

시스템 프로그래밍 실습

[Assignment3-3]

Class : [D]

Professor : [최상호교수님]

Student ID : [2020202060]

Name : [홍왕기]

Introduction

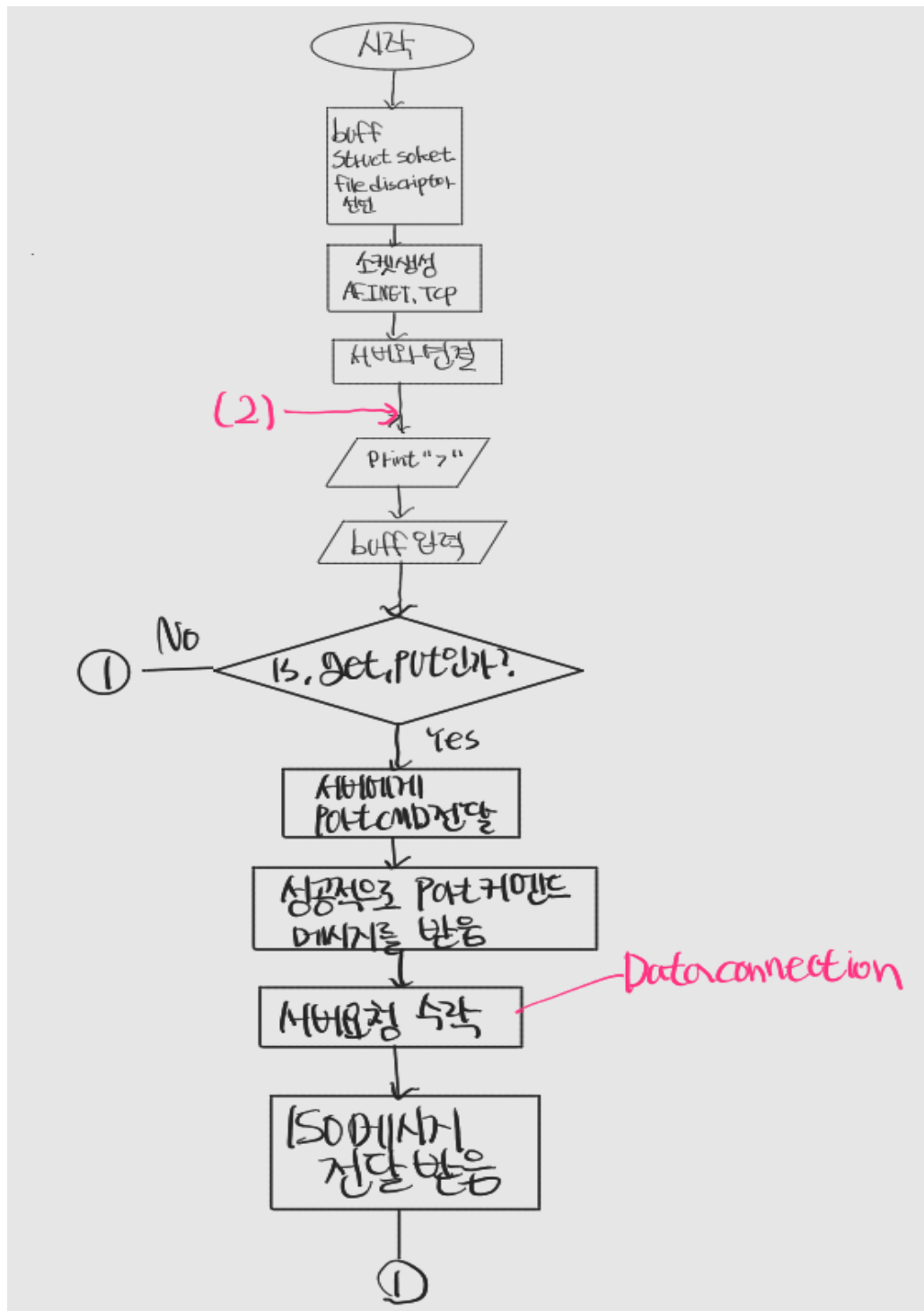
이는 FTP 서버를 만들기 위한 마지막 프로젝트로, 소켓을 사용해 클라이언트와 서버를 연결하여 클라이언트가 명령어를 요청했을 때 서버는 그에 맞는 응답을 전해야 한다. 12 가지 명령어와 로그인 기능, 그리고 로그 파일을 통해 서버를 구현한다. 이 프로젝트의 목표는 다각적이다. 먼저, 클라이언트-서버 간의 안정적인 통신을 구축하여 사용자와 서버 간의 데이터 전송이 원활하게 이루어지도록 한다. 이를 위해 소켓 프로그래밍을 활용하여 프로토콜 연결을 설정하고 유지하는 방법을 익힌다. 또한, 클라이언트가 파일을 업로드, 다운로드, 삭제, 디렉토리 변경 등 다양한 작업을 수행할 수 있도록 표준 FTP 명령어 12 가지를 구현할 계획이다. 이 명령어들은 FTP 프로토콜의 핵심 기능을 포괄하며, 클라이언트가 원하는 작업을 정확하게 수행할 수 있도록 한다.

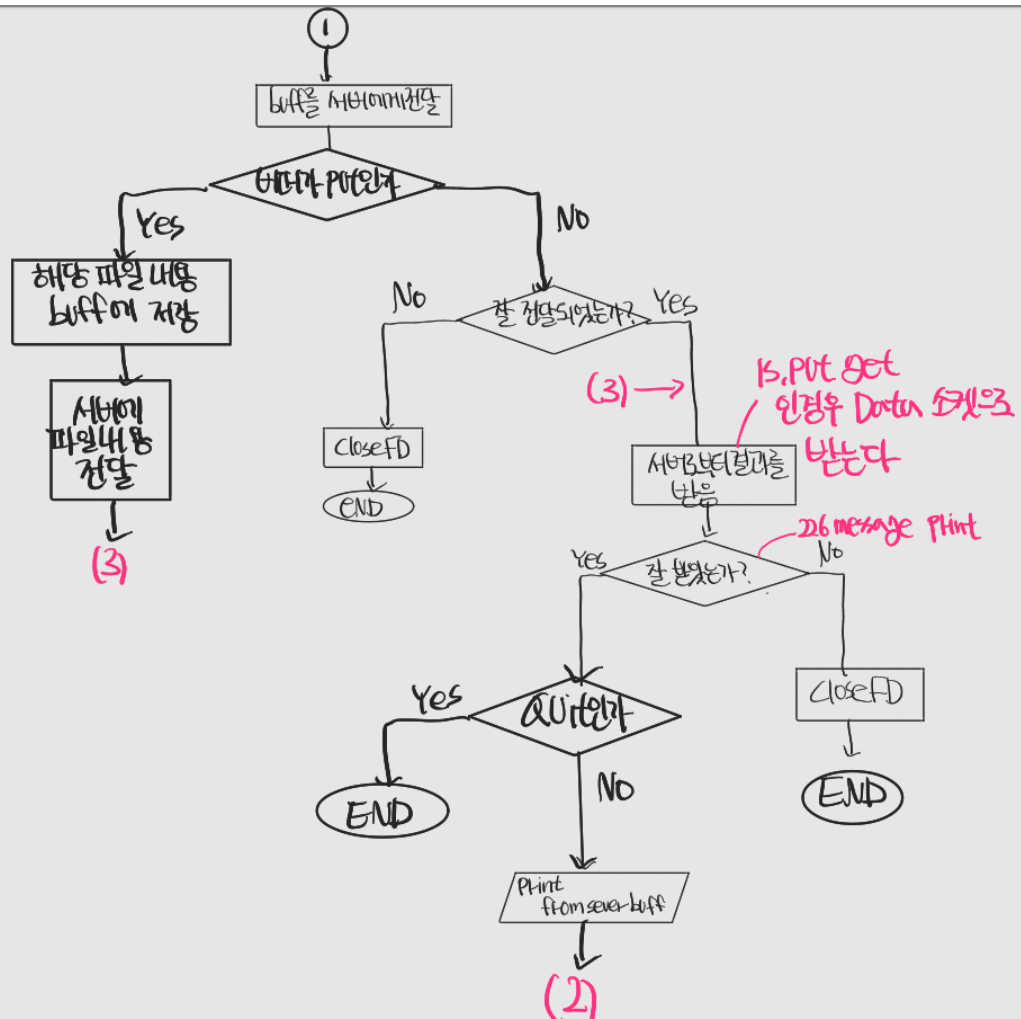
더 나아가, 보안 측면에서도 철저히 준비하고자 한다. 사용자 인증을 위한 로그인 기능을 구현하여, 승인된 사용자만이 서버에 접속하고 명령어를 실행할 수 있도록 제한한다. 이를 통해 불법적인 접근을 방지하고 서버의 안전성을 높인다. 각 사용자의 로그인 시도와 명령어 실행 내역은 모두 로그 파일에 기록된다. 이 로그 파일은 이후에 서버 활동을 추적하고 분석하는 데 중요한 자료가 되며, 문제 발생 시 신속한 대처를 가능하게 한다.

결론적으로, 이 프로젝트는 기술적 역량을 향상시키고 실무적인 경험을 쌓는 중요한 기회이다. 클라이언트-서버 통신의 기초부터 보안 강화, 로그 관리에 이르기까지 전반적인 FTP 서버 구축 과정을 통해, 실제 서비스 운영에 필요한 다양한 기술을 습득하는 것을 목표로 한다.

Flow chart

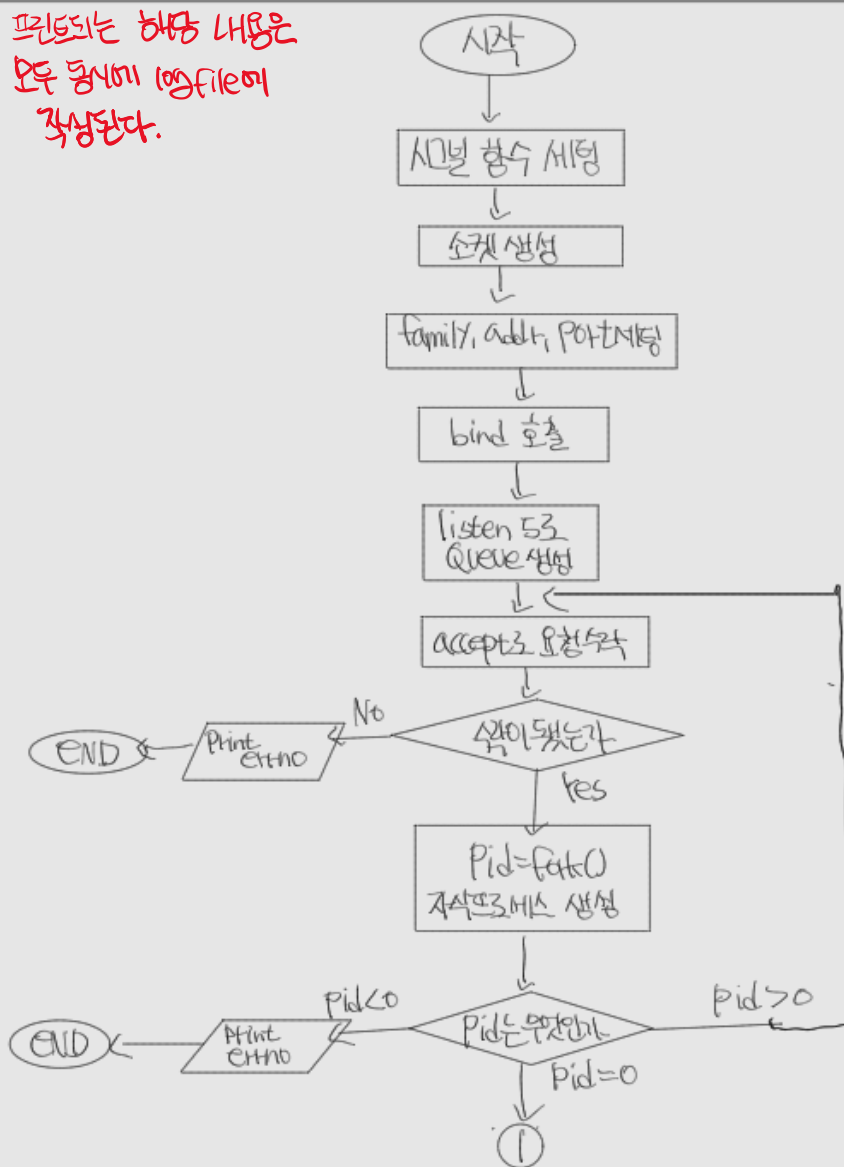
cli.c

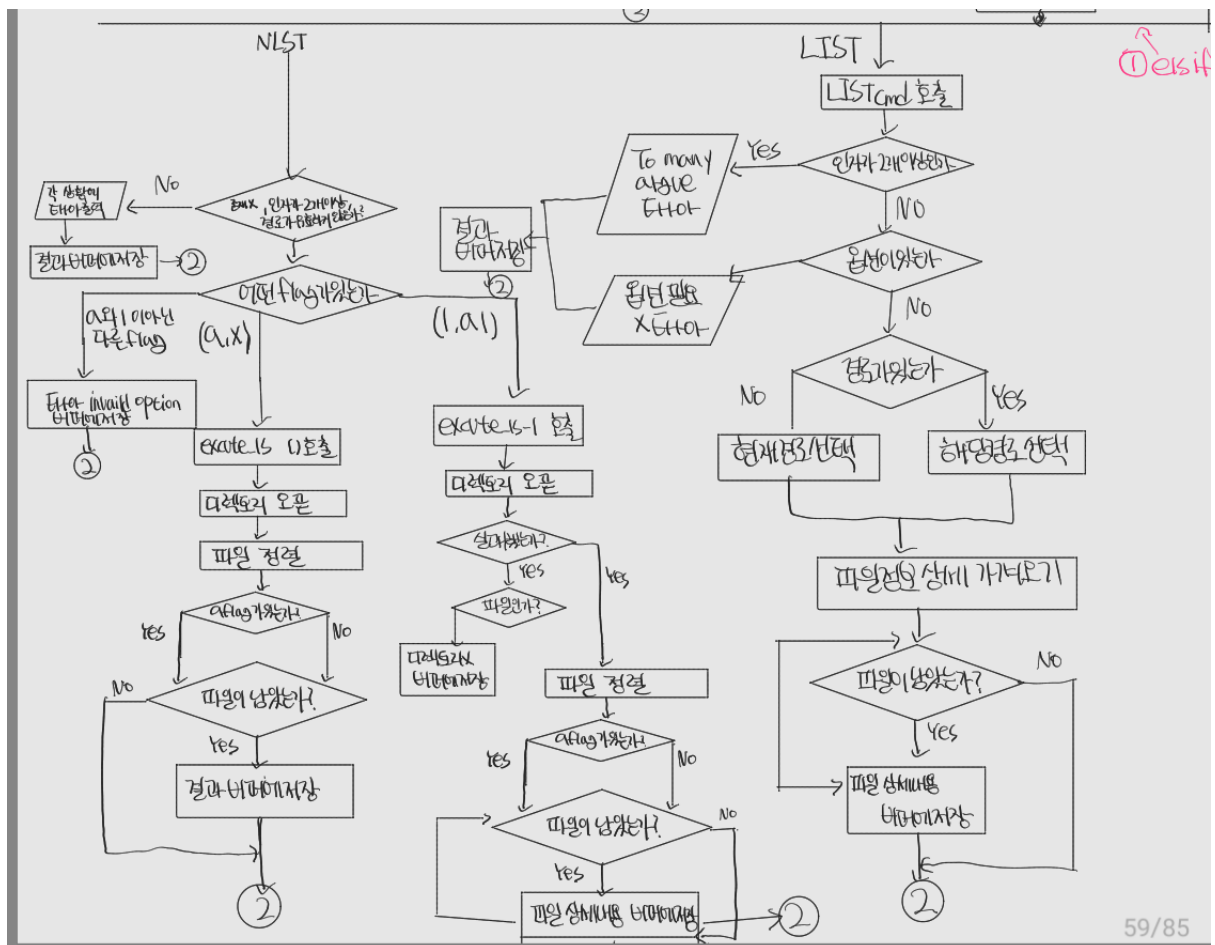
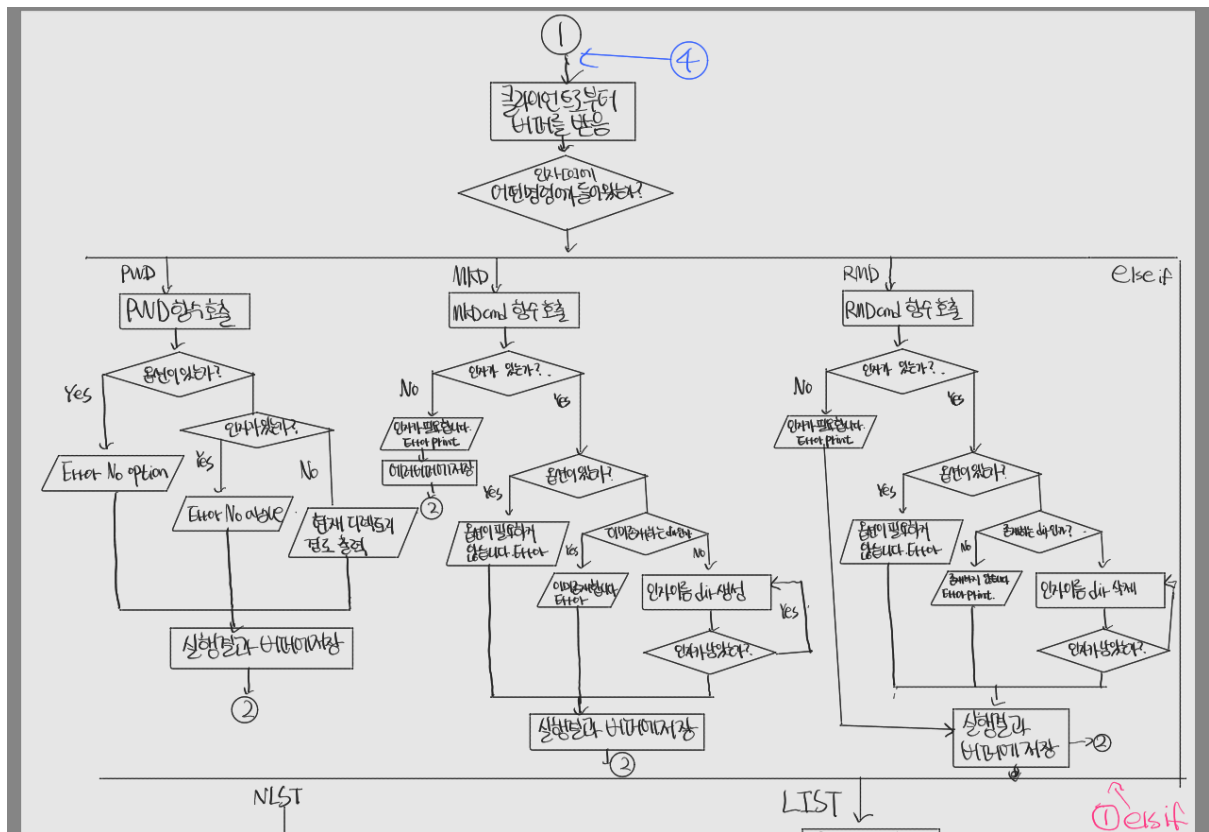


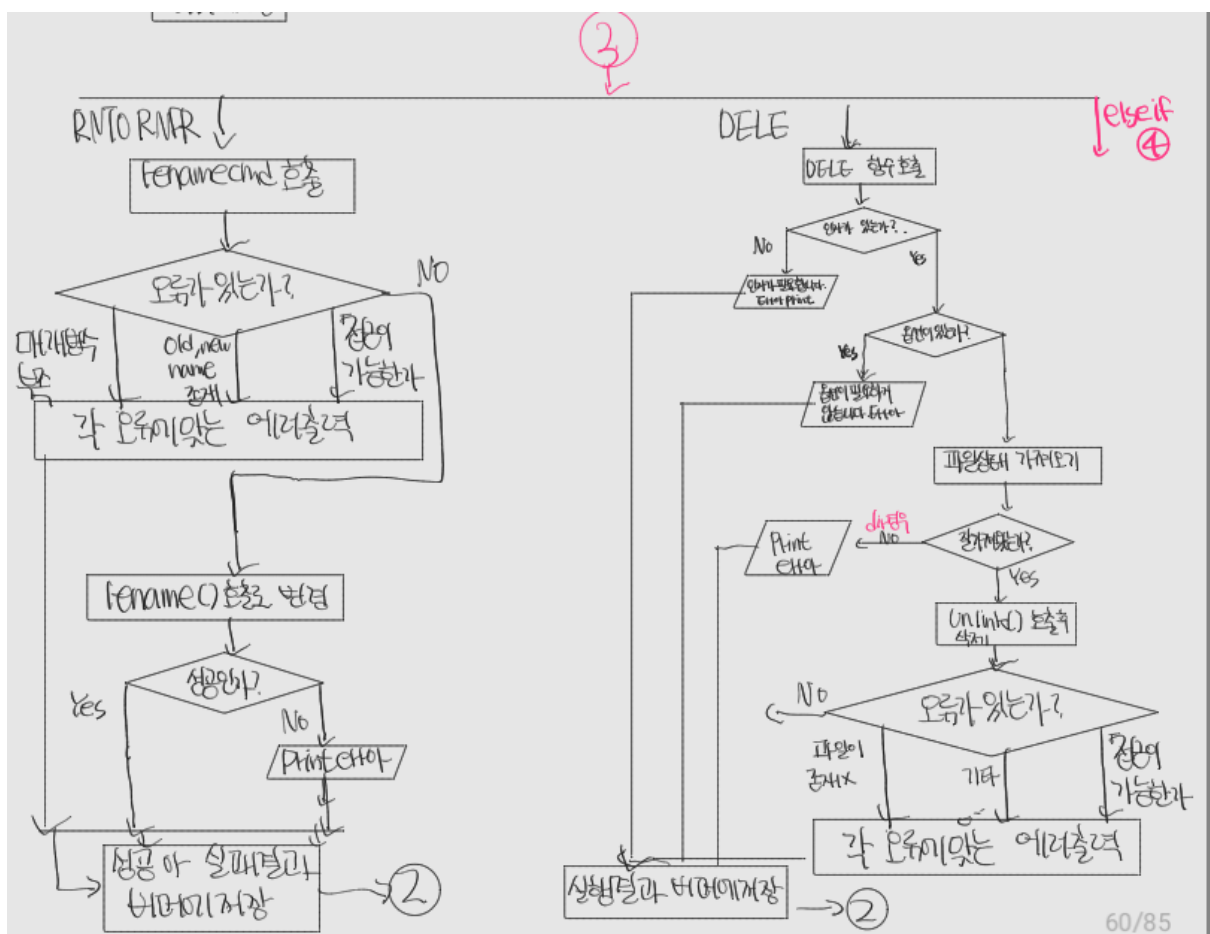
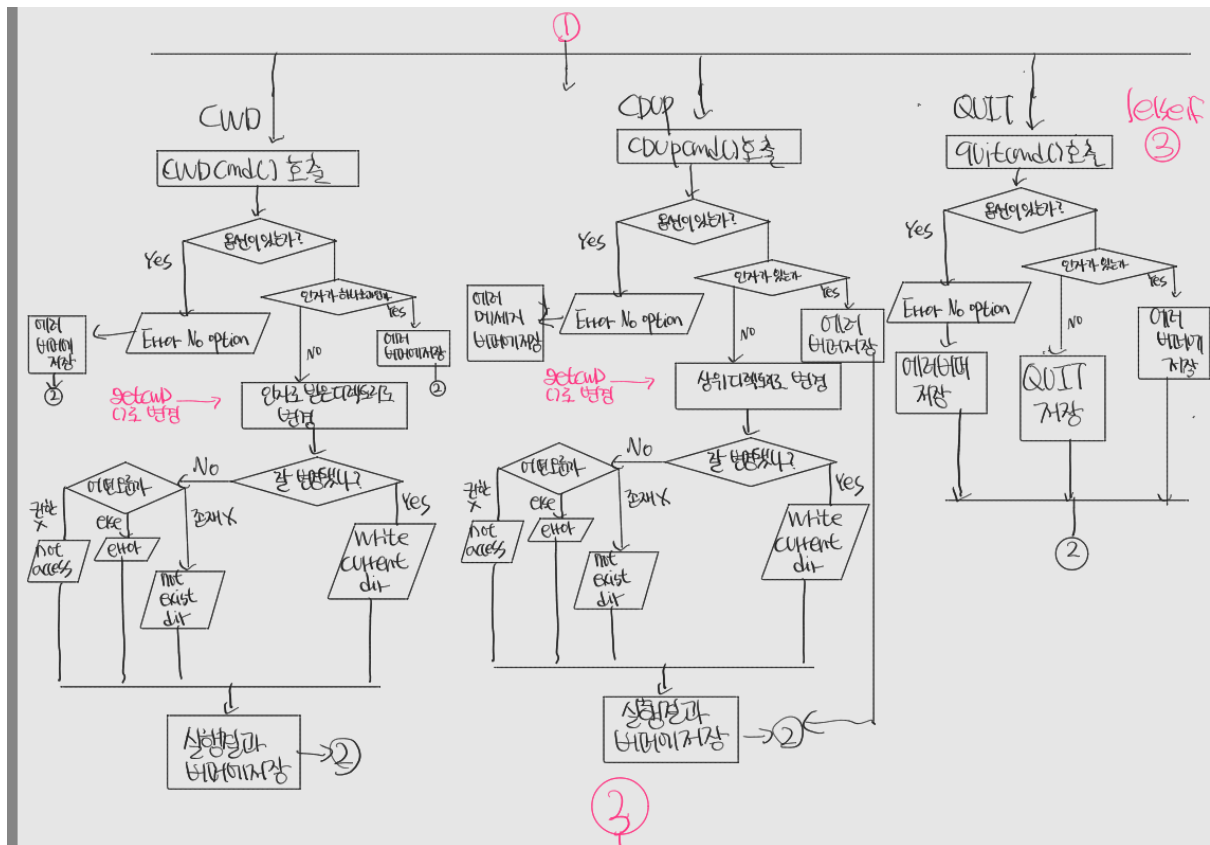


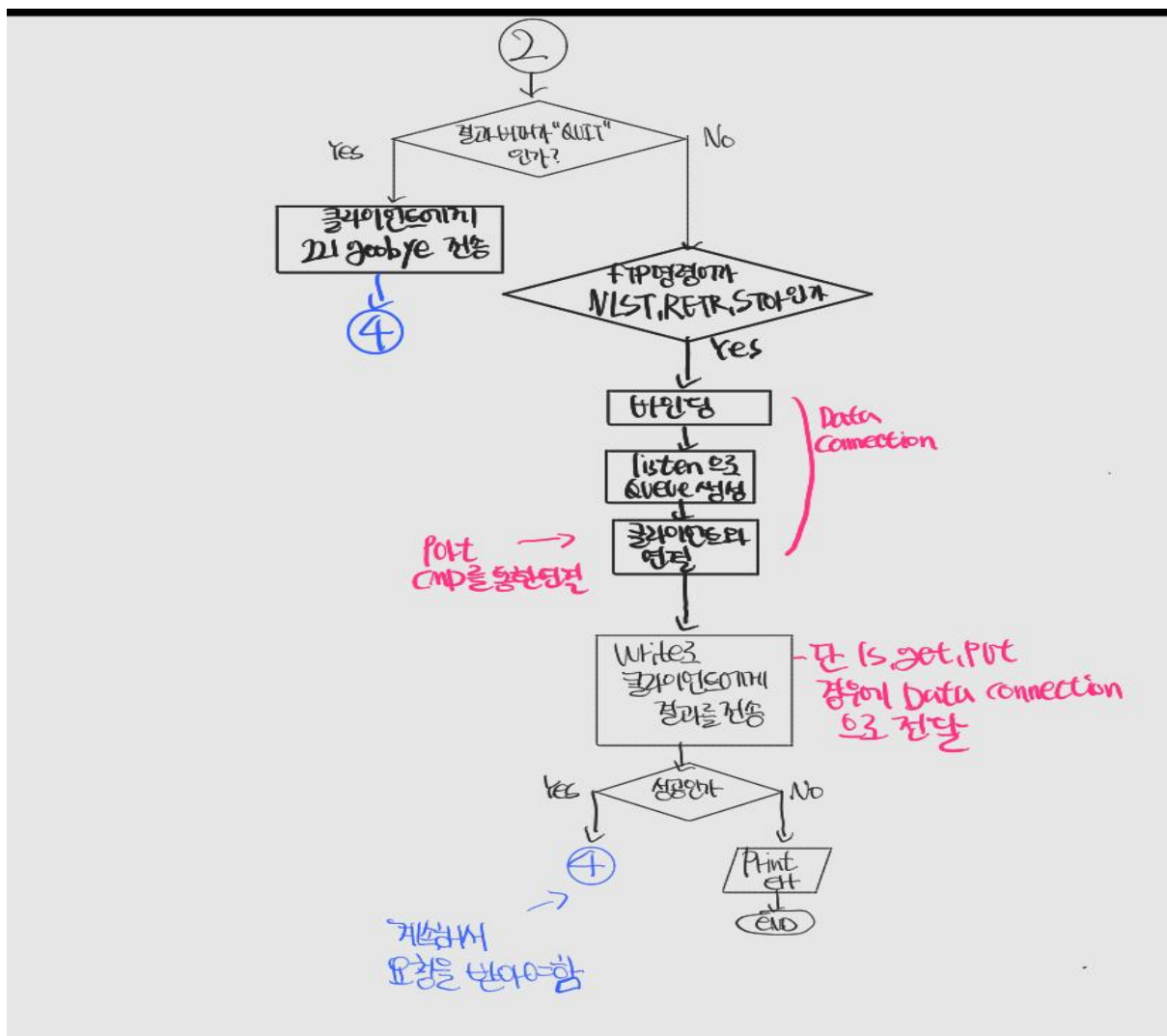
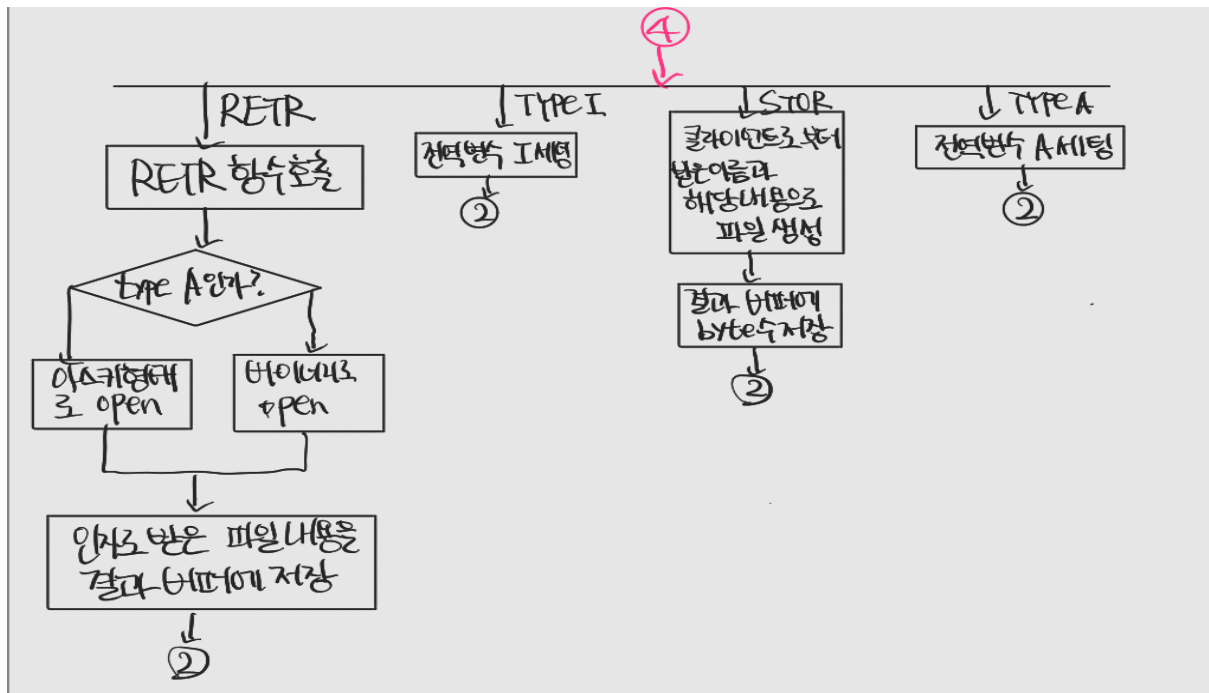
Srv.c

프린트되는 해당 내용은
모두 공백이 영파일에서
작성된다.









Pseudo code

cli.c

```
void SendPUT(char *fileName, char type) {  
    if(type=A){  
        | 아스키 모드로 파일 오픈  
    }  
    else if(type=I){  
        | 바이너리 모드로 파일 오픈  
    }  
    else{  
        | 550 에러 버퍼에 저장  
    }  
    if(파일이 없다면){  
        | 550 에러 버퍼에 저장  
    }  
  
    파일 내용을 읽어 버퍼에 저장후 길이를 bytesRead2에 저장  
    만약 오류가 났다면 550 에러 버퍼에 저장  
  
    파일 닫기  
}
```

```
5  
6 v char* convert_str_to_addr(char *str, unsigned int *port) {  
7     static char addr[32]  
8     char copystr[MAX_BUFF]  
9     char ip_part[4][4]  
10    int ip[4], p1, p2  
11  
12    copystr=str  
13  
14    *,*,*,*, 형식에 대한 str을 ,기준으로 ip 배열에 파싱  
15    남은 *,*를 ,을 기준으로 p1 p2로 파싱  
16  
17    파싱한 것들을 PORT *,*,*,*,*,*로 합쳐 addr에 저장  
18  
19    return addr  
20 }
```

```
void conv_cmd(char *input, char *output) {  
    char *token  
    char *tokens[BUF_SIZE]  
    int num_tokens = 0  
    char input_copy[BUF_SIZE]  
    strcpy(input_copy, input)  
  
    Tokenize the input string based on whitespace  
  
    if (num_tokens > 0 && token[0] == 'ls'){  
        output = NLST  
        Attaching to the remaining factor output  
    }  
    else if (num_tokens > 0 && token[0] == 'quit'){  
        output = QUIT  
        Attaching to the remaining factor output  
    }  
  
    else if (num_tokens > 0 && token[0] == 'dir'){  
        output = LIST  
        Attaching to the remaining factor output  
    }  
}
```

```
else if (num_tokens > 0 && token[0] == 'pwd'){
output = PWD
Attaching to the remaining factor output
}

else if ((num_tokens > 0 && token[0] == 'cd' AND token[1] = ' .. ') != 0){
output = CWD
Attaching to the remaining factor output
}

else if ((num_tokens > 0 && token[0] == 'cd' AND token[1] = ' .. ') == 0){
output = CDUP
Attaching to the remaining factor output
}

else if (num_tokens > 0 && token[0] == 'mkdir'){
output = MKD
Attaching to the remaining factor output
}

else if (num_tokens > 0 && token[0] == 'delete'){
output = DLELE
Attaching to the remaining factor output
}

else if (num_tokens > 0 && token[0] == 'rmdir'){
output = RMD
Attaching to the remaining factor output
}
```

```

    }
    else if (num_tokens > 0 && token[0] == 'rename'){
        if (num_tokens == 3) {
            output= "RNFR token[0] RNT0 token[1]"
        }
        else output=err
    }
    ✓ else if (num_tokens > 0 && strcmp(tokens[0], "put") == 0) {
        output=STOR
        Attaching to the remaining factors output
        ✓ if (num_tokens==2){
            |     Sendput 함수에 타입과 파일 이름 전달
            | }
            else 에러코드 버퍼에 저장
        }
    ✓ else if (num_tokens > 0 && strcmp(tokens[0], "bin") == 0) {
        output=TYPE I
        Attaching to the remaining factors output
        Type 전역변수 I로 세팅
    }
    ✓ else if (num_tokens > 0 && strcmp(tokens[0], "type") == 0
        && num_tokens > 1 && strcmp(tokens[1], "binary") == 0) {
        output=TYPE I
        Attaching to the remaining factors output
        Type 전역변수 I로 세팅
    }
    ✓ else if (num_tokens > 0 && strcmp(tokens[0], "ascii") == 0 ){
        output=TYPE A
        Attaching to the remaining factors output
        Type 전역변수 A로 세팅
    }
    ✓ else if (num_tokens > 0 && strcmp(tokens[0], "type") == 0 &&
        num_tokens > 1 && strcmp(tokens[1], "ascii") == 0) {
        output=TYPE A
        Attaching to the remaining factors output
        Type 전역변수 A로 세팅
    }
    ✓ else{
        |     Attaching to the remaining factors output
    }
}

```

```

int main(){
    인자가 3개 이상이면 오류 출력후 종료
    소켓 생성-> 만약 실패시 오류 출력 후 종료

    소켓 sin_family, addr, port 세팅

    서버와 연결
    while(1){
        printf ftp>
        커맨드 입력 후 버퍼에 저장
    }
    만약 아무것도 입력하지 않았다면 재입력

    conv_cmd로 커맨드 전달 -> ftp커맨드로 변경이 이루어진다

    if(ftp command가 NLST, RETR, STOR경우){
        데이터 소켓 생성-> 만약 실패시 에러 출력 후 종료
        10000~60000사이에 랜덤포트 생성

        데이터 연결할 소켓 세팅

        바인딩 후
        listend함수로 queue생성

        convert_str_to_addr함수로 host ip설정
        host ip 서버로 전달-
        200 포트 커맨드 성공 메시지를 read
        해당 메시지 출력

        ftp커맨드 서버에 전달

        150 성공 메시지를 전달받아서 출력
        서버로부터 받은 연결 요청 수락(data connection)
    }
}

```

```

if(ftp커맨드가 STOR이면){
    데이터 connection으로 버퍼 전송
    226 성공메시지를 read하여 출력
    해당 바이트 내용 출력
}
else{
    data connection으로 결과 출력 (ls ,get)

    if(만약 ftp커맨드가 RETR이라면){
        버퍼 출력
    }
    else{
        if(CheckList==0){
            buff가 550 오류 코드면 해당 내용 출력
        }
        else{
            파일을 쓰기 모드로 열기
            해당 내용을 buff에 저장
            close
        }
    }
}
}
서버로 부터 read한 것을 버퍼에 저장 후 출력
if(ftp커맨드가 NLST라면){
    사이즈가 0이 아니면 해당 바이트 출력
    0이라면 550 오류 출력
}
if(ftp커맨드가 RETR이라면){
    서버로부터 받은 내용을 버퍼에 기록 후 출력
}
}
close data 소켓 영역
}

```

```

else{
    서버로 ftp명령어 전달

    서버로 부터 받은 결과 출력 -> 에러시 오류 출력후 종료
    만약 221 Goodbye를 받았다면 종료
}
}

```

Srv.c

```
char* convert_str_to_addr(const char *str, unsigned int *port) {  
    받은 portcmd  
    PORT *,*,*,*,*,* 형태를  
    ,를 기준으로 IP와 port 1 2 로 파싱  
  
    받은 port 1 2를 결합  
    ex-> 127,0,0,1,15000 형태를 addr에 저장  
    return addr  
}
```

```
void execute_NLST(char *result_buff, char **argv) {  
    버퍼를 초기화  
  
    만약 path가 없다면{  
        존재하지 않는 파일, 접근 권한중 맞는 속성에 대한  
        에러 결과를 결과 버퍼에 저장  
    }  
    디렉토리를 open  
    만약 실패할 경우 디렉토리가 아님을 결과 버퍼에 저장  
  
    readdir을 통해  
    해당 디렉토리를 순회하며 결과 결과 버퍼에 저장  
  
    버퍼를 정렬  
}
```

```

5 void execute_NLST_a(char *result_buff, char **argv) {
6     버퍼를 초기화
7
8     만약 path가 없다면{
9         존재하지 않는 파일, 접근 권한중 맞는 속성에 대한
10        에러 결과를 결과 버퍼에 저장
11    }
12    디렉토리를 open
13    만약 실패할 경우 디렉토리가 아님을 결과 버퍼에 저장
14
15    readdir을 통해 숨김파일을 포함하여
16    해당 디렉토리를 순회하며 결과 결과 버퍼에 저장
17
18    버퍼를 정렬
19 }
20
21 void execute_NLST_l(char *result_buff, char **args) {
22     버퍼를 초기화
23
24     만약 path가 없다면{
25         존재하지 않는 파일, 접근 권한중 맞는 속성에 대한
26        에러 결과를 결과 버퍼에 저장
27    }
28    디렉토리를 open
29    만약 실패할 경우 디렉토리가 아님을 결과 버퍼에 저장
30
31    readdir을 통해
32    해당 디렉토리를 순회하며 해당 파일 자세한 내용을
33    결과 버퍼에 저장
34
35    버퍼를 정렬
36 }

```



```

void execute_NLST_al(char *result_buff, char **args) {
    버퍼를 초기화

    만약 path가 없다면{
        존재하지 않는 파일, 접근 권한중 맞는 속성에 대한
        에러 결과를 결과 버퍼에 저장
    }
    디렉토리를 open
    만약 실패할 경우 디렉토리가 아님을 결과 버퍼에 저장

    readdir을 통해 숨김파일을 포함하여
    해당 디렉토리를 순회하며 해당 파일 자세한 내용을
    결과 버퍼에 저장

    버퍼를 정렬
}

void RMDcmd(char *result_buff, char **argv)
{
    if (no argue)
    {
        result_buff= Error : aregument is required->return
    }
    if (option use)
    {
        result_buff= Error : invaild option->return
    }
    remove a directory using rmdir() with the name received as a factor
    if (dir no exist) { result_buff= Error : fail to remove }
    return
}

```

```

void RMDcmd(char *result_buff, char ** argv){
if (no argue){
result_buff= Error : aregument is required->return
}
if (option use){
result_buff= Error : invaild option->return
}
remove a directory using mkdir() with the name received as a factor
if (dir no exist) { result_buff= Error : fail to remove }
return
}

```

```

void CWDcmd(char *result_buff, char **argv)
{
    if (option use)
    {
        result_buff= Error : invaild option->return
    }
    if (argue > 1)
    {
        result_buff= Error : too many arguements->return
    }
    use chdir(), getcwd()
    if (success change directory)
    {
        if (getcwd() == 1)
        |   result_buff= CWD \n current directory else error print
        }
    else if (errno == EACCES)
    {
        result_buff= Error : permission denied
        return
    }
    else if (errno == ENOENT)
    {
        result_buff= Error : directroy not found
        return
    }
    else result_buff= error ->return
}

```

```

2
3  ✓ void RETR(char *result_buff, char **argv) {
4      result_buff 초기화
5      filename =argv[1]< 파일 이름이 담겨있다
6      타입이 A면 아스키 형태로 오픈
7      타입이 I면 바이너리 형태로 오픈
8
9      ✓ if(file==null){
10         |   550 에러 코드 버퍼에 저장
11         }
12
13         파일에 사이즈 결정
14         만약 파일 사이즈가 buff크기보다 크다면 오류출력
15
16         해당 파일 내용을 읽어 버퍼에 저장
17         해당 내용 길이를 bytes_read에 저장
18
19         파일 닫기
20     }
21
22  ✓ void STORcmd(const char *data, const char *filename, char mode) {
23      |   타입이 A면 아스키 형태로 오픈
24      ✓   타입이 I면 바이너리 형태로 오픈
25      ✓   if(file==null){
26         |   550 에러 코드 버퍼에 저장
27         }
28
29         해당 버퍼를 읽어 파일에 쓰기
30         해당 내용 길이를 bytes_read에 저장
31
32         파일 닫기
33     }

```

```

void CDUPcmd(char *result_buff, char **argv)
{
    if (option use)
    {
        result_buff= Error : invaild option->return
    }
    if (argue > 1)
    {
        result_buff= Error : too many arguements->return
    }
    use chdir(), getcwd()
    if (success change directory(..))
    {
        if (getcwd() == 1)
        | result_buff= CDUP \n current directory else error print
    }
    else if (errno == EACCES)
    {
        result_buff= Error : permission denied
        return
    }
    else if (errno == ENOENT)
    {
        result_buff= Error : directroy not found
        return
    }
    else result_buff= error ->return
}

```

```

void DELEcmd(char *result_buff, char **argv)
{
    if (argv[1] == NULL) result_buff= Error : missing operand
    if (use option) result_buff= Error : invaild option
    int i = 1
    import the status of a file or directory

    if (Failed to get the status of a file or directory)
    {
        result_buff=error
        i++ continue
    }
    if (is file)
    {
        use unlink() delete file
    }
    else
    {
        if (file no exist) result_buff=Error: file name does not exist
        else if(file no access) result_buff= Error:permission denied for File
        else result_buff= error
    }
    return
}

```

```

void RenameCmd(char *result_buff, char **argv)
{
    버퍼 초기화
    if (argv 1 || 2 || 3 || 4 = NULL)
    {
        result_buff= Error invaild number of arguments
        return
    }
    if (old name not exist)
    {
        result_buff= Error : file name does not exist
        return
    }
    if (RNFR && RNT0 != argv[0] && argv[2])
    {
        result_buff= Error: invalid command format
        return
    }
    if (old name or new name is NULL)
    {
        result_buff= Error: invaild file name
        return
    }
    if (file or dir is exist)
    {
        result_buff= Error: name to change already exist
        return
    }
    rename(oldname, newname) if (rename == -1) result_buff=Error
    return
}

```

```

void LISTcmd(char *result_buff, char **argv)
{
    버퍼를 초기화

    만약 path가 없다면{
        존재하지 않는 파일, 접근 권한중 맞는 속성에 대한
        에러 결과를 결과 버퍼에 저장
    }
    디렉토리를 open
    만약 실패할 경우 디렉토리가 아님을 결과 버퍼에 저장

    readdir을 통해 숨김파일을 포함하여
    해당 디렉토리를 순회하며 해당 파일 자세한 내용을
    결과 버퍼에 저장

    버퍼를 정렬
}

```

```

int cmd_process(char *buff, char *result_buff) {
클라이언트로 부터 받은 버퍼를 공백 기준으로 파싱
args배열에 저장

if (배열 첫번째가 NLST 인경우){
(인자갯수 초과, 옳지 않은 옵션 인 경우에는 에러 결과를
결과 버퍼에 저장)

-a =excute ls -a 함수 호출
-al =excute ls -al 함수 호출
-l= excute ls -l 함수 함수 호출
no opt= excute ls 함수 호출
}

else if (argv[0] == 'PWD') go PWDcmd(result_buff,args)
else if (argv[0] == 'MKD') go MKDcmd(result_buff,args)
else if (argv[0] == 'RMD') go RMDcmd(result_buff,args)
else if (argv[0] == 'QUIT') go QUITcmd(result_buff,args)
else if (argv[0] == 'CWD') go CWDcmd(result_buff,args)
else if(argv[0] == 'CDUP') go CDUPcmd(result_buff,args)
else if (argv[0] == 'DELE') go DELEcmd(result_buff,args)
else if (argv[0] == 'LIST') go LISTcmd(result_buff,args)
else if (argv[0] == 'RNFR' && argv[2] == 'RNT0') go renamecmd(result_buff,args)
else if (argv[0] == 'QUIT') {옵션, 인자가 있는 경우는 에러를 버퍼에 저장,
| | | | | | | | | | 그것이 아닌 경우에는 QUIT출력 }
}
}

```

```

else if(argv[0]&&argv[1]== TYPE I) ->print TYPE I
else if(argv[0]&&argv[1]== TYPE A) ->print TYPE A
else if(argv[0]==RETR)-> go RETR()
else if(argv[0]==SOTR)-> putCheak=1
}

```

```

int main(int argc, char **argv) {

    인자가 하나일 경우에는 인자부족 에러 출력 후 종료
    인자가 2개 이상일 경우에는 인자 많은 에러 출력 후 종료
    포트번호가 숫자가 아닐경우 에러 출력 후 종료

    소켓 생성
    sin_family, addr, port 세팅
    바인딩 후 listen으로 queue생성

    while(1){
        accept함수로 서버와 연결
        로그를 찍기 위해 전역변수 세팅 후 port와 ip를 저장
        pid = fork();
        if(자식 프로세스일 경우){
            while(1){
                버퍼 초기화
                클라이언트에서 명령어 read -> 에러시 에러 출력 후 종료

                if(버퍼가 port일 경우){
                    convert_str_to_addr 함수로 host ip설정
                    데이터 소켓 생성-> 실패시 에러 출력 후 종료

                    데이터 소켓 sin_family,addr, port 세팅
                    클라이언트에게 connect함수로 연결 요청

                    200 port 커맨드 메시지 출력 후 클라이언트에게 전달
                }
                else if(버퍼가 NLST or STOR or RETR일 경우){
                    150 성공 메시지 출력후 클라이언트에게 전달
                    로그 작성

                    if(버퍼가 STOR일 경우){
                        TYPE이 A면 150 메시지를 아스키 모드 형태로 전달
                        TYPE이 I면 150 메시지를 바이너리 모드 형태로 전달
                        해당 로그 작성
                        파일내용을 소켓을 통해 받아서 STORcmd()에 전달
                    }
                }
            }
        }
    }
}

```

만약 실패시 550 에러 메시지를 클라이언트에게 전달
전송을 성공했다면 226 성공 메시지를 전달

```
}  
if(버퍼가 RETR일 경우){  
    buff를 통해 파일 이름 추출  
    유효하지 않다면 550 에러 메시지 저장  
  
    타입이 A라면 150 아스키 형태 메시지 저장후 로그작성  
    타입이 I라면 150 바이너리 형태 메시지 저장후 로그작성  
}  
해당 메시지 출력 후 클라이언트에게 전송
```

```
}
```

```
}
```

cmd_process함수를 호출하여 result_buff세팅
해당 데이터 클라이언트에게 전달

```
if(버퍼가 RETR이면){  
    OK메시지 바이트와 함께 출력후 로그작성  
    메시지 클라이언트에게 전달  
}
```

```
if(버퍼가 QUIT이면){  
    221 Goodbye메시지 클라이언트에게 전달  
    자식 프로세스 종료  
}
```

```
else{  
    cmd_process함수를 호출하여 result_buff세팅  
    결과를 클라이언트에게 전달  
}
```

```
}
```

```
else{  
    //부모 프로세스  
}
```

close 소켓

return

```
}
```


결과화면

ls 명령어

```
kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ./cli 127.0.0.1 2223
ftp> ls
200 Port command successful
150 Opening data connection for directory list.
cli
cli.c
d3/
de/
exam1
logfile
makefile
nono/
srv
srv.c
test/
test1
test2
test3
226 Complete transmission.
OK. 81 bytes is received.
ftp>

kw2020202060@ubuntu: ~/Downloads/Assignment3-3
kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ./srv 2223
200 Port command successful
150 Opening data connection for directory list.
NLST
226 Complete transmission.
```

위 결과를 확인하면 ls 명령어에 경우 port 커맨드가 전송되어 연결된 것을 확인할 수 있다. 또한 ls 결과가 올바르게 수행된 것을 볼 수 있다.

```

kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ./cli 127.0.0.1 2224
ftp> ls -a
200 Port command successful
150 Opening data connection for directory list.
./
../
cli
cli.c
d3/
de/
exam1
logfile
makefile
nono/
srv
srv.c
test/
test1
test2
test3
226 Complete transmission.
OK. 88 bytes is received.
ftp>

```

```

kw2020202060@ubuntu: ~/Downloads/Assignment3-3
kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ./srv 2224
200 Port command successful
150 Opening data connection for directory list.
NLST -a
226 Complete transmission.

```

Ls -a 명령어는 숨김 파일까지 출력되는 것을 확인할 수 있다.

```

kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ./srv 2225
200 Port command successful
150 Opening data connection for directory list.
NLST -l
226 Complete transmission.
200 Port command successful
150 Opening data connection for directory list.
NLST -al
226 Complete transmission.

```

```

kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ./cli 127.0.0.1 2225
ftp> ls -l
200 Port command successful
150 Opening data connection for directory list.
-rwxrwxr-x 1 kw2020202060 kw2020202060 26816 May 31 20:55 cli
-rw-rw-r-- 1 kw2020202060 kw2020202060 21618 May 31 11:29 cli.c
drwxrwxr-x 2 kw2020202060 kw2020202060 4096 May 28 21:12 d3/
drwxrwxr-x 2 kw2020202060 kw2020202060 4096 May 31 00:00 de/
-rw-rw-r-- 1 kw2020202060 kw2020202060 66 May 31 10:55 exam1
-rw-rw-r-- 1 kw2020202060 kw2020202060 37774 May 31 22:54 logfile
-rw-rw-r-- 1 kw2020202060 kw2020202060 74 May 28 19:58 makefile
d----- 2 kw2020202060 kw2020202060 4096 May 30 23:38 nono/
-rwxrwxr-x 1 kw2020202060 kw2020202060 57456 May 31 11:23 srv
-rw-rw-r-- 1 kw2020202060 kw2020202060 63663 May 31 11:23 srv.c
drwxrwxr-x 2 kw2020202060 kw2020202060 4096 May 31 22:49 test/
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 31 22:49 test1
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 31 22:49 test2
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 31 22:49 test3
226 Complete transmission.
OK. 871 bytes is received.

```

```

OK. 871 bytes is received.
ftp> ls -al
200 Port command successful
150 Opening data connection for directory list.
drwxrwxr-x 6 kw2020202060 kw2020202060 4096 May 31 22:49 ./
drwxr-xr-x 10 kw2020202060 kw2020202060 4096 May 31 01:18 ../
-rwxrwxr-x 1 kw2020202060 kw2020202060 26816 May 31 20:55 cli
-rw-rw-r-- 1 kw2020202060 kw2020202060 21618 May 31 11:29 cli.c
drwxrwxr-x 2 kw2020202060 kw2020202060 4096 May 28 21:12 d3/
drwxrwxr-x 2 kw2020202060 kw2020202060 4096 May 31 00:00 de/
-rw-rw-r-- 1 kw2020202060 kw2020202060 66 May 31 10:55 exam1
-rw-rw-r-- 1 kw2020202060 kw2020202060 38286 May 31 22:54 logfile
-rw-rw-r-- 1 kw2020202060 kw2020202060 74 May 28 19:58 makefile
d----- 2 kw2020202060 kw2020202060 4096 May 30 23:38 nono/
-rwxrwxr-x 1 kw2020202060 kw2020202060 57456 May 31 11:23 srv
-rw-rw-r-- 1 kw2020202060 kw2020202060 63663 May 31 11:23 srv.c
drwxrwxr-x 2 kw2020202060 kw2020202060 4096 May 31 22:49 test/
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 31 22:49 test1
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 31 22:49 test2
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 31 22:49 test3
226 Complete transmission.
OK. 993 bytes is received.

```

Ls -l 과 ls -al 에 대해서도 모두 올바르게 수행되는 것을 볼 수 있다.

dir 명령어

```
OK: 993 bytes received.  
ftp> dir  
drwxrwxr-x 6 kw2020202060 kw2020202060 4096 May 31 22:49 ./  
drwxr-xr-x 10 kw2020202060 kw2020202060 4096 May 31 01:18 ../  
-rwxrwxr-x 1 kw2020202060 kw2020202060 26816 May 31 20:55 cli  
-rw-rw-r-- 1 kw2020202060 kw2020202060 21618 May 31 11:29 cli.c  
drwxrwxr-x 2 kw2020202060 kw2020202060 4096 May 28 21:12 d3/  
drwxrwxr-x 2 kw2020202060 kw2020202060 4096 May 31 00:00 de/  
-rw-rw-r-- 1 kw2020202060 kw2020202060 66 May 31 10:55 exam1  
-rw-rw-r-- 1 kw2020202060 kw2020202060 38525 May 31 22:56 logfile  
-rw-rw-r-- 1 kw2020202060 kw2020202060 74 May 28 19:58 makefile  
d----- 2 kw2020202060 kw2020202060 4096 May 30 23:38 nono/  
-rwxrwxr-x 1 kw2020202060 kw2020202060 57456 May 31 11:23 srv  
-rw-rw-r-- 1 kw2020202060 kw2020202060 63663 May 31 11:23 srv.c  
drwxrwxr-x 2 kw2020202060 kw2020202060 4096 May 31 22:49 test/  
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 31 22:49 test1  
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 31 22:49 test2  
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 31 22:49 test3  
ftp>
```

Dir 명령어는 ls -al 과 동일한 결과가 나타남을 볼 수 있다.

MKDIR RMDIR

```
kw2020202060@ubuntu: ~/Downloads/Assignment3-3  
kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ls  
cli d3 exam1 makefile srv test test2  
cli.c de logfile nono srv.c test1 test3  
kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ./cli 127.0.0.1 2226  
ftp> mkdir 1 2 3  
250 MKD command performed successfully.  
250 MKD command performed successfully.  
250 MKD command performed successfully.  
ftp> rmdir 1 2 3 4  
250 RMD command performed successfully.  
250 RMD command performed successfully.  
250 RMD command performed successfully.  
550: failed to remove 4  
ftp> mkdir 1 2  
250 MKD command performed successfully.  
250 MKD command performed successfully.  
ftp> ^C  
kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ls  
1 cli d3 exam1 makefile srv test test2  
2 cli.c de logfile nono srv.c test1 test3  
kw2020202060@ubuntu:~/Downloads/Assignment3-3$
```

위 결과를 확인하면 mkdir rmdir 명령어가 ls 명령어를 통해서 모두 올바르게 삭제와 생성됐음을 볼 수 있다.

DELETE

```
kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ls
cli  d3  exam1  makefile  srv  test  test2
cli.c de  logfile nono      srv.c test1 test3
kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ./cli 127.0.0.1 2226
ftp> delete test1 test2 test3 test4
250 DELE command performed successfully.
250 DELE command performed successfully.
250 DELE command performed successfully.
550: No such file or directory
ftp> ^C
kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ls
cli cli.c d3 de exam1 logfile makefile nono  srv  srv.c  test
kw2020202060@ubuntu:~/Downloads/Assignment3-3$
```

위 결과를 확인하면 DELETE 명령어를 통해서 원하고자 하는 파일이 삭제됨을 볼 수 있고 없는 파일일 경우에는 실패가 발생하는 것을 확인할 수 있다.

RENAME

```
ftp> ^C
kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ls
cli cli.c d3 de exam1 logfile makefile nono  srv  srv.c  test
kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ./cli 127.0.0.1 2226
ftp> rename d3 d1
350 File exists, ready to rename
250 RNT0 command performed successfully.
ftp> ^C
kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ls
cli cli.c d1 de exam1 logfile makefile nono  srv  srv.c  test
kw2020202060@ubuntu:~/Downloads/Assignment3-3$
```

Rename 명령어를 통해서 d3 이 d1 로 이름이 변경된 형태를 볼 수 있다.

CD CD.. PWD

```
kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ./cli 127.0.0.1 2226
ftp> pwd
257 /home/kw2020202060/Downloads/Assignment3-3 is current directory
ftp> cd ./d1
250 CWD command performed successfully.
ftp> pwd
257 /home/kw2020202060/Downloads/Assignment3-3/d1 is current directory
ftp> cd ..
250 CDUP command performed successfully.
ftp> pwd
257 /home/kw2020202060/Downloads/Assignment3-3 is current directory
ftp>
```

위를 확인하면 CWD, CDUP 명령어가 PWD 명령어에 의해서 모두 정상적으로 수행됨을 볼 수 있다.

Bin, ascii, type binary, type ascii

```
kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ftp
ftp> bin
201 Type set to I.
ftp> ascii
201 Type set to A.
ftp> type binary
201 Type set to I.
ftp> type ascii
201 Type set to A.
ftp>
```

위를 확인하면 해당 명령어들에 따라서 TYPE 이 바이너리 형태인지, 아스키 형태인지 바뀌는 것을 확인할 수 있다.

Get Put

이 명령어들은 정밀한 검증을 하기 위해서 서로 다른 경로에 대해 테스트하였다

```
kw2020202060@ubuntu: ~/Downloads/Assignment3-3/test
kw2020202060@ubuntu:~/Downloads/Assignment3-3$ cd test
kw2020202060@ubuntu:~/Downloads/Assignment3-3/test$ ls
cli cli.c makefile sendtt SUDOcli.c SUDOSrv.c
kw2020202060@ubuntu:~/Downloads/Assignment3-3/test$ ./cli 127.0.0.1 2227
ftp> get exam1
200 Port command successful
150 Opening BINARY mode data connection for exam1.
226 Complete transmission.
OK. 66 bytes is received
ftp> ^C
kw2020202060@ubuntu:~/Downloads/Assignment3-3/test$ ls
cli cli.c exam1 makefile sendtt SUDOcli.c SUDOSrv.c
kw2020202060@ubuntu:~/Downloads/Assignment3-3/test$

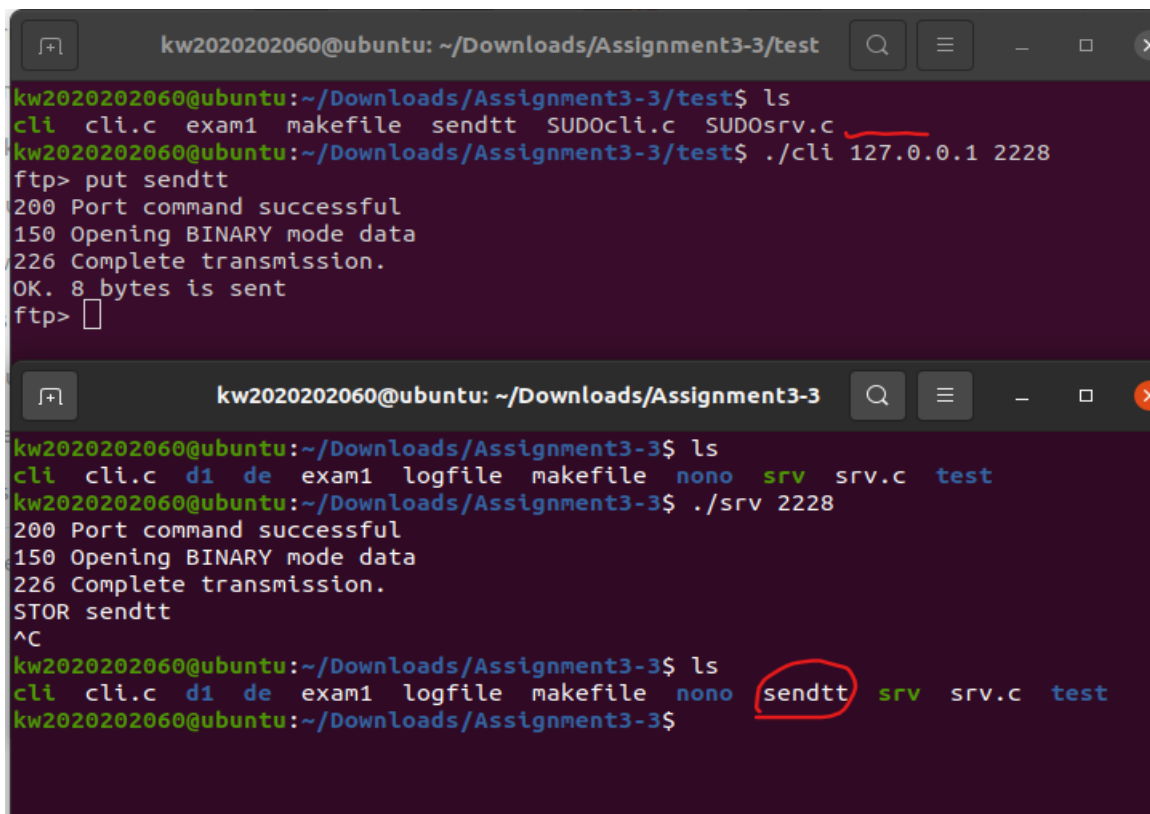
kw2020202060@ubuntu: ~/Downloads/Assignment3-3
kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ls
cli cli.c d1 de exam1 logfile makefile nono srv srv.c test
kw2020202060@ubuntu:~/Downloads/Assignment3-3$ ./srv 2227
200 Port command successful
150 Opening BINARY mode data connection for exam1.
RETR exam1
226 Complete transmission.
OK. 66 bytes is received
read error or client closed connection: Bad address
```

위는 서로 다른 경로인 것을 볼 수 있지만 클라이언트가 get exam1 을 입력하자 해당 파일을 생성하여 클라이언트 경로에 생긴 것을 볼 수 있다.

```
1 sadasdasdsadsadasdasdasdass
2 asdasdasdsad
3 asdasdasdasd
4 asdsadsadasd
```

해당 내용까지 정확하게 복사된 것을 볼 수 있다.

다음은 put 명령어이다.



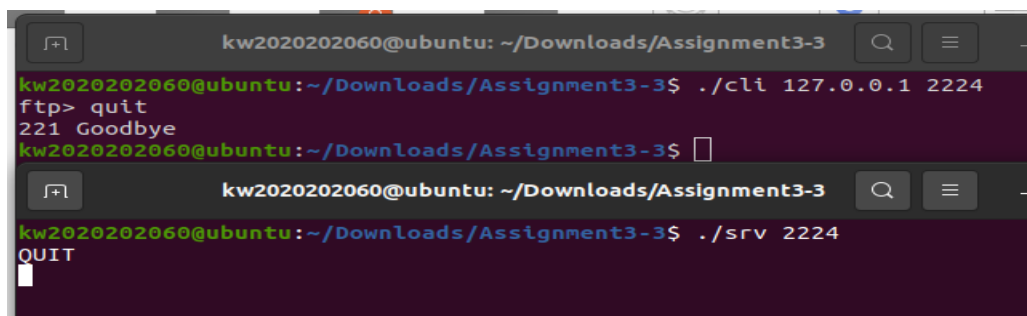
The first terminal window shows the user in the directory ~/Downloads/Assignment3-3/test. They run 'ls' and see files: cli, cli.c, exam1, makefile, sendtt, SUDOcli.c, and SUDOSrv.c. Then they run './cli 127.0.0.1 2228' and enter an FTP session. They type 'put sendtt' and receive a success message: '200 Port command successful', '150 Opening BINARY mode data', '226 Complete transmission.', 'OK. 8 bytes is sent'.

The second terminal window shows the user in the directory ~/Downloads/Assignment3-3. They run 'ls' and see files: cli, cli.c, d1, de, exam1, logfile, makefile, nono, srv, srv.c, and test. They run './srv 2228' and enter an FTP session. They type 'STOR sendtt' and receive a success message: '200 Port command successful', '150 Opening BINARY mode data', '226 Complete transmission.', 'STOR sendtt', '^C'. Then they run 'ls' and see the same files as before, but 'sendtt' is now circled in red, indicating it has been successfully transferred.

```
1 hi
2 hello
```

위를 확인하면 서버 경로에 없었던 sendtt 파일이 클라이언트를 통해 전달받고 생성이 되었다.

QUIT



The first terminal window shows the user in the directory ~/Downloads/Assignment3-3. They run './cli 127.0.0.1 2224' and enter an FTP session. They type 'quit' and receive a success message: '221 Goodbye'.

The second terminal window shows the user in the directory ~/Downloads/Assignment3-3. They run './srv 2224' and enter an FTP session. They type 'QUIT' and receive a success message: '221 Goodbye'.

Quit 를 입력하자 종료 메시지와 함께 클라이언트는 종료되는 것을 볼 수 있다.

LOGfile

2024-05-31	22:53:31	[127.0.0.1:44926]	kw2020202060	221 Goodbye
2024-05-31	22:51:22	[127.0.0.1:54746]	kw2020202060	Server is started
2024-05-31	22:51:23	[127.0.0.1:54746]	kw2020202060	PORT 127,0,0,1,175,233
2024-05-31	22:51:23	[127.0.0.1:54746]	kw2020202060	200 Port command successful
2024-05-31	22:51:23	[127.0.0.1:54746]	kw2020202060	150 Opening data connection for directory list.
2024-05-31	22:51:23	[127.0.0.1:54746]	kw2020202060	received: NLST
2024-05-31	22:51:23	[127.0.0.1:54746]	kw2020202060	226 Complete transmission.
2024-05-31	22:51:23	[127.0.0.1:54746]	kw2020202060	OK. 81 bytes is received
2024-05-31	22:53:01	[127.0.0.1:35880]	kw2020202060	Server is started
2024-05-31	22:53:08	[127.0.0.1:35880]	kw2020202060	PORT 127,0,0,1,192,108
2024-05-31	22:53:08	[127.0.0.1:35880]	kw2020202060	200 Port command successful
2024-05-31	22:53:08	[127.0.0.1:35880]	kw2020202060	150 Opening data connection for directory list.
2024-05-31	22:53:08	[127.0.0.1:35880]	kw2020202060	received: NLST -a
2024-05-31	22:53:08	[127.0.0.1:35880]	kw2020202060	226 Complete transmission.
2024-05-31	22:53:08	[127.0.0.1:35880]	kw2020202060	OK. 88 bytes is received
2024-05-31	22:54:13	[127.0.0.1:58442]	kw2020202060	Server is started
2024-05-31	22:54:15	[127.0.0.1:58442]	kw2020202060	PORT 127,0,0,1,127,196
2024-05-31	22:54:15	[127.0.0.1:58442]	kw2020202060	200 Port command successful
2024-05-31	22:54:15	[127.0.0.1:58442]	kw2020202060	150 Opening data connection for directory list.
2024-05-31	22:54:15	[127.0.0.1:58442]	kw2020202060	received: NLST -l
2024-05-31	22:54:15	[127.0.0.1:58442]	kw2020202060	226 Complete transmission.
2024-05-31	22:54:15	[127.0.0.1:58442]	kw2020202060	OK. 871 bytes is received
2024-05-31	22:54:17	[127.0.0.1:58442]	kw2020202060	PORT 127,0,0,1,45,112
2024-05-31	22:54:17	[127.0.0.1:58442]	kw2020202060	200 Port command successful
2024-05-31	22:54:17	[127.0.0.1:58442]	kw2020202060	150 Opening data connection for directory list.
2024-05-31	22:54:17	[127.0.0.1:58442]	kw2020202060	received: NLST -al
2024-05-31	22:54:17	[127.0.0.1:58442]	kw2020202060	226 Complete transmission.
2024-05-31	22:54:17	[127.0.0.1:58442]	kw2020202060	OK. 993 bytes is received

위를 확인한다면 어떠한 명령을 할 때마다 해당 내용에 대한 기록이 되는 것을 볼 수 있다.

고찰

한 학기 동안 최종적으로 FTP 서버를 구현하였다. 비록 이전에 계획했던 로그인 기능은 시간상 완성하지 못했지만, 추후 시간이 있다면 반드시 이 부분을 완성하고 싶다는 생각을 하였다. 프로젝트를 진행하면서 가장 큰 도전은 오류 처리를 하는 것이었다. 프로젝트의 규모가 크기 때문에 오류가 발생하면 이를 찾아내고 수정하는 것이 매우 어려웠다. 이러한 어려움 속에서 여러 가지 문제를 해결하기 위해 많은 노력을 기울였고, 이 과정에서 많은 것을 배울 수 있었다.

프로젝트 초반에는 FTP 서버의 기본 구조를 이해하고, 소켓 프로그래밍을 통해 클라이언트와 서버 간의 통신을 설정하는 것부터 시작하였다. 소켓을 사용하여 프로토콜 연결을 설정하고, 클라이언트로부터 명령어를 수신하여 서버가 이에 맞는 응답을 반환하는 과정을 구현하는 것이 주요 과제였다. 이 과정에서 발생하는 작은 오류 하나가 전체 통신을 방해할 수 있기 때문에, 코드를 작성할 때 매우 신중해야 했다.

특히, 오류 처리는 예상치 못한 상황에서 프로그램이 어떻게 반응하는지를 결정하는 중요한 부분이었다. 예를 들어, 클라이언트가 잘못된 명령어를 입력했을 때, 서버가 적절한 오류 메시지를 반환하고 연결을 유지하는 것이 중요했다. 이를 위해 다양한 예외 상황을 처리할 수 있는 코드를 작성하고, 여러 번의 테스트를 통해 이러한 상황을 시뮬레이션하였다.

프로젝트 중반에는 파일 전송 기능을 구현하면서 더 복잡한 오류가 발생하기 시작했다. 파일을 업로드하거나 다운로드하는 과정에서 데이터의 손실이나 변형을 방지하기 위해 정확한 데이터 전송을 보장해야 했다. 여기서 네트워크의 불안정성이나 파일 크기 등의 문제로 인해 예기치 못한 오류가 자주 발생하였다. 이러한 문제를 해결하기 위해 데이터 무결성을 확인하는 방법을 도입하였다.

프로젝트 후반에는 로그 파일을 통해 서버의 활동을 기록하는 기능을 구현하였다. 이를 통해 서버의 상태를 모니터링하고, 발생한 오류를 추적하여 원인을 분석하는 데 큰 도움이 되었다. 로그 파일을 분석하면서 발생한 오류의 패턴을 파악하고, 이를 바탕으로 코드를 개선하는 과정을 반복하였다. 이 과정에서 디버깅 기술과 문제 해결 능력을 크게 향상시킬 수 있었다.

비록 로그인 기능을 완성하지 못했지만, 이 프로젝트를 통해 얻은 경험은 매우 값 졌다. 특히, 대규모 프로젝트에서 오류를 처리하고 해결하는 능력은 앞으로의 개발 과정에서도 큰 자산이 될 것이다. 추후 시간적 여유가 생긴다면, 미완성된 로그인 기능을 보완하고, 전체적인 코드의 안정성을 더욱 강화하고자 한다. 이 프로젝트를 통해 학습한 내용을 바탕으로, 앞으로 더욱 발전된 프로그램을 개발할 수 있을 것이라 확신한다.