문자 삽입
    trim_whitespace(입력의 앞 뒤 공백을 삭제);

    divide_input_string = strtok(공백을 기준으로 구분)
    int index_num = 0;

    while (문자열을 나눠 인덱스 배열에 저장)
    {
        index[index_num] = divide_input_string;   // Store the cut string
        divide_input_string = strtok(NULL, " "); // Find the next token
        index_num++;                              // Move to the next array index
    }

    if (strcmp(index[0], "bin") == 0) // canage mode
        type = 0; // 타입 변환
    else if (strcmp(index[0], "ascii") == 0)
        type = 1;

    cmd_num = 0; // Initialize command index

    while (명령어 탐색)
    {
        if (strcmp(command[cmd_num], index[0]) == 0) // Check if command exists
```

```
            break;
        cmd_num++;
    }

    if (to_srv)
    {
        데이터에 보내기 전 서버 데이터 초기화
    }

    if (command[cmd_num] == NULL)
    {
        to_srv = (char *)malloc(strlen(index[0]) + 1);
        strcpy(to_srv, index[0]);
    }

    else if (인자가 하나일 경우) // Command without arguments
    {
        print_data("₩n");                                   // Print newline
        to_srv = (char *)malloc(strlen(convert[cmd_num]) + 1); // Allocate memory for
converted command
        strcpy(to_srv, convert[cmd_num]);              // Copy converted command
    }

    else if (인자가 2개일 경우) // Command with one argument
    {
        to_srv = (char *)malloc(strlen(convert[cmd_num]) + strlen(index[1]) + 2); //
Allocate memory for command and argument
        strcat(to_srv, convert[cmd_num]);             // Copy command
        strcat(to_srv, " ");                                  // Add space
        strcat(to_srv, index[1]);                                 // Copy argument
    }
    else if (인자가 2개 이상인 경우) // Command with multiple arguments
    {
        to_srv = (char *)malloc(strlen(convert[cmd_num])); // Allocate memory for
command
```

```
        len_to_srv = strlen(to_srv);

        for (인덱스 결합)
            len_to_srv = len_to_srv + strlen(index[k]); // Calculate total length

        len_to_srv = len_to_srv + index_num - 1;

        to_srv = (char *)malloc(len_to_srv); // Allocate memory for command and
arguments
        memset(to_srv, 0, sizeof(to_srv));
        strcat(to_srv, convert[cmd_num]); // Copy command

        for (인자 개수 확인) // Append arguments
        {
            strcat(to_srv, " ");
            strcat(to_srv, index[k]);
        }
    }

    to_srv[strlen(to_srv)] = '₩0';                    // Null-terminate string
    write(control_sockfd, to_srv, strlen(to_srv) + 1); // Send the command to the
server

    if (데이터 소켓이 필요한 명령인 경우 열기)
    {
        read(control_sockfd, buff, sizeof(buff));
        memset(buff, 0, sizeof(buff));

        int random_port = 10000 + rand() % 20001; // Gener

        data_cli_fd = socket(데이터 소켓 생성);
        memset 데이터 클라이언트 초기화
        data_cli.sin_family = AF_INET;
        data_cli.sin_addr.s_addr = inet_addr(argv[1]);
        data_cli.sin_port = htons(random_port);
        hostport = convert_addr_to_str(data_cli.sin_addr.s_addr, data_cli.sin_port);

        write(데이터 전송)
        read(응답 대기; // code 200
        print_data(recv_from_srv);
        memset(recv_from_srv, 0, 4000);

        // Bind the control socket to the control address
        bind(data_cli_fd, (struct sockaddr *)&data_cli, sizeof(data_cli));

        // Listen for incoming connections
        listen(data_cli_fd, 5);
    }

    int len = sizeof(data_srv);
    write(control_sockfd, " ", 2);
    if (데이터 소켓을 열었을 경우)
        data_srv_fd = accept(소켓 연결;

    if (디렉토리를 탐색하는 명령인 경우)
    {
        read(control_sockfd, recv_from_srv, 4000); // code150
        print_data(recv_from_srv);
        memset(recv_from_srv, 0, 4000);
    }
    else if (명령어 get 인 경우)
    {
        int code_i;

        read(control_sockfd, recv_from_srv, 4000); // code 200
        code_i = atoi(recv_from_srv);
        print_data(recv_from_srv);
        memset(recv_from_srv, 0, 4000);
        char buffer[1024] = {0};
```

```
        int n;
        FILE *file;

        int bytes_received;
        int total_bytes_received = 0;
        if (정상적인 실행)
        {
            if (type == 0)
                file = fopen(index[1], "wb");
            else
                file = fopen(index[1], "w");
            // Receive and save file content from the server
            while ((bytes_received = recv(data_srv_fd, buffer, sizeof(buffer), 0)) > 0)
            {
                fwrite(buffer, 1, bytes_received, file);
                total_bytes_received += bytes_received;
            }
            char msg[256];
            sprintf(입력받은 데이터 길이 출력);
            print_data(msg);
            fclose(file);
        }

    }
    else if ((명령어가 put 인 경우) == 0))
    {
        read(control_sockfd, recv_from_srv, sizeof(recv_from_srv));
        print_data(recv_from_srv);                        // Concatenate output
        memset(recv_from_srv, 0, sizeof(recv_from_srv)); // code 150

        char buffer[1024];
        FILE *file;
        if (type == 0)
            file = fopen(index[1], "rb");
        else
```

```
}

            file = fopen(index[1], "r");

        if (file != NULL)
        {
            write(control_sockfd, "exist", strlen("exist"));
            int file_size = 0;
            while (!feof(file))
            {
             클라이언트로 부터 데이터를 읽어옴
            }

            fclose(file);
            close(data_srv_fd);

            read(control_sockfd, recv_from_srv, sizeof(recv_from_srv));

            print_data(recv_from_srv);
            char msg[256];
            sprintf(입력받은 파일 크기 출력); // Convert the total
            print_data(msg);
            memset(recv_from_srv, 0, sizeof(recv_from_srv));
            continue;
        }
        else
        {
            write(control_sockfd, "fail", strlen("fail"));
            close(data_srv_fd);
            read(컨트롤 데이터로 데이터를 받음);
            print_data(recv_from_srv);
            memset(recv_from_srv, 0, sizeof(recv_from_srv));
            continue;
        }
    }
// Use write to output the total received bytes
```

```
        if (strcmp(index[0], "ls") == 0 || strcmp(index[0], "dir") == 0 || strcmp(index[0],
"get") == 0 || strcmp(index[0], "put") == 0)
        // Data Connection Output
        {
            read(데이터 커넥션을 읽어와서 출력);
            print_data(recv_from_srv); // Concatenate output
            write(control_sockfd, " ", 2);
        }
        int i = strlen(recv_from_srv);
        memset(recv_from_srv, 0, sizeof(recv_from_srv));
        read(마지막 데이터 입력 받음)); // Read the final message from the server
        print_data(recv_from_srv);

        if (파일 탐색 명령어인 경우) //
        {
            char str_size[200];
            sprintf(파일 길이 출력);
            if (i > 2)
                print_data(str_size);
            memset(str_size, 0, 200);
        }

        memset(recv_from_srv, 0, sizeof(recv_from_srv));
        if (데이터 소켓을 열었을 경우 소켓 종료) == 0)
        // if data fd open close it
        {
            close(data_cli_fd);
            close(data_srv_fd);
        }

        if (strcmp(index[0], "quit") == 0)
            exit(1);
    }
    // Generate a random port for the data connection
    return;
```

```
}

int main(int argc, char **argv)
{
    LogBook = fopen(로그 파일 생성);
    if (LogBook == NULL)
    {
        perror("로그 파일 열기 실패");
        return 1;
    }

    // 현재 시간 가져오기 및 초기화
    time(&rawtime);
    timeinfo = localtime(&rawtime);

    // 시간 형식 지정 및 로그 작성
    strftime(time_buf, sizeof(time_buf), "%a %b %d %H:%M:%S %Y", timeinfo);
    sprintf(log_buf, "%s %s", time_buf, "Server is started₩n");

    // 로그 파일에 쓰기
    fprintf(LogBook, "%s", log_buf);
    fflush(LogBook);
    // Create a control socket
    control_fd = socket(PF_INET, SOCK_STREAM, 0);

    // Set up the control address struct
    memset(컨트롤 소켓 생성));
    control_addr.sin_family = AF_INET;
    control_addr.sin_addr.s_addr = htonl(INADDR_ANY); // Use any available address
    control_addr.sin_port = htons(atoi(argv[1]));       // Use the provided port number

    // Bind the control socket to the control address
    bind(control_fd, (struct sockaddr *)&control_addr, sizeof(control_addr));
    listen(연결 요청대기);
```

```
for (;;)
{
    int len = sizeof(control_cli);
    ctrl_cli_fd = accept(연결 요청);

    if ((pid = fork()) < 0)
    {
    }
    else if (pid == 0)
    {
        client_ip = inet_ntoa(control_cli.sin_addr);
        sprintf(포트 번호 저장)
        // Open the access file to check client's IP
        fp_checkIP = fopen(로그인 데이터 확인을 위해 열기)

        if (fp_checkIP == NULL)
        {
            printf("**file ₩"access.txt₩" doesn't exist**");
            return 1;
        }
        while (모든 ip 번호를 확인)
        {
            while ((check_ip[i] != '₩0') && (client_ip[j] != '₩0'))
            {
            if ((check_ip[i] == '₩0') && (client_ip[j] == '₩0'))
            {
                i = 100;
                break;
            }
        }

        // If client IP is matched, allow connection
        if (i == 100)
```

```
{
    write_code(실패시 출력)
}
else
{
    // If client IP is not matched, reject connection
    write_code(성공 메시지 출력)
    continue;
}

fclose(fp_checkIP);

if (3 번 실패시 종료)
{ // if 3 times fail (ok : 1, fail : 0)
    printf("** Fail to log-in **₩n");
    break;
}

while (1)
{
    strcpy(concatenate_output, " ");
    memset(버퍼 초기화)
    memset(read_buf, 0, 256);
    n = read(ctrl_cli_fd, read_buf, 256); // Read data from the client/
    read_buf[n] = '₩0';
    print_data("input : ");
    if (strcmp(read_buf, "CWD ..") == 0)
        print_data("CDUP");
    else
        print_data(read_buf);
    print_data("₩n");
    write_log(read_buf);

    index = strtok(읽어온 데이터를 공백으로 구분))// Tokenize the input
buffer
```

```
        num_of_sindex = 0;
        while (인덱스 개수 저장 및 분류)
        {
            s_index[num_of_sindex] = index; // Store the tokenized string
            num_of_sindex++;
            index = strtok(NULL, " ");       // Find the next token
        }

        if (데이터 커넥션이 필요한 명령일 경우 예외처리) == 0)
        {
            write(ctrl_cli_fd, " ", 2); // Write a space to the control connection
            memset(temp, 0, 30);           // Clear the temp buffer
            read(ctrl_cli_fd, temp, sizeof(temp));
            temp[strlen(temp)] = '₩0';
            host_ip = convert_str_to_addr(temp, (unsigned int *)&port_num);
            write_code(소켓 오픈 메시지 보냄)// Send response code 200 to client
            data_fd = socket(데이터 소켓 생성);
            memset 데이터 소켓 주소 설정
            data_addr.sin_family = AF_INET;
            data_addr.sin_addr.s_addr = inet_addr(host_ip);
            data_addr.sin_port = htons(port_num);

            read(준비 완료를 알리는 메시지 대기);
            connect(데이터 커넥션 연결));
        }
명령어에 따라 동작 수행
        if (strcmp(s_index[0], "NLST") == 0) // If the command is NLST
        {
            strcpy(concatenate_output, " "); // Initialize concatenate_output
            code = 150;                          // Set response code to 150
            DIR *dp = NULL;                       // Variable for directory traversal
            struct dirent *dirp = 0;
            char *dir = " ";                   //
char *files[100];    // Array to store the list of files
int file_count = 0; // Initialize file_count to store the number of files in
the list
int option_i = 0;    // Store options as integers: 0 for none, 1 for a, 2
for l, 3 for al

if (인덱스가 1 개)
{                  // If only the command exists
    dir = "."; // Set dir to the current directory
}
else if (인덱스 2 개) // If two arguments are received
{
    if (옵션인 경우) // If the second argument is an option
    {
        dir = ".";
        option_i = check_option(s_index[1]); // Classify the option
        if (option_i > 99)
            code = 501;
    }
    else
        dir = s_index[1]; // Set dir to the given address
}
else if (인자가 3 개)
{
    dir = s_index[2];
    option_i = check_option(옵션 확인); // Store the option
    if (옵션 예외)
        code = 501; // Set response code to 501
}
else
{
    code = 501; // Set response code to 501
}
write_code(code, ctrl_cli_fd, " "); // Send response code to client
```

```
// Directory open
if (opendir(dir) == NULL || code != 150)
{
    code = 550; // Set response code to 550
    if (errno == EACCES)
        strcpy(code_buff, 에러 종류에 따라 오류 코드 변환
    else
        strcpy(code_buff, 에러 종류에 따라 오류 코드 변환
}
else
{
    code = 226;
    dp = opendir(dir);

    while ((읽어온 파일 목록 저장)
    {
        char file_path[300];
        struct stat file_stat;
        sprintf(file_path, "%s/%s", dir, dirp->d_name);
        files[file_count] = dirp->d_name; // Store the file name
        stat(file_path, &file_stat);        // Get information about the file
        if (S_ISDIR(file_stat.st_mode))
        {
            strcat(files[file_count], "/"); // Append '/' if it's a directory
        }
        file_count++; // Move to the next file
    }
    closedir(dp); // Close the directory

    bubbleSort(files, file_count); // Sort the list of file names

    switch (옵션에 따라 수행_i) //
    {
    case 0:
        print_non_option(files, file_count); // If there's no option
```

```
                        break;
                    case 1:
                        print_a(files, file_count); // If the option is 'a'
                        break;
                    case 2:
                        print_l(files, file_count); // If the option is 'l'
                        break;
                    case 3:
                        print_al(files, file_count); // If the option is 'al'
                        break;
                    default:
                        strcpy(concatenate_output, "invalid option");
                        break;
                    }

                    n = strlen(concatenate_output);
                    concatenate_output[n] = '₩0';    // Ensure null-termination of the
string
                }
            }
            else if (list 명령어== 0) // Command to list files in a directory
            {
                code = 150;                              // Set response code to 150
                strcpy(concatenate_output, " "); // Initialize concatenate_output
                DIR *dp = NULL;                         // Variable for directory traversal
                struct dirent *dirp = 0;            // Structure to hold information about
files within a directory
                char *dir = " ";                        // Initialize dir variable to store the
directory address to be searched

                char *files[100];    // Array to store the list of files
                int file_count = 0; // Initialize file_count to store the number of files in
the list
                int option_i = 0;    // Store options as integers: 0 for none, 1 for a, 2
for l, 3 for al
```

```
if (num_of_sindex == 1)
{                    // If only the command exists
    dir = "."; // Set dir to the current directory
}
else if (num_of_sindex == 2) // If two arguments are received
{
    if (s_index[1][0] == '-') // If the second argument is an option
    {
        code = 501; // Set response code to 501
    }
    else
        dir = s_index[1]; // If it's not an option, dir is the given address
}
else
    code = 501;                              // Set response code to 501
write_code(code, ctrl_cli_fd, " "); // Send response code to client

// Directory open
if (opendir(dir) == NULL || code != 150)
{
    code = 550; // Set response code to 550
    if (errno == EACCES)
        strcpy(code_buff, "NLSTA"); // Set error code_buff to "NLSTA"
    else
        strcpy(code_buff, "NLSTD"); // Set error code_buff to "NLSTD"
}
else
{
    code = 226;           // Set response code to 226
    dp = opendir(dir); // Traverse the directory

    while ((dirp = readdir(dp)) != NULL) // Get information about files
in the directory
    {
            char file_path[300];
            struct stat file_stat;
            sprintf(file_path, "%s/%s", dir, dirp->d_name);

            files[file_count] = dirp->d_name; // Store the file name
            stat(file_path, &file_stat);        // Get information about the file
            if (S_ISDIR(file_stat.st_mode))
            {
                strcat(files[file_count], "/"); // Append '/' if it's a directory
            }
            file_count++; // Move to the next file
        }
        closedir(dp); // Close the directory

        bubbleSort(정렬); // Sort the list of file names

        print_al(files, file_count); // If the option is 'al'
    }

    n = strlen(concatenate_output);
    concatenate_output[n] = '₩0';
}
else if (현재 디렉토리 출력== 0)
{
    strcpy(concatenate_output, " ");
    if (num_of_sindex > 1)
    {
        code = 501; // Set response code to 501
    }
    else // If no additional arguments are provided
    {
        getcwd(code_buff, 256); // Get the current working directory
        code = 257;                 // Set response code to 257
    }
}
```

```
        else if (현재 디렉토리 변경") == 0)
        {
            strcpy(concatenate_output, " "); // Initialize concatenate_output
            strcpy(code_buff, "CWD");          // Set code_buff to "CWD"
                strcpy(code_buff, "CWD");                // Set code_buff to "CWD"
                code = 250;                              // Set response code to 250
        else if (strcmp(s_index[0], "MKD") == 0) // Command to make directories
        {
            memset(code_buff, 0, 1024);        // Clear code_buff
            strcpy(concatenate_output, " "); // Initialize concatenate_output
            if (잘못된 수의 인덱스)
                code = 501;                       // Set response code to 501
            else if (s_index[1][0] == '-')
                code = 501;                       // Set response code to 501
            else
            {
                j = 1;                            // Start checking arguments from index 1
                code = 1000;                  // Set response code to 1000
                while (j < num_of_sindex) // Loop through all received arguments
                {
                    if (mkdir(디렉토리 생성)
                    {
                        strcat(code_buff, "550 ");
                        strcat(에러 출력)
                        strcat(code_buff, ": can’t create directory₩n");
                    }
                    else
                        strcat(code_buff, "250 MKD command performed
successfully₩n");
                    j++; // Move to the nxt argument
                }
            }
        }
        else if (strcmp(s_index[0], "DELE") == 0) // Command to delete files
        {
            strcpy(concatenate_output, " "); // Initialize concatenate_output
            if (s_index[1] == NULL)            // If no file names are provided as
arguments
                code = 501;                    // Set response code to 501
            else if (s_index[1][0] == '-')    // If an option is provided instead of a
file name
                code = 501;                    // Set response code to 501
            else
            {
                memset(code_buff, 0, 512); // Clear code_buff
                j = 1;                        // Start checking arguments from index 1
                code = 1000;                  // Set response code to 1000
                while (j < num_of_sindex)   // Loop through all received arguments
                {
                    if (unlink(해당 파일 삭제) // Attempt to delete the file
                    {
                          실패시 출력
                        strcat(code_buff, "550 ");
                        strcat(code_buff, s_index[j]);
                        strcat(code_buff, ": Can't find such file or directory₩n"); //
                    }
                    else
                        strcat(code_buff, 성공 메시지 출력");
                    j++; // Move to the next argument
                }
            }
        }
        else if (strcmp(s_index[0], "RMD") == 0)
        {
            strcpy(concatenate_output, " "); // Initialize concatenate_output
                    if (rmdir(디렉토리 삭제) // Attempt to remove the directory
                    {
                          에러 발생시 출력
                        strcat(code_buff, "550 ");
                        strcat(code_buff, s_index[j]);
```

```
                                strcat(code_buff, ": can’t remove directory₩n");
                            }
                            else
                                strcat(code_buff, "250 RMD command performed
successfully₩n");

                            j++; // Move to the next argument
                        }
                    }
                }
                else if (strcmp(s_index[0], "RNFR&RNTO") == 0)
                {
                    strcpy(concatenate_output, " ");            // Initialize concatenate_output
                    memset(code_buff, 0, sizeof(code_buff)); // Clear code_buff
                    code = 1000;                              // Set response code to 1000

                    if (인덱스 개수 확인)
                        code = 501;                                           // Set response
code to 501
                    else if (s_index[1][0] == '-' || s_index[2][0] == '-') // If an option is
provided as an argument
                        code = 501;                                           // Set response
code to 501
                    else
                    {

                        if (access(존재하는 파일인지 확인)
                        {
                            strcat(code_buff, "550 ");                                     //
Append an error message if the source file or directory doesn't exist
                            strcat(code_buff, s_index[1]);                                 //
Append the name of the file or directory causing the error
                            strcat(code_buff, ": Can’t find such file or directory₩n"); // Add
a newline character
```

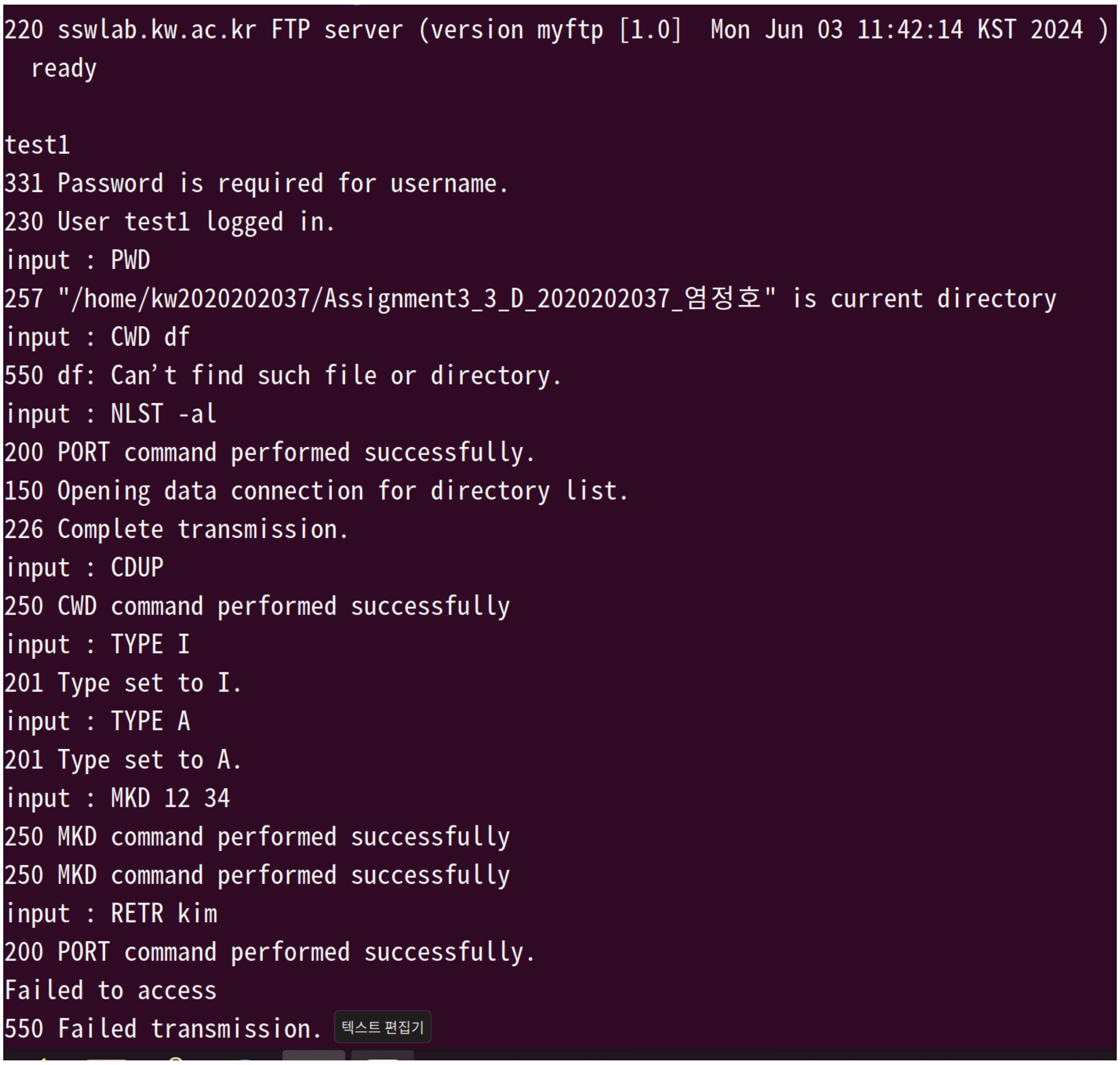
```
                            strcat(code_buff, "550 ");                              //
Append an error message if the target file or directory doesn't exist
                            strcat(code_buff, s_index[2]);                          //
Append the name of the file or directory causing the error
                            strcat(code_buff, ": can’t be renamed₩n");                  //
Add a newline character
                        }
                        else
                        {
                            strcat(code_buff, "350: File exists, ready to rename₩n"); //
Append the command name "RNFR"
                            // Check if the target file or directory already exists
                            if (생성하려는 파일 이름이 존재하지 않아야 함)
                            {
                                strcat(code_buff, "550 ");                  // Append an error
message if the target file or directory already exists
                                strcat(code_buff, s_index[2]);              // Append the
name of the file or directory causing the error
                                strcat(code_buff, ": can’t be renamed₩n"); // Add a newline
character
                            }
                            else
                            {
                                rename(s_index[1], s_index[2]);                     // Rename
the file or directory
                                strcat(code_buff, "250: RNTO command succeeds₩n"); //
Append success message
                                strcat(concatenate_output, "₩n");                   // Add a
newline character
                            }
                        }
                    }
                }
                else if (strcmp(s_index[0], "TYPE") == 0) // Command to set transfer mode
                {
```

```
        strcpy(concatenate_output, " "); // Initialize concatenate_output
        code = 201;                          // Set response code to 201

        // Check if the correct number of arguments is provided
        if (num_of_sindex != 2)
            code = 501; // Set response code to 501

        // Set transfer mode based on the argument
        if (입력값에 따라 타입 변경)
            type = 0; // Binary mode
        else
            type = 1; // ASCII mode
    }
    else if (strcmp(s_index[0], "RETR") == 0)
    {
        strcpy(concatenate_output, " ");          // Initialize concatenate_output
        memset(code_buff, 0, sizeof(code_buff)); // Clear code_buff
        strcpy(code_buff, "file");                // Set file type
        code = 150;                               // Set response code to 150

        char buffer[1024];
        FILE *file;

        // Open the file in the appropriate mode
        if (타입에 따라 파일 생성)
            file = fopen(s_index[1], "rb"); // Binary mode
        else
            file = fopen(s_index[1], "r"); // ASCII mode

        if (파일이 존재하는지 확인) // Check if the file exists
        {
            code = 550;                               // Set response code to
550
            memset(code_buff, 0, sizeof(code_buff)); // Clear code_buff
            strcpy(code_buff, "NLSTA");               // Set error message
        }

        write_code(code, ctrl_cli_fd, code_buff); // Write response code to
client

        if (code == 150) // If file retrieval starts
        {
            while (파일 끝까지 전송)
            {
                int bytes_read = fread(buffer, 1, sizeof(buffer), file);
                send(data_fd, buffer, bytes_read, 0); // Send file data to client
            }

            fclose(file);    // Close the file
            close(data_fd); // Close the data connection
        }

        // Set appropriate response code based on file retrieval success
        if (code == 150)
            code = 226; // File transfer successful
        else
        {
            code = 550;                    // File transfer failed
            strcpy(code_buff, "NLSTD"); // Set error message
        }
    }

    else if (strcmp(s_index[0], "STOR") == 0) // Command to store file on
server
    {
        write_code(150, ctrl_cli_fd, "file"); // Send acknowledgment to client
        strcpy(concatenate_output, " ");        // Initialize concatenate_output
        char buffer[1024] = {0};                // Buffer for receiving data
        int bytes_received;
```

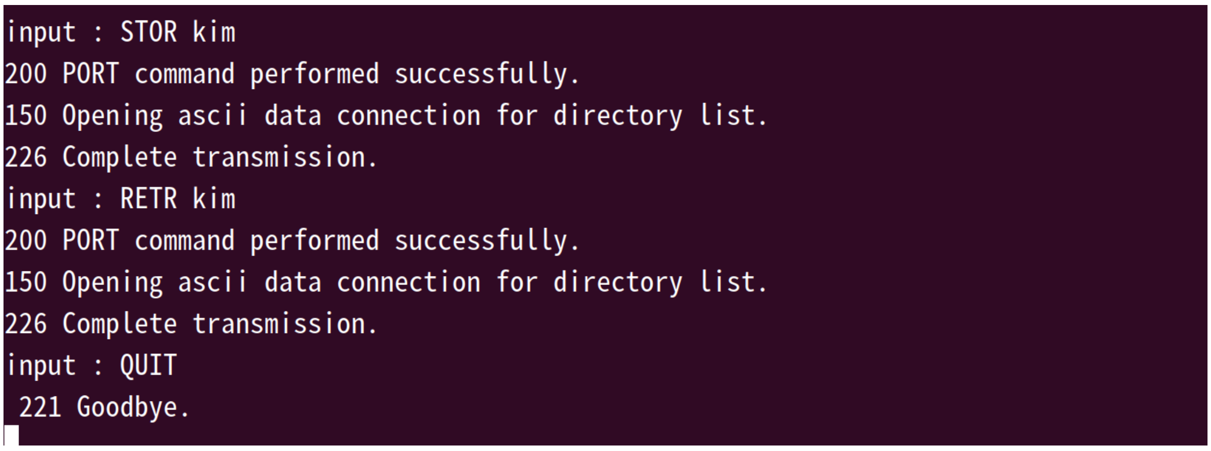
```
                    FILE *file;
                    memset(trash_buf, 0, sizeof(trash_buf));            // Clear trash_buf
                    read(ctrl_cli_fd, trash_buf, sizeof(trash_buf)); // Read acknowledgment
from client

                    if (strcmp(trash_buf, "exist") == 0) // Check if the file exists
                    {
                        // Open the file in the appropriate mode
                        if (타입에 따라 파일 열기)
                            file = fopen(s_index[1], "wb"); // Binary mode
                        else
                            file = fopen(s_index[1], "w"); // ASCII mode

                        while ((bytes_received = recv(data_fd, buffer, sizeof(buffer), 0)) > 0)
// Receive data from client
                        {
                            fwrite(buffer, 1, bytes_received, file); // Write received data to
file
                            fflush(file);                            // Flush the buffer to write
data to the file
                        }

                        write_code(226, ctrl_cli_fd, " "); // Send acknowledgment to client
                        continue;
                    }
                    else
                    {
                        write_code(550, ctrl_cli_fd, " "); // Send error code to client
                        continue;
                    }
                }
                else if (strcmp(s_index[0], "QUIT") == 0) // Handling the QUIT command
                {
                    strcpy(concatenate_output, " ");                            //
Initialize concatenate_output
```

```
                    write(data_fd, concatenate_output, sizeof(concatenate_output)); // Send
acknowledgment to client
                    print_data(concatenate_output);                          // Print
acknowledgment message

                    read(ctrl_cli_fd, trash_buf, 2);    // Read acknowledgment from client
                    write_code(221, ctrl_cli_fd, " "); // Send acknowledgment to client
                    close(data_fd);                      // Close the data connection
                    close(ctrl_cli_fd);                 // Close the control connection
                    exit(1);                              // Exit the program
                }
                else
                {
                    strcpy(concatenate_output, " "); // Initialize concatenate_output
                    code = 500;                          // Set response code to 500
                }

                if (데이터 커넥션을 생성한 경우 해당 커넥션 종료") == 0)
                {
                    n = write(data_fd, concatenate_output, strlen(concatenate_output));
                }

                read(ctrl_cli_fd, trash_buf, 2); // Read acknowledgment from client

                s_index[0] = "df";                          // Set default value for s_index[0]
                write_code(code, ctrl_cli_fd, code_buff); // Write response code to client
                memset(code_buff, 0, sizeof(code_buff));   // Clear code_buff
                close(data_fd);                             // Close the data connection
            }
        }
    }
    close(control_fd);
    return 0;
}
```

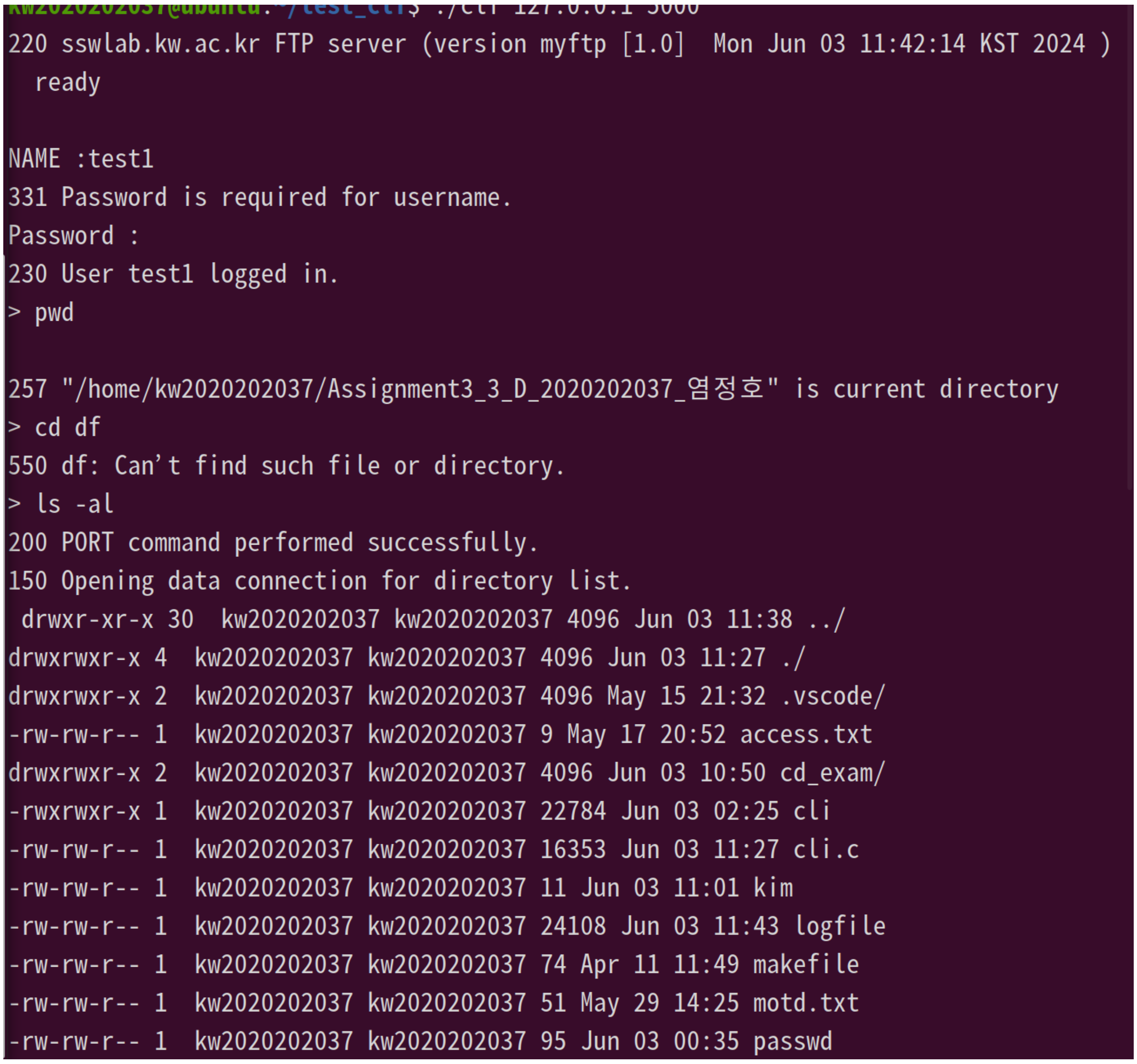
결과화면



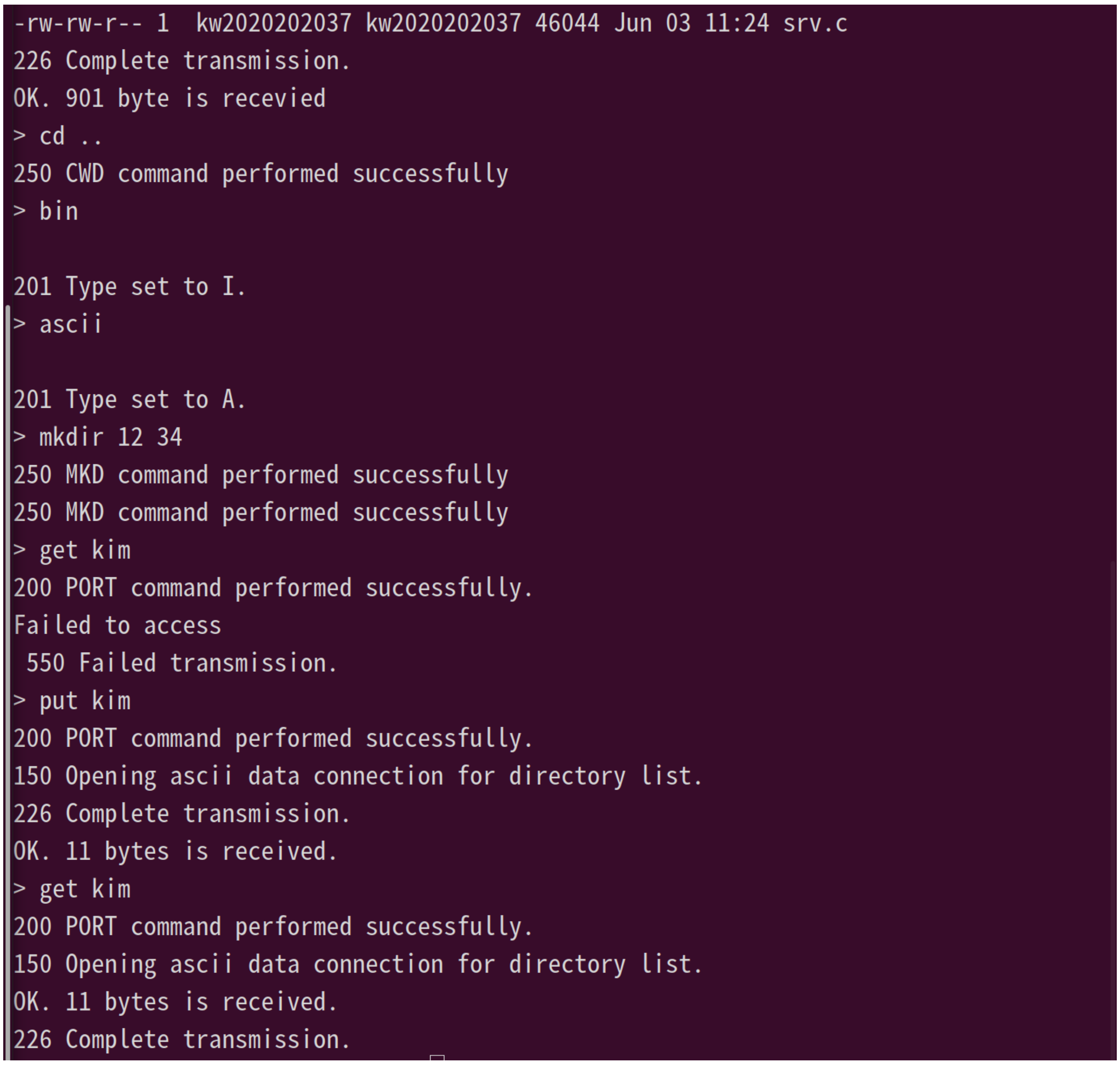
```
220 sswlab.kw.ac.kr FTP server (version myftp [1.0]  Mon Jun 03 11:42:14 KST 2024 )
  ready

test1
331 Password is required for username.
230 User test1 logged in.
input : PWD
257 "/home/kw2020202037/Assignment3_3_D_2020202037_염정호" is current directory
input : CWD df
550 df: Can't find such file or directory.
input : NLST -al
200 PORT command performed successfully.
150 Opening data connection for directory list.
226 Complete transmission.
input : CDUP
250 CWD command performed successfully
input : TYPE I
201 Type set to I.
input : TYPE A
201 Type set to A.
input : MKD 12 34
250 MKD command performed successfully
250 MKD command performed successfully
input : RETR kim
200 PORT command performed successfully.
Failed to access
550 Failed transmission.
```



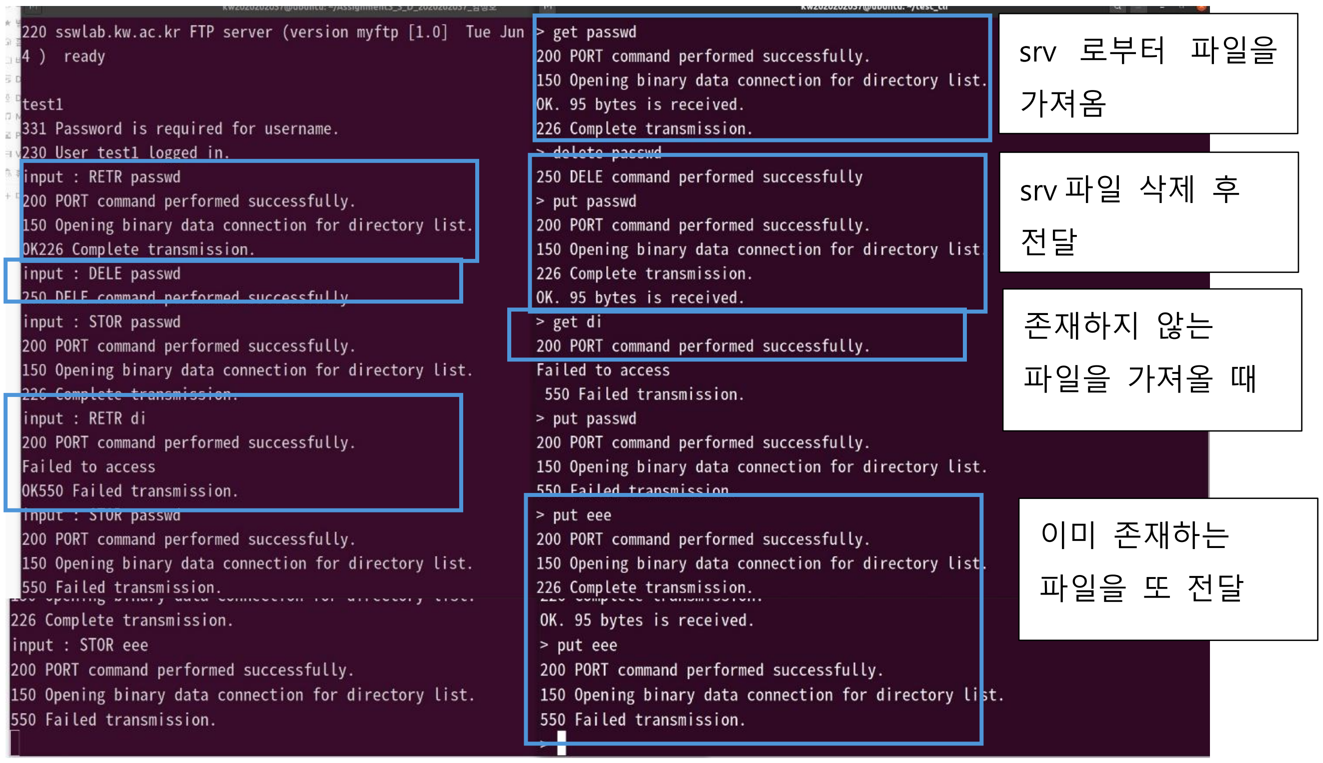
```
input : STOR kim
200 PORT command performed successfully.
150 Opening ascii data connection for directory list.
226 Complete transmission.
input : RETR kim
200 PORT command performed successfully.
150 Opening ascii data connection for directory list.
226 Complete transmission.
input : QUIT
 221 Goodbye.
```



```
kw2020202037@ubuntu:~/test_cli$ ./cli 127.0.0.1 5000
220 sswlab.kw.ac.kr FTP server (version myftp [1.0]  Mon Jun 03 11:42:14 KST 2024 )
  ready

NAME :test1
331 Password is required for username.
Password :
230 User test1 logged in.
> pwd

257 "/home/kw2020202037/Assignment3_3_D_2020202037_염정호" is current directory
> cd df
550 df: Can't find such file or directory.
> ls -al
200 PORT command performed successfully.
150 Opening data connection for directory list.
 drwxr-xr-x 30  kw2020202037 kw2020202037 4096 Jun 03 11:38 ../
drwxrwxr-x 4  kw2020202037 kw2020202037 4096 Jun 03 11:27 ./
drwxrwxr-x 2  kw2020202037 kw2020202037 4096 May 15 21:32 .vscode/
-rw-rw-r-- 1  kw2020202037 kw2020202037 9 May 17 20:52 access.txt
drwxrwxr-x 2  kw2020202037 kw2020202037 4096 Jun 03 10:50 cd_exam/
-rwxrwxr-x 1  kw2020202037 kw2020202037 22784 Jun 03 02:25 cli
-rw-rw-r-- 1  kw2020202037 kw2020202037 16353 Jun 03 11:27 cli.c
-rw-rw-r-- 1  kw2020202037 kw2020202037 11 Jun 03 11:01 kim
-rw-rw-r-- 1  kw2020202037 kw2020202037 24108 Jun 03 11:43 logfile
-rw-rw-r-- 1  kw2020202037 kw2020202037 74 Apr 11 11:49 makefile
-rw-rw-r-- 1  kw2020202037 kw2020202037 51 May 29 14:25 motd.txt
-rw-rw-r-- 1  kw2020202037 kw2020202037 95 Jun 03 00:35 passwd
```



```
-rw-rw-r-- 1  kw2020202037 kw2020202037 46044 Jun 03 11:24 srv.c
226 Complete transmission.
OK. 901 byte is recevied
> cd ..
250 CWD command performed successfully
> bin

201 Type set to I.
> ascii

201 Type set to A.
> mkdir 12 34
250 MKD command performed successfully
250 MKD command performed successfully
> get kim
200 PORT command performed successfully.
Failed to access
 550 Failed transmission.
> put kim
200 PORT command performed successfully.
150 Opening ascii data connection for directory list.
226 Complete transmission.
OK. 11 bytes is received.
> get kim
200 PORT command performed successfully.
150 Opening ascii data connection for directory list.
OK. 11 bytes is received.
226 Complete transmission.
```



```
220 sswlab.kw.ac.kr FTP server (version myftp [1.0]  Tue Jun
4 )  ready

test1
331 Password is required for username.
230 User test1 logged in.
input : RETR passwd
200 PORT command performed successfully.
150 Opening binary data connection for directory list.
OK226 Complete transmission.
input : DELE passwd
250 DELE command performed successfully
input : STOR passwd
200 PORT command performed successfully.
150 Opening binary data connection for directory list.
226 Complete transmission.
input : RETR di
200 PORT command performed successfully.
Failed to access
OK550 Failed transmission.
input : STOR passwd
200 PORT command performed successfully.
150 Opening binary data connection for directory list.
550 Failed transmission.
150 Opening binary data connection for directory list.
226 Complete transmission.
input : STOR eee
200 PORT command performed successfully.
150 Opening binary data connection for directory list.
550 Failed transmission.
```

```
> get passwd
200 PORT command performed successfully.
150 Opening binary data connection for directory list.
OK. 95 bytes is received.
226 Complete transmission.
> delete passwd
250 DELE command performed successfully
> put passwd
200 PORT command performed successfully.
150 Opening binary data connection for directory list.
226 Complete transmission.
OK. 95 bytes is received.
> get di
200 PORT command performed successfully.
Failed to access
 550 Failed transmission.
> put passwd
200 PORT command performed successfully.
150 Opening binary data connection for directory list.
550 Failed transmission.
> put eee
200 PORT command performed successfully.
150 Opening binary data connection for directory list.
226 Complete transmission.
 OK. 95 bytes is received.
 > put eee
 200 PORT command performed successfully.
 150 Opening binary data connection for directory list.
 550 Failed transmission.
 >
```

고찰

이번 프로젝트의 진행 과정은 거의 대부분 이전에 구현한 코드를 결합하는 형식으로 진행할 예정이어서 순탄하게 진행 될것으로 생각했으나 그렇지 못했다. 여러 코드를 가져오는 과정에서 해당 코드의 실행순서나 구조, 파일 입출력 문제로 인해 코드들이 서로 두 코드 모두 동작하지 못하는 과정이 발생했으며, 이전에 구현한 명령어들이 새로 짠 코드에서는 동작하지 않는 모습도 볼 수 있었다. 이를 위해 새로운 명령어를 가져올 때 마다 결과값을 찍어 직접 확인을 했어야 했으며, 각 명령어들을 현재 프로젝트에 맞게 수정 및 보완이 필요했다.