시스템 프로그래밍 실습

[Assignment2-1]

Class : **[D]**

Professor : **[최상호교수님]**

Student ID : **[2020202060]**

Name : **[홍왕기]**

Introduction

이 프로젝트는 소켓 프로그래밍을 활용하여 서버와 클라이언트 간에 명령어 상호작용을 구현하는 것이다. 클라이언트는 사용자로부터 명령어를 입력받고, 이를 FTP 명령어로 변환하여 서버에 전송한다. 서버는 받은 FTP 명령어를 해석하고 해당 명령어를 실행하여 결과를 클라이언트에게 반환한다. 이 과정에서 소켓 프로그래밍의 핵심 요소인 bind, listen, accept, connect, send, recv 함수 등이 활용된다.

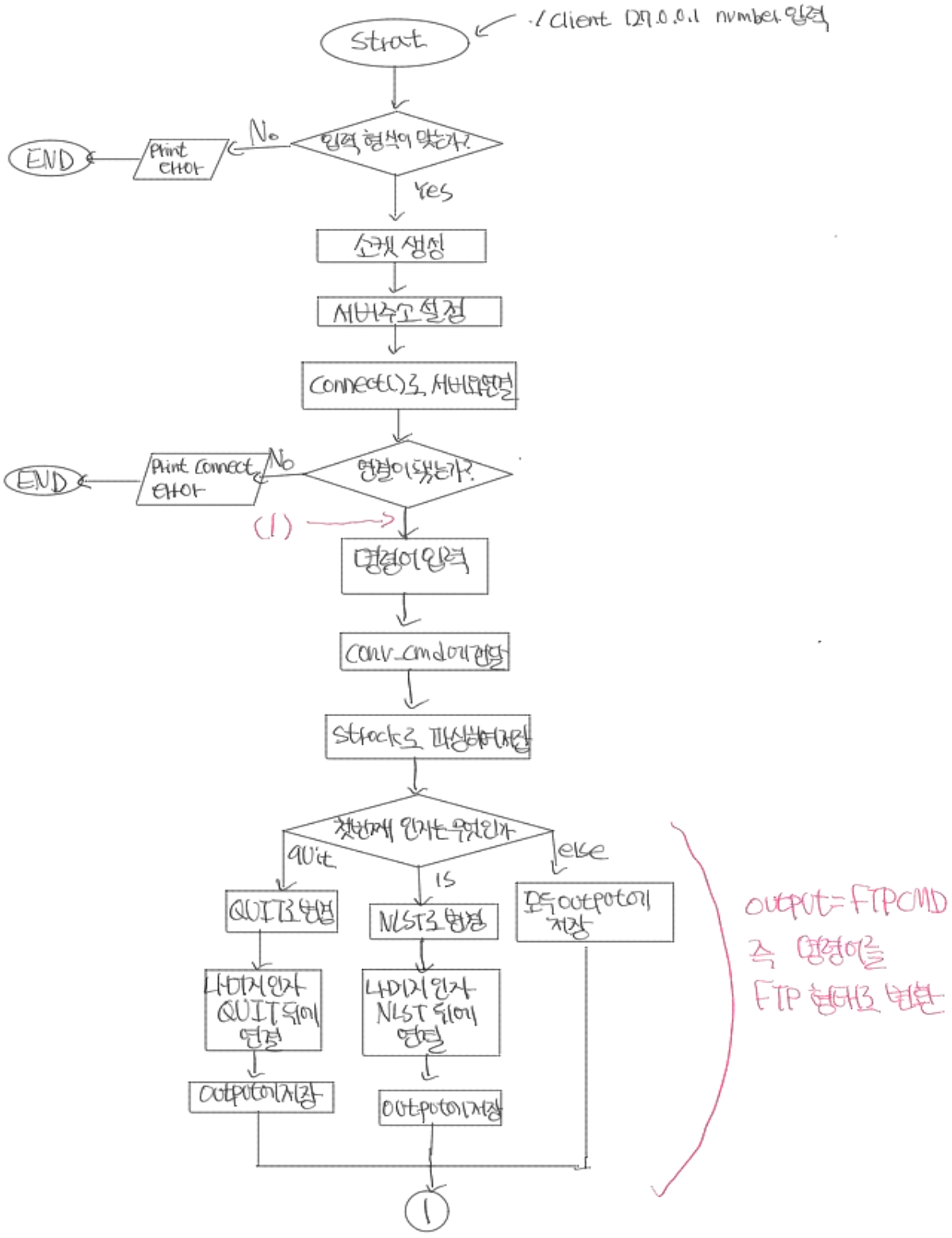
클라이언트는 사용자가 입력한 명령어를 받아 FTP 명령어로 변환하는 역할을 담당한다. 이를 통해 사용자는 명령어를 직관적으로 입력할 수 있으며, 클라이언트가 이를 FTP 프로토콜에 맞게 변환하여 서버로 전송한다. 이렇게 전송된 FTP 명령어는 서버에서 수신되고, 서버는 해당 명령어를 해석하여 실행한다.

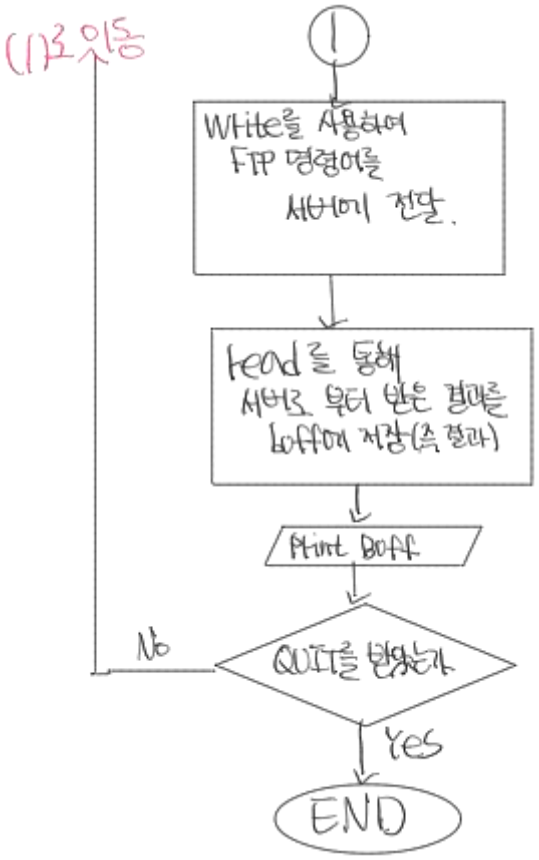
서버는 클라이언트의 연결 요청을 수신하고, 이를 받아들이는 역할을 한다. 서버는 클라이언트로부터 전송된 FTP 명령어를 받아들이고, 해당 명령어를 실행하여 그 결과를 다시 클라이언트에게 반환한다. 이 과정에서 서버는 소켓 프로그래밍의 핵심 메서드들을 사용하여 클라이언트와의 통신을 하는 것이다.

이 프로젝트는 소켓 프로그래밍을 활용하여 FTP 서버를 일부를 구현하는 것이며, 클라이언트와 서버 간의 명령어 통신을 효과적으로 수행할 수 있도록 한다.

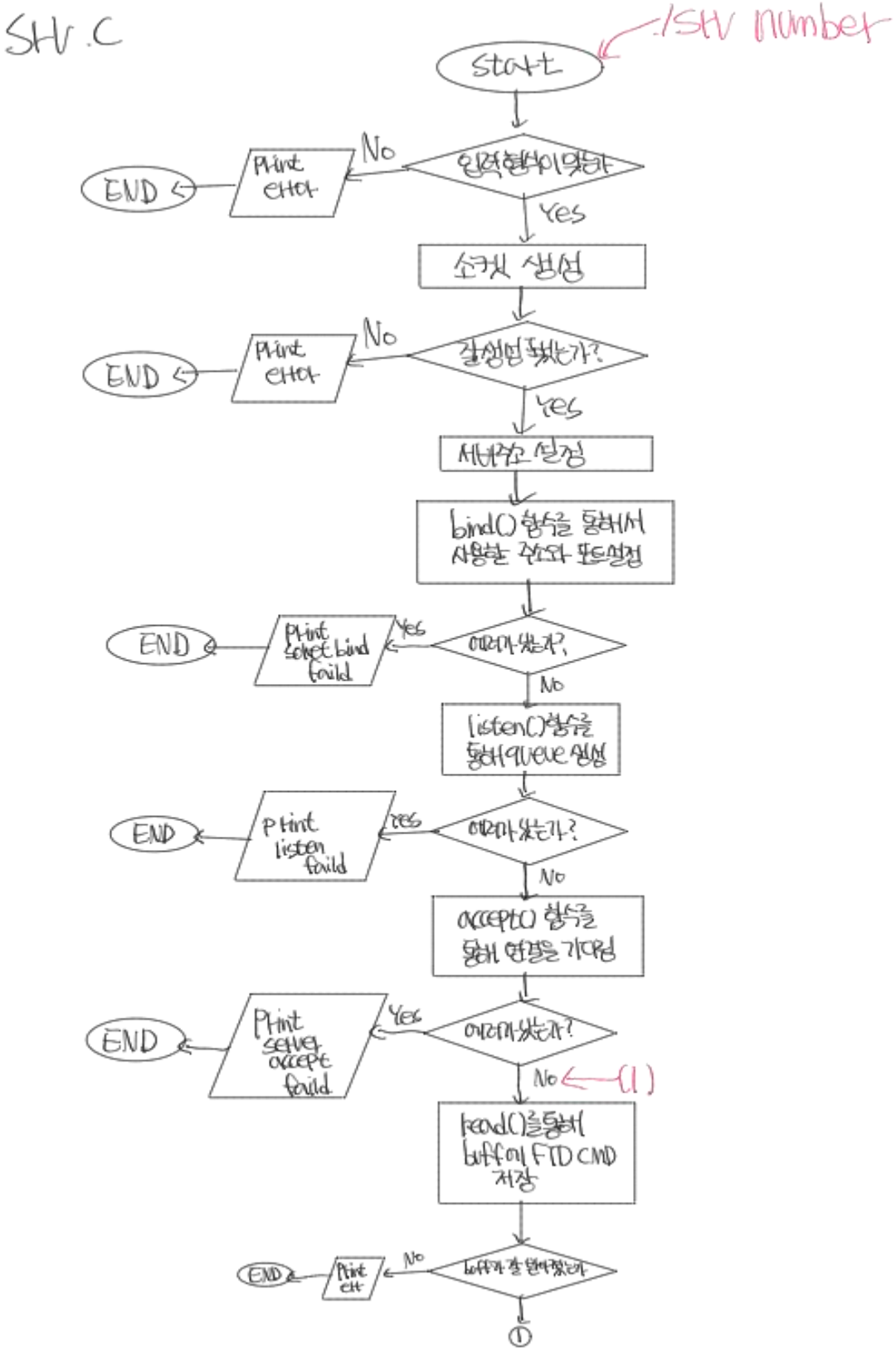
Flow chart

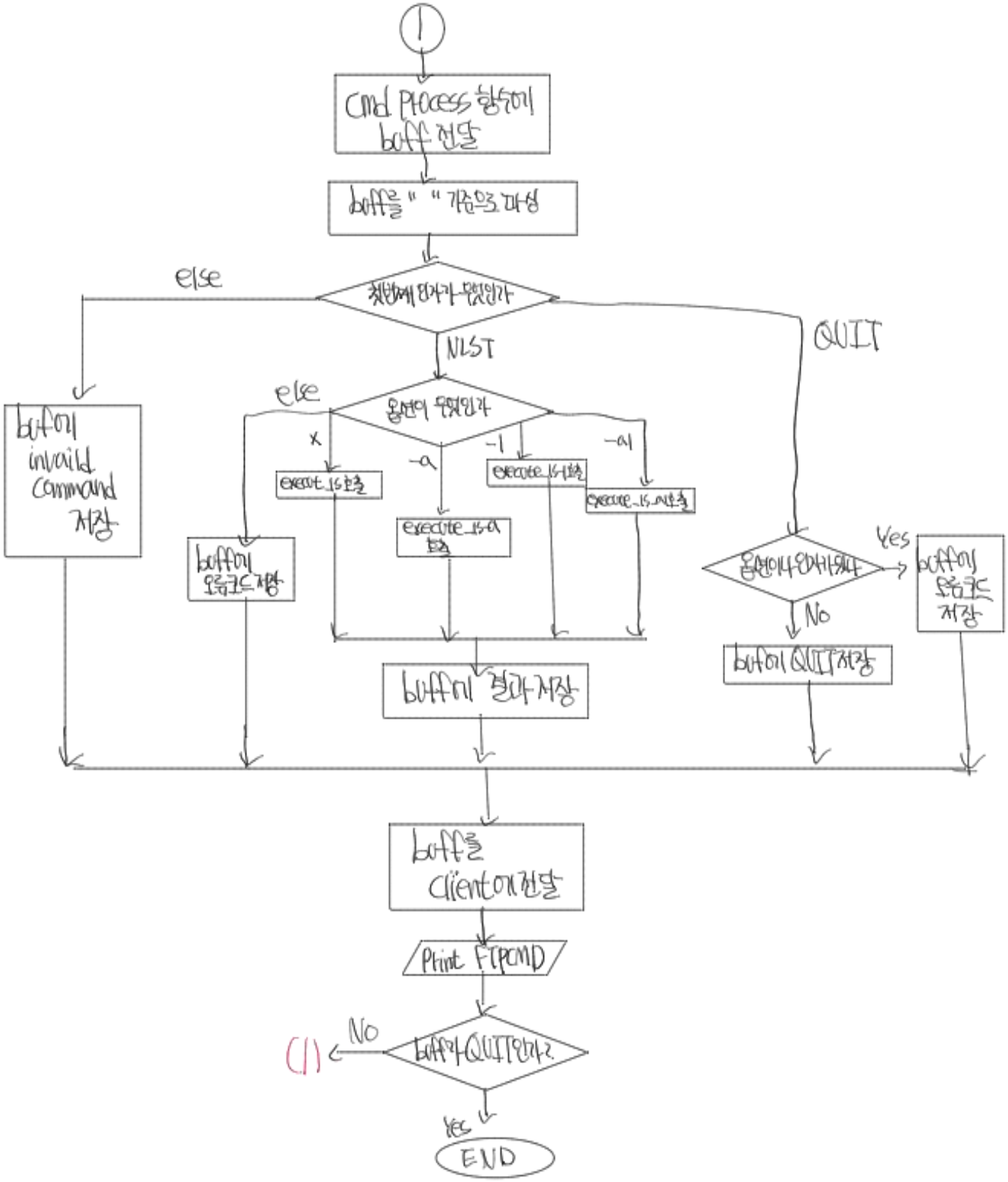
Cli.c



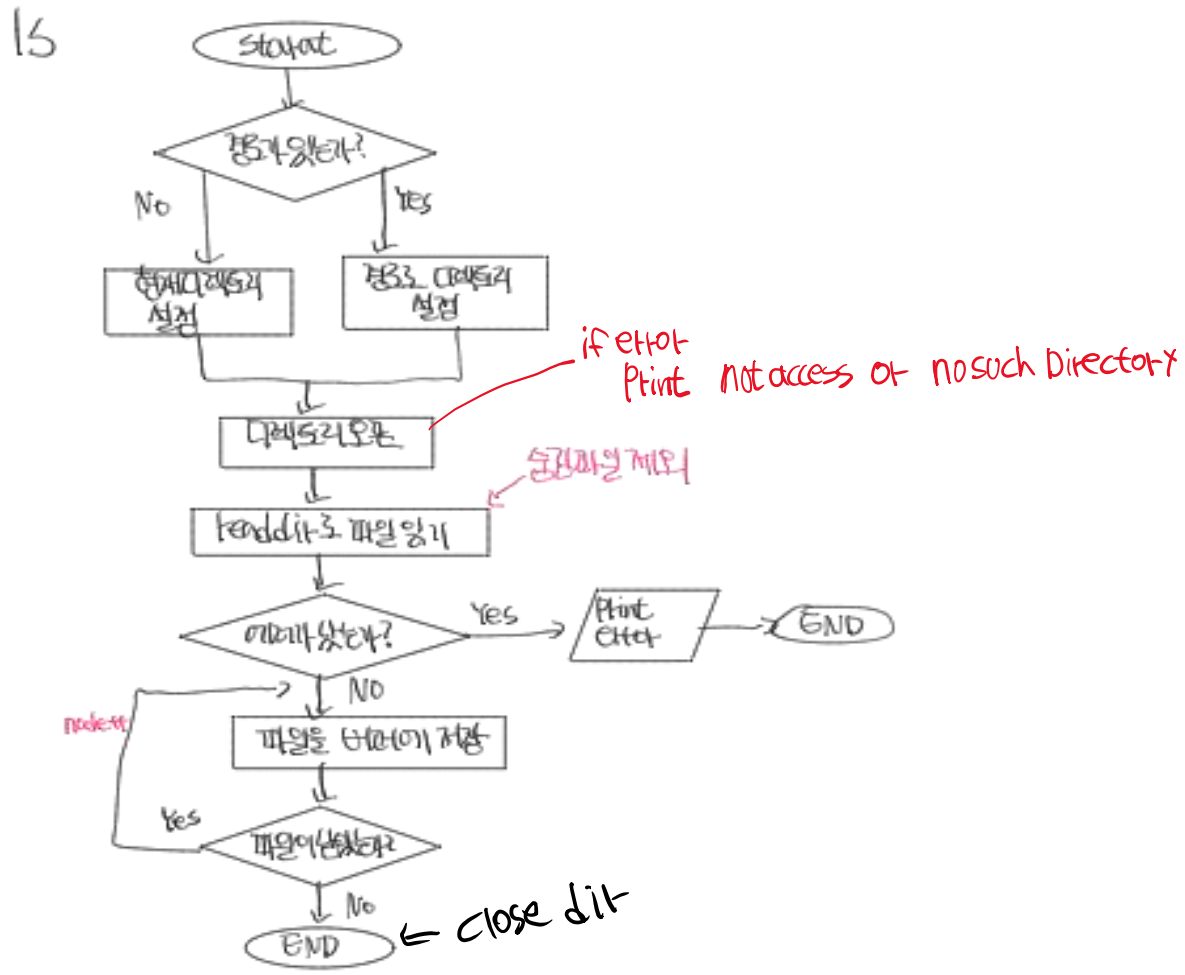

(1)로 이동
1
Write를 사용하여
FTP 명령어를
서버에 전달.
read를 통해
서버로 부터 받은 결과를
buffer에 저장(즉 출력)
Print buff
No
QUIT을 입력했는가
Yes
END

Srv.c

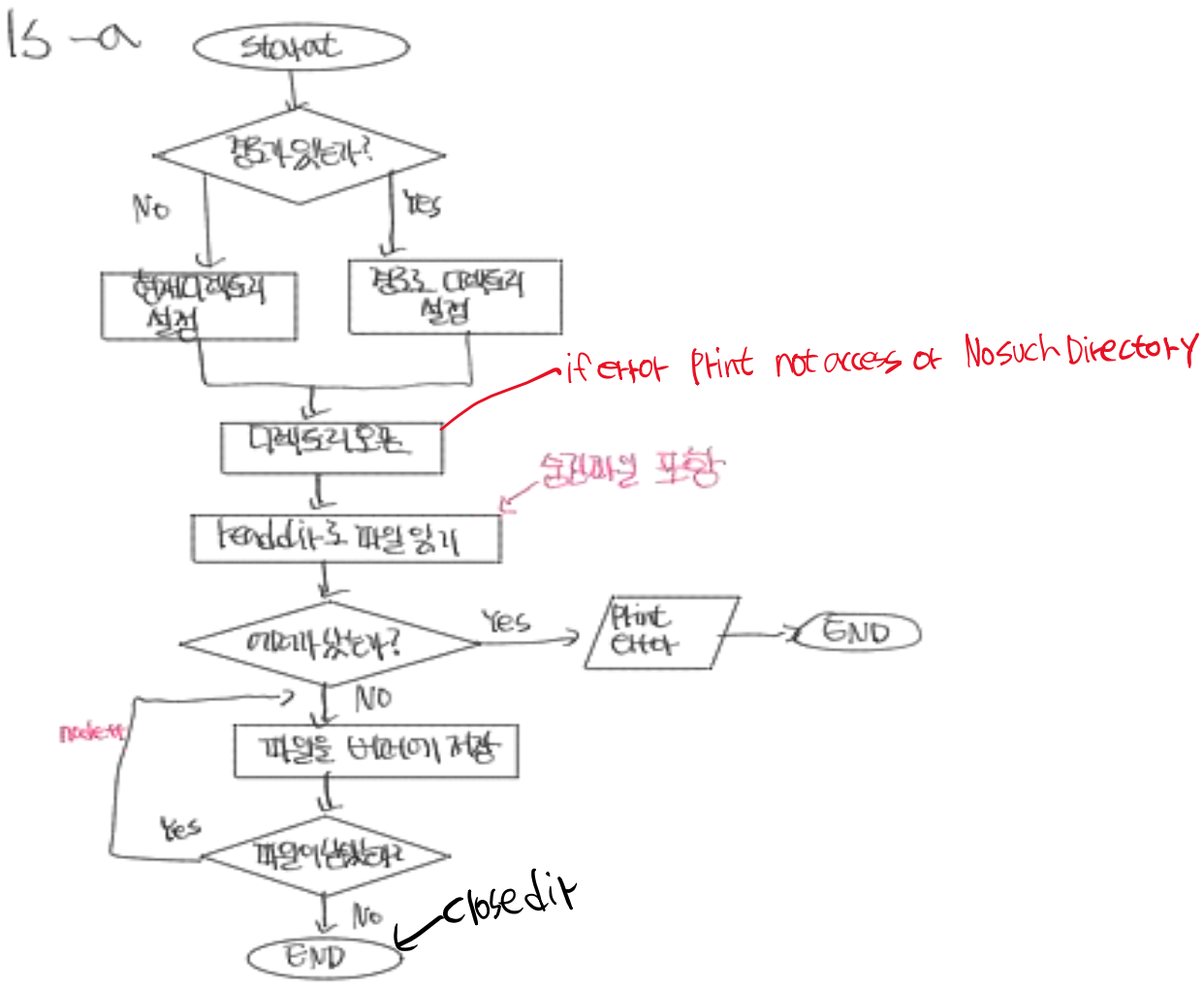

SRV.C
./SRV number
START
인자 형식이 맞는가
No
Print error
END
Yes
소켓 생성
생성됐는가?
No
Print error
END
Yes
서버정보 설정
bind() 함수를 통해서
사용할 주소와 포트설정
에러가 났는가?
Yes
Print socket bind faild
END
No
listen() 함수를
통해 queue 생성
에러가 났는가?
Yes
Print listen faild
END
No
accept() 함수를
통해 연결을 기다림
에러가 났는가?
Yes
Print server accept faild
END
No
(1)
read() 함수를 통해
buff에 FTP CMD
저장
buff에 값이 들어왔는가
No
Print err
END
①


Cmd Process 함수의 buff 전달
buff를 " " 기준으로 파싱
처음에 입력한 명령어
else
NLST
QUIT
buff의 invaild Command 저장
옵션이 있는가
else
x
-a
-l
-al
execute_ls호출
execute_ls-a 호출
execute_ls-l호출
execute_ls-al호출
buff의 옵션코드 저장
옵션에 a가 있는가
Yes
No
buff의 옵션코드 저장
buff에 QUIT저장
buff에 결과 저장
buff를 client에 전달
Print FTPCMD
buff에 QUIT인가?
No
Yes
END

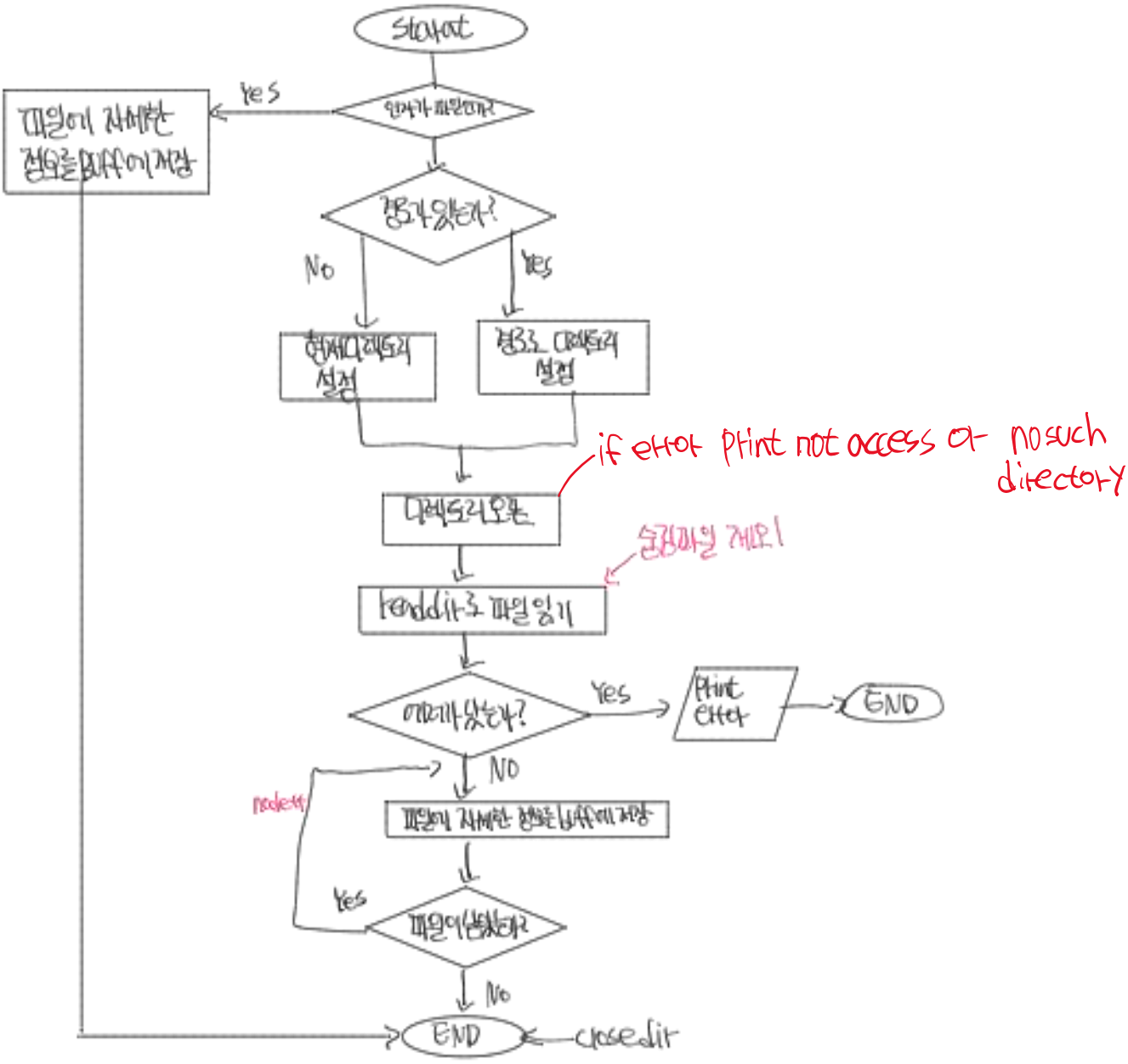
Execute_ls


ls
start
옵션이 있는가?
No
Yes
현재디렉토리 설정
옵션과 디렉토리 설정
if error
Print not access or no such Directory
디렉토리오픈
옵션파일 체크
readdir로 파일읽기
에러가 났는가?
Yes
Print Error
END
NO
node++
파일을 버퍼에 저장
Yes
파일이 남았는가?
No
END
close dir

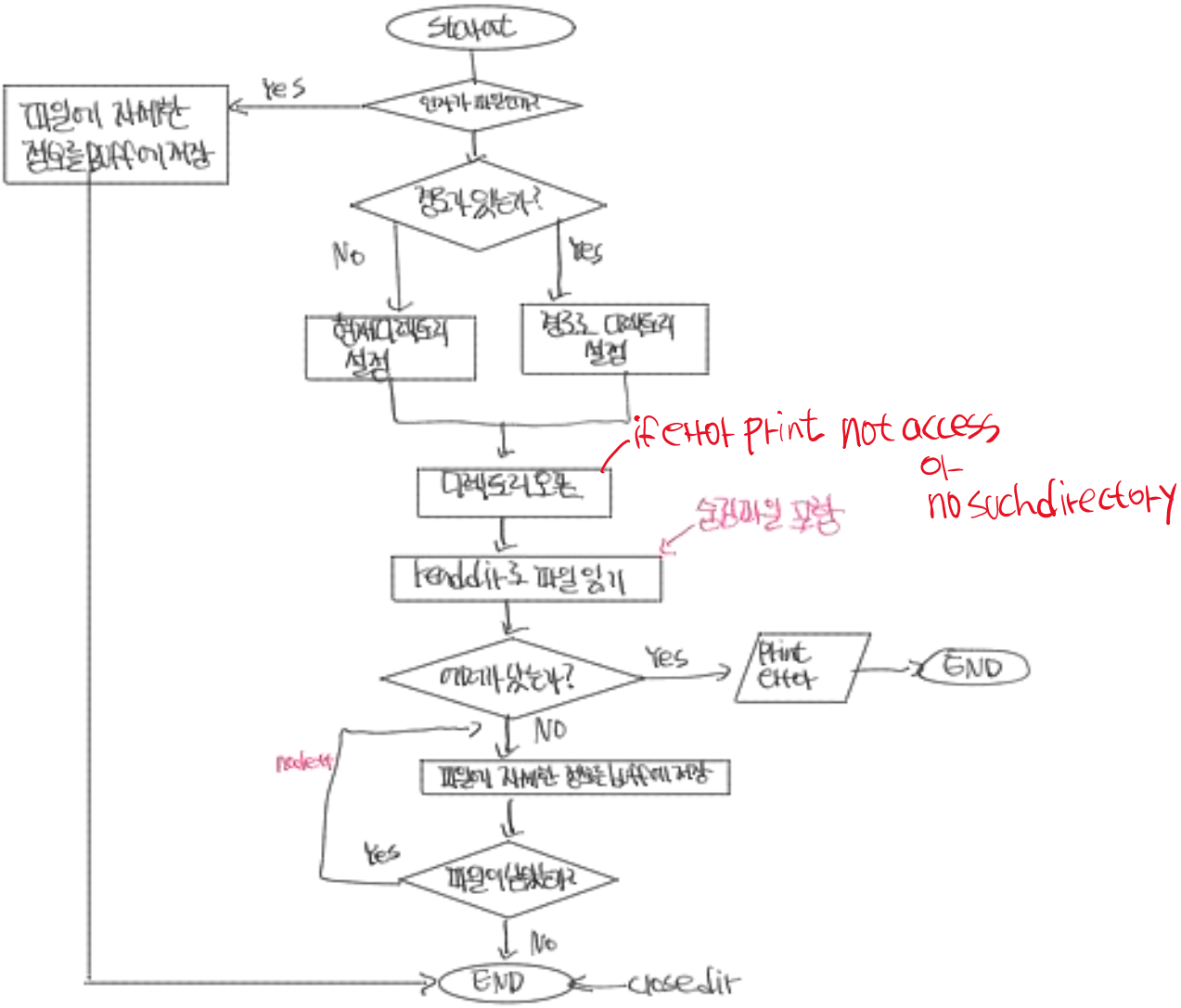
Execute_ls-a


ls -a
start
옵션있는가?
No
Yes
현재디렉토리 설정
옵션의 디렉토리 설정
if error print not access or Nosuch Directory
디렉토리오픈
숨김파일 포함
readdir로 파일읽기
에러났는가?
Yes
Print Error
END
NO
mode=0
파일을 버퍼에 저장
Yes
파일이 남았는가
No
END
closedir

Execute_ls-l


start
Yes
No
Yes
if error print not access or no such directory
END
Yes
print error
END
NO
Yes
No
END
close dir

Execute_ls-al



Pseudo code

Cli.c

```
void conv_cmd(char *input, char *output) {
    char *token
    char *tokens[5012]
    int num_tokens=0
    char input_copy[5012]
    copy input_copy -> input
    while(token ==null){
    token parse (" ")
    token save tokens[numtoken++]
    }
    if(token[0]==ls){
        strcpy(output, "NLST")
        and strcat(output, all token)
    }
    else if(token[0]==quit){
        strcpy(output, "QUIT")
        and strcat(output, all token)
    }
    else{
        all token= output
    }
}
```

```
int main(int argc, char **argv) {
    int sockfd, n;
    struct sockaddr_in servaddr, cliaddr;
    char buff[5012], ftp_cmd[5012];

    if (argc == 1 ||argc==2) {
        write "less argument\n"
        exit(1) //fail
    }

    if(argc>3){
        write "too many arguments\n"
        exit(1)
    }

    socket make (AF_INET, SOCK_STREAM,0)
    if(socket make fail){
        print errno
        exit(EXIT_FAILURE)
    }
    server family, addr, prot set

    connect server
    if(fail server connect){
        print errno
        exit(EXIT_FAILURE)
    }
```

```
    while(1){
        write "> "
        conv_cmd(buf, ftp_cmd)
        Send the converted command to the server
        if(send err){
            print errno
            exit(EXIT_FAILURE)
        }
        recive buf to the server

        print buff
        if(buf==Program quit!!){
            break
        }

    }
    close socket
}
```

Srv.c

Client_info fun

```
int client_info(struct sockaddr_in *cliaddr) {
    //ip inet_nota use
    char *ip = inet_ntoa(cliaddr->sin_addr);
    //port ntohs use
    uint16_t port = ntohs(cliaddr->sin_port);

   // Make client information a string
    char output[256]; // Maximum size of the output string

    write "==========Client info==========="
    wirte       "client IP: ip"
    write      "client port: port"
    write "================================"
    return 0
}
```

NLST fun

```
void execute_NLST(char *result_buff, char **argv) {
    path = NULL
    dir = NULL
    struct dirent *entry

    // Explore the second argument first
    for i = 1 to the end of argv:
        // Consider it a path if it is a non-optional argument
        if the first character of argv[i] is not '-':
            path = argv[i]
            break

    // Open the directory
    if path is not NULL:
        open the directory at path
    else:
        open the current directory

    // Handle error if directory opening fails and exit
    if(not exist){
        result_buff= "Directory does not exist\n"
        print errno
        return
    }
    if(not access){
        result_buff= "Permission denied\n"
        print errno
        return
    }
    while (readdir(dir)!=NULL){
        if(hidden file) -> no save
        result buf= entry->name
    }
    closedir(dir)
    }
```

NLST-a fun

```
void execute_NLST_a(char *result_buff, char **argv) {
    path = NULL
    dir = NULL
    struct dirent *entry

    // Explore the second argument first
    for i = 1 to the end of argv:
        // Consider it a path if it is a non-optional argument
        if the first character of argv[i] is not '-':
            path = argv[i]
            break

    // Open the directory
    if path is not NULL:
        open the directory at path
    else:
        open the current directory

    // Handle error if directory opening fails and exit
    if(not exist){
        result_buff= "Directory does not exist\n"
        print errno
        return
    }
    if(not access){
        result_buff= "Permission denied\n"
        print errno
        return
    }
    while (readdir(dir)!=NULL){
        if(hidden file) -> also save buf
        result_buff= entry->name
    }
    closedir(dir)
    }
```

NLST-l fun

```
void execute_NLST_l(char *result_buff, char **argv){
    DIR *dir
    struct dirent *entry
    struct stat fileStat
    char buf[256]
    path argument is X= path->current
    struct file_info

    if(no exist file or dirctory){
        print errno
        result_buff="Error: No such file or directory\n"
    }
    if(path= file name){
        file flag=1;
    }
    if(file flag=0 && not access ==-1){
        print errno
        result_buff="Error: No read permission\n"
    }

    if(path ==directory){
        opendir(path)
        if(dir==NULL){
            print errno
            result_buff="Error opening directory\n"
            return
        }
```

```
101         if(not read){
102             print errno
103             result_buff="Error: No read permission\n"
104             closedir
105             return
106         }
107         while ((entry = readdir(dir)) != NULL) {
108             if(hiddenfile-> no save)
109             result_buff= Save that information in detail
110         }
111         closedir
112     }
113     else {//file case
114     result_buff= Save that information in detail
115     }
116 }
```

NLST -al fun

```
void execute_NLST_al(char *result_buff, char **argv){
    DIR *dir
    struct dirent *entry
    struct stat fileStat
    char buf[256]
    path argument is X= path->current
    struct file_info

    if(no exist file or dirctory){
        print errno
        result_buff="Error: No such file or directory\n"
    }
    if(path= file name){
        file flag=1;
    }
    if(file flag=0 && not access ==-1){
        print errno
        result_buff="Error: No read permission\n"
    }

    if(path ==directory){
        opendir(path)
        if(dir==NULL){
            print errno
            result_buff="Error opening directory\n"
            return
        }
```

```
        if(not read){
            print errno
            result_buff="Error: No read permission\n"
            closedir
            return
        }
        while ((entry = readdir(dir)) != NULL) {
            if(hiddenfile-> also save)
            result_buff= Save that information in detail
        }
        closedir
    }
    else {//file case
    result_buff= Save that information in detail
    }
}
```

Cmd process fun

```
int cmd_process(char *buff, char *result_buff) {
    // Initialize result_buff
    memset(result_buff, 0, SEND_BUFF);

    char *command;
    char *args[5012];
    int argc = 0;

    Parse buff by " " and store it in a command array

    if(args[0]==NLST){
        //ls -a
        if(args[1]=='-a'){
            write FTP command
            execute_NLST_a(result_buff, args);
        }
        //ls -l
        else if(args[1]=='-l'){
            write FTP command
            execute_NLST_l(result_buff, args);
        }
        //ls -al
        else if(args[1]=='-al'){
            write FTP command
            execute_NLST_al(result_buff, args);
        }
        //ls
        else if(argc==2 && args[1][0]!='-'){
            write FTP command
            execute_NLST_al(result_buff, args);
        }
```

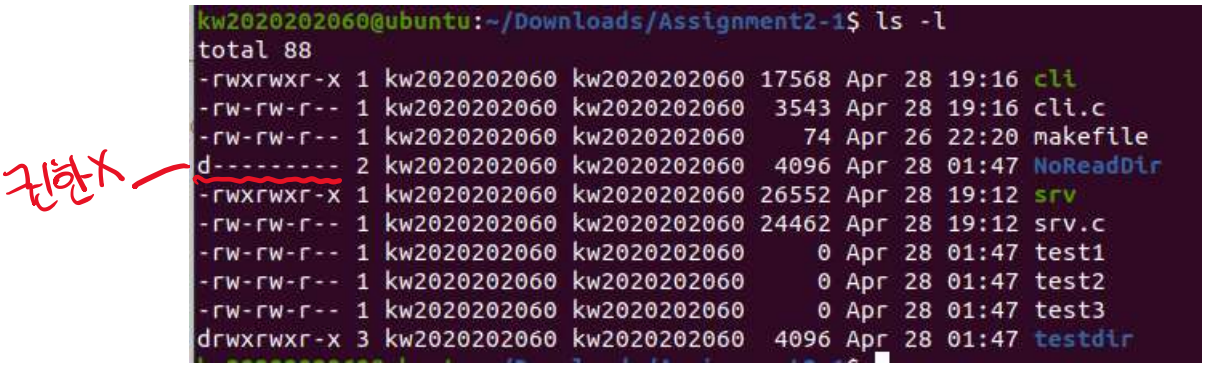
```
    else{
        if(invalid options){
        write "Invalid option\n"
        result_buff= "Invalid option\n"
        }
        else{
        write "too many path arguments\n"
        result_buff= "too many path arguments\n"
        }
    }
}
else if(args[0]==QUIT){
    if(argc>1){
        if(use option){
            write "QUIT is Not option\n"
            result_buff= "QUIT is Not option\n"
        }
        else{
            write "QUIT is No argument\n"
            result_buff= "QUIT is No argument\n"
        }
    }
    else{
        wirte QUIT
        result_buff=Program quit!!
    }
}
else{
    write Input Command
    result_buff="Invalid command\n"
}
return
}
```

main

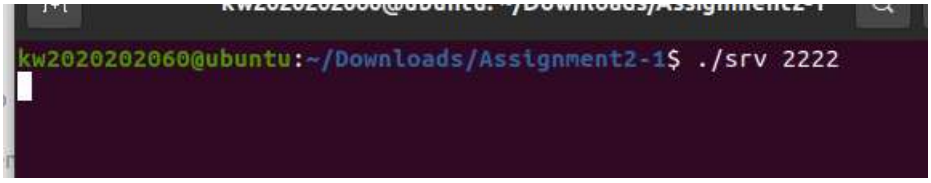
```
int main(int argc, char **argv) {
    int listenfd, connfd, n;
    struct sockaddr_in servaddr, cliaddr;
    char buff[MAX_BUFF], result_buff[SEND_BUFF];

    if(argc==1){
        write "less arguments\n"
        exit(1)
    }
    if(argc>2){
        write "too many arguments\n"
        exit(1)
    }
    if(If argv is not a number){
        write "Port number must be numeric"
        exit(1)
    }
    listenfd = socket(AF_INET, SOCK_STREAM, 0)
    if(listenfd is err){
        print errno
        exit(EXIT_FAILURE)
    }
    memset(&servaddr, 0, sizeof(servaddr));
    sever famuly= AF_INET
    server addr =htons(INADDR_ANY)
    server port=htons(atoi(argv[1]))
    bind()
    if(bind is fail){
        print errno
        exit(EXIT_FAILURE)
    }
    listen(setting queue size 5)
    if(listen fail){
        print errno
        exit(EXIT_FAILURE)
    }
```

```
    for(;;){
        Wait for client connection
        Accept client connection
        Print client information
        Receive command from client
        Process command
        Send result to client
        If result is "Program quit!!\n":
        Close connection and exit program
    }
    close(connfd)
    close(listenfd)
    return 0
}
```

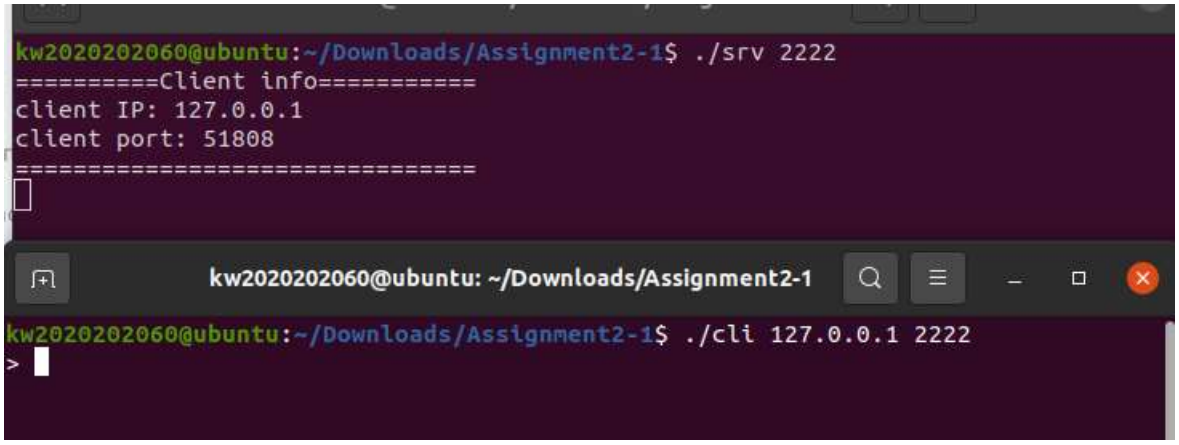
결과화면



먼저 결과를 확인하기 전에 디렉토리 파일들을 보여준다.

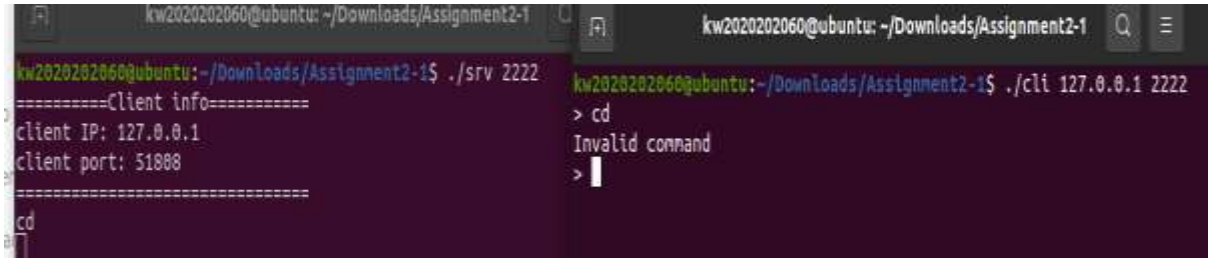


위 서버를 실행한다면 클라이언트에서 연결이 될 때까지 무한 대기를 하는 것을 확인할 수 있다. 이때 아래 그림 과 같이 클라이언트가 연결 요청이 한다면

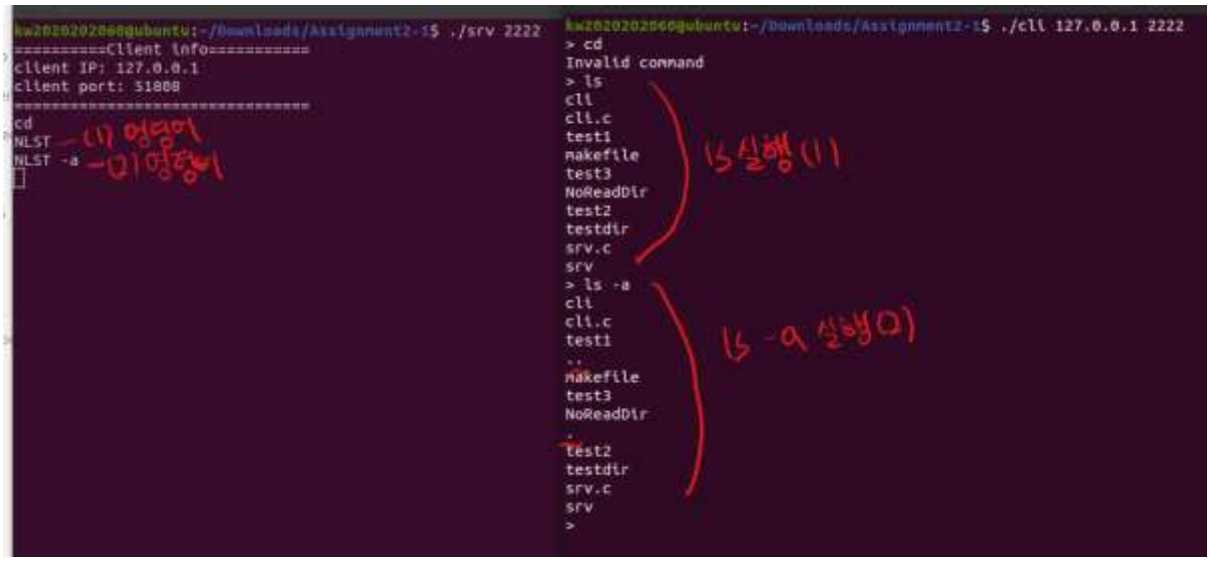


명령어 입력할 수 있는 칸이 클라이언트에서 생성되고 잘 연결되었다는 표시가 서버로 전달됨을 확인할 수 있다.

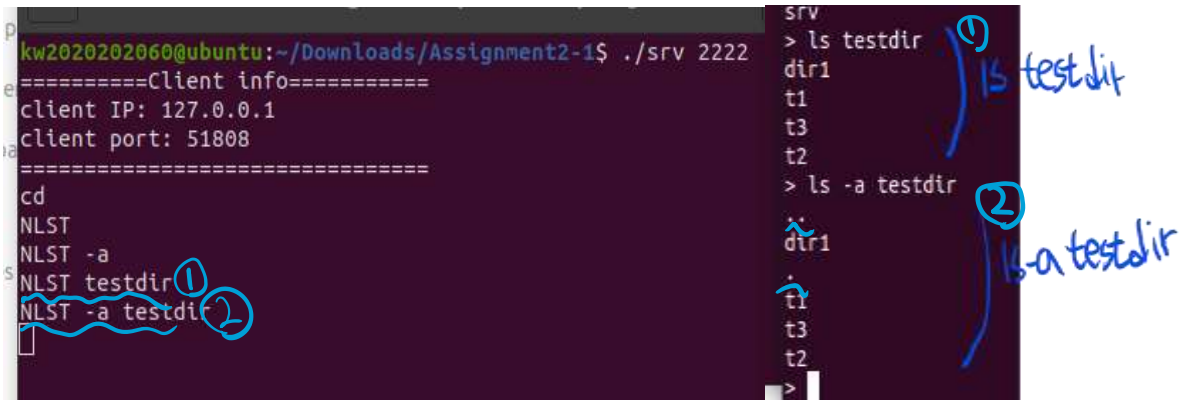
이번 프로젝트에서는 ls 와 quit 명령어만 수행하기에 옳지 않은 명령어가 들어올 경우에는 다음과 같이 에러가 출력됨을 볼 수 있다.



아래는 정상적인 명령어가 들어올 경우에 대한 결과 화면이다.

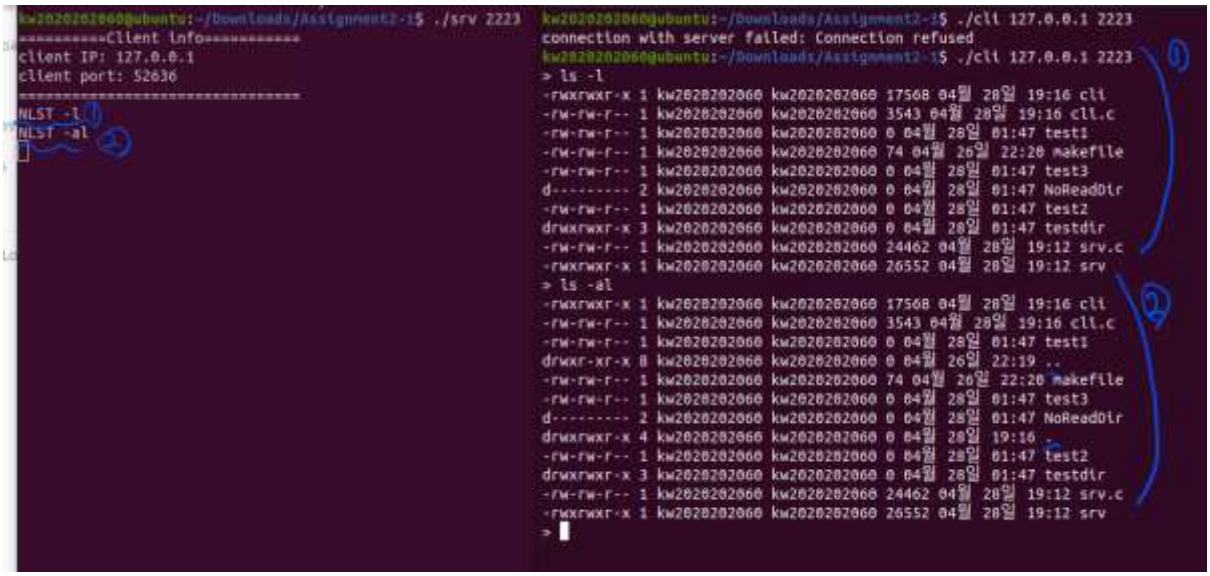


위를 확인하면 ls 와 ls-a 명령어가 수행됨에 따라 그 결과를 클라이언트가 받아서 출력됨을 볼 수 있다. 다음 경우에는 이 명령어들에 추가로 경로가 들어올 때에 대한 검증 결과이다.



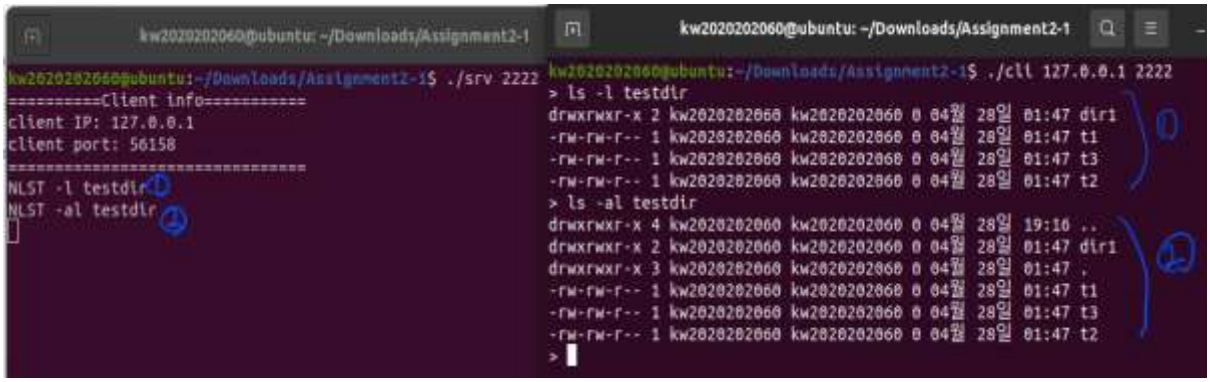
위 결과를 확인하면 testdir 디렉토리 경로에 대한 파일들이 각자 명령어에 맞게 수행됨을 확인할 수 있다.

다음은 ls -l 과 ls -al 에 대한 검증결과이다.



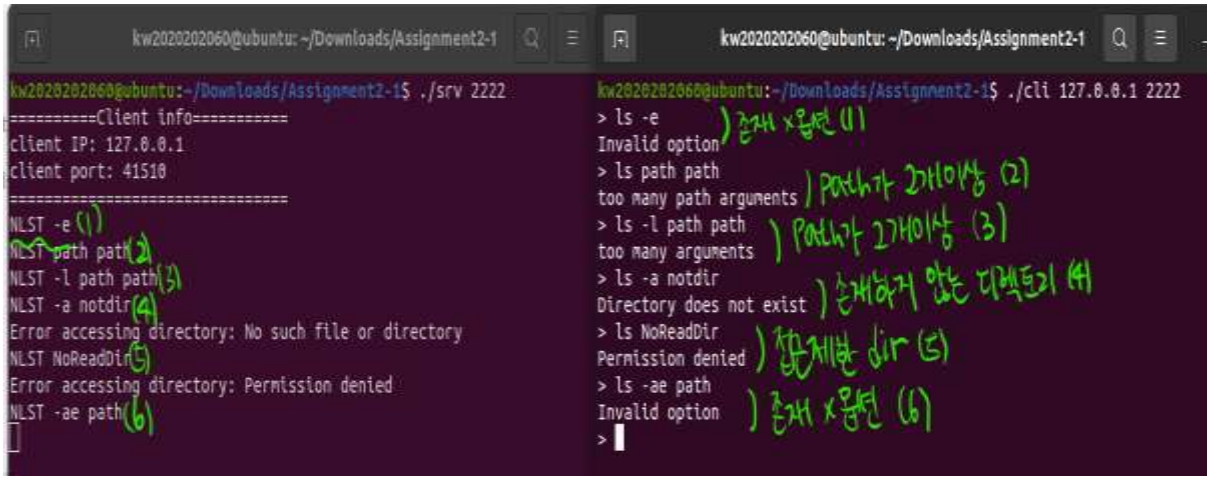
위 결과를 확인한다면 ls -l 결과와 ls -al 결과과 모두 리눅스 형식과 동일하게 출력됨을 볼 수 있다. 이때 -a 옵션이 있다면 숨김 파일까지 출력됨을 확인할 수 있다.

다음은 ls -l, ls -al 에 경로가 들어올 경우에 대한 검증이다.



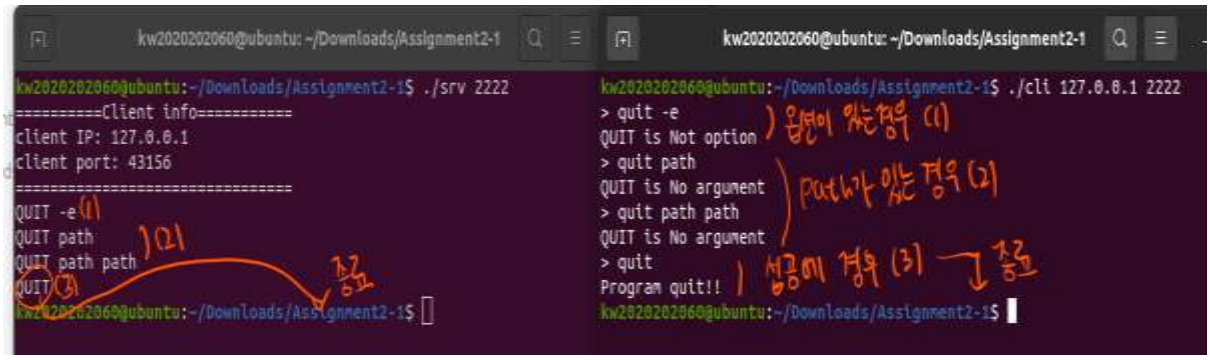
위 결과를 확인하면 testdir 디렉토리 경로에 대한 내용들이 자세히 출력됨을 확인할 수 있다. 이때 -al 인 경우에는 숨김 파일까지 출력됨을 확인할 수 있다.

다음은 정의되지 않는 옵션이나 너무 많은 경로가 들어올 경우, 접근 제한이 없을 경우에 대한 예외 케이스 검증이다.



위 결과화면을 설명과 함께 확인한다면 예외별로 오류 출력 내용이 출력됨을 확인할 수 있다.

다음은 QUIT 명령어가 들어왔을 경우에 대한 결과 화면이다. 이는 불필요한 인자나 옵션이 들어왔을 경우에 프로그램을 종료하지 않고 오류를 출력한다. 만약 QUIT 만 들어온다면 성공 메시지와 함께 서버와 클라이언트를 종료하는 알고리즘으로 설계하였다. 출력 결과는 다음과 같다.



위결과를 설명과 함께 확인한다면 예외처리와 성공적으로 종료될 때 서버와 클라이언트가 종료됨을 확인할 수 있다.

고찰

프로젝트를 진행하면서 겪은 어려움은 기존의 코드를 변형하고 연결하는 작업이었다. 처음에는 이전 프로젝트에서 사용한 코드를 그대로 가져와서 수정하려고 했지만, 새로운 기능을 추가하고 기존 기능을 확장하는 과정에서 많은 문제가 발생했다. 예를 들어 지난 프로젝트에서는 수행되는대로 출력을 하였지만, 이번 프로젝트에서는 이러한 명령어를 실행한 결과를 클라이언트로 다시 전송해야 했기에 복잡하였다.

소켓에 대한 개념과 작동 방식을 이해하는 것이 중요했기 때문에 처음에는 이에 대한 공부를 진행하였다. 소켓을 연결하고 데이터를 전송하는 과정에서 함수들의 순서와 각각의 역할을 이해하는 것이 중요했다. 이를 위해 소켓과 관련된 다양한 함수를 숙지하고, 이들을 어떻게 조합하여 원하는 기능을 구현할 수 있는지를 공부하며 구현할 수 있었다.

이러한 과정에서 소켓에 대한 정의와 작동 원리에 대해 보다 심층적으로 이해할 수 있는 계기가 되었다. 또한, 기존의 코드를 변형하고 새로운 요구사항을 충족시키기 위해 어떻게 노력해야 하는지에 대한 경험도 쌓을 수 있었다. 이러한 경험을 통해 프로그래밍의 핵심 개념을 향상시킬 수 있었고, 앞으로의 프로젝트에 도움되어 최종적인 FTP 서버를 구현할 수 있다고 생각하였다.