시스템 프로그래밍 실습

[Assignment2-3]

Class : **[D]**

Professor : **[최상호교수님]**

Student ID : **[2020202060]**

Name : **[홍왕기]**

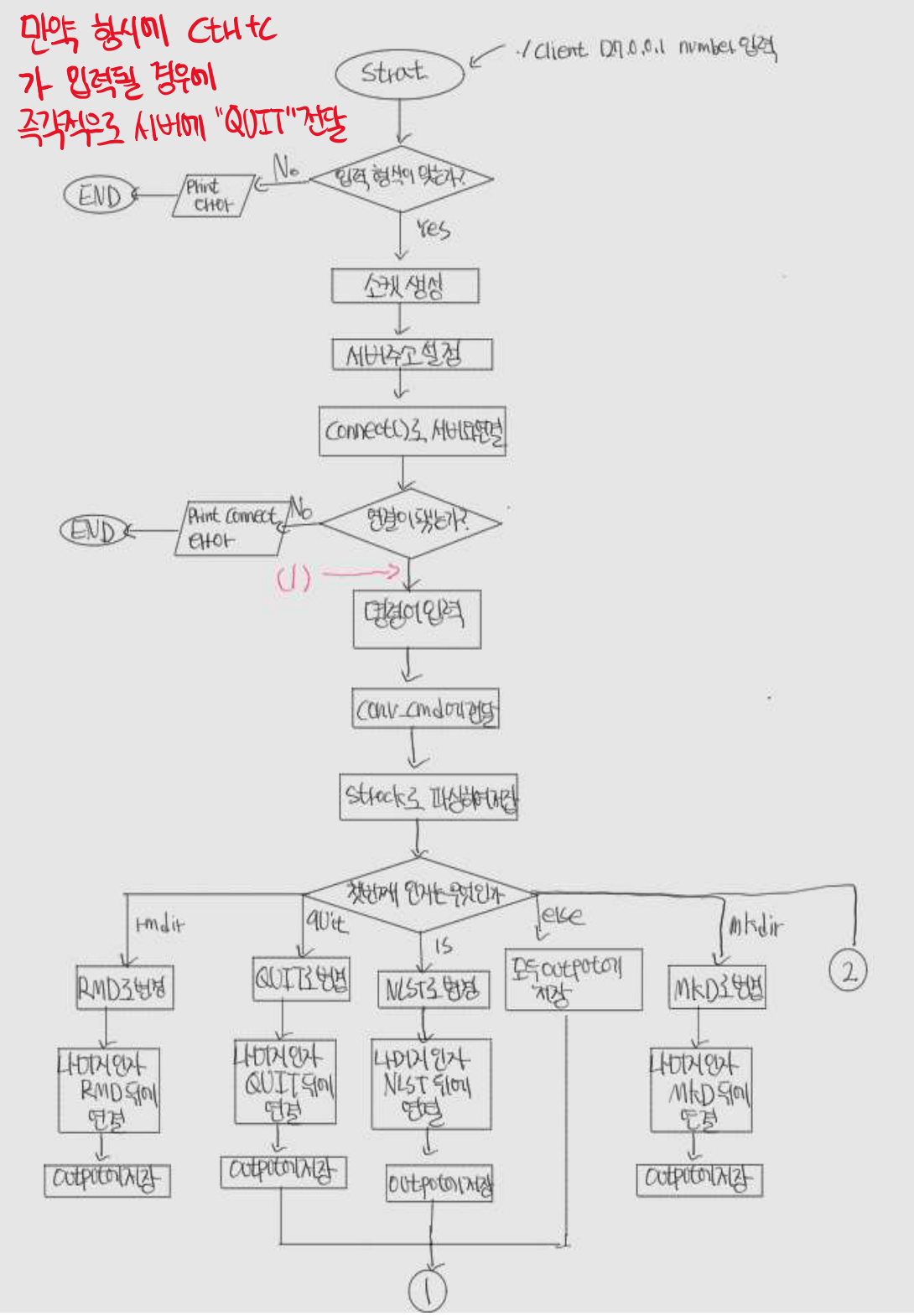
Introduction

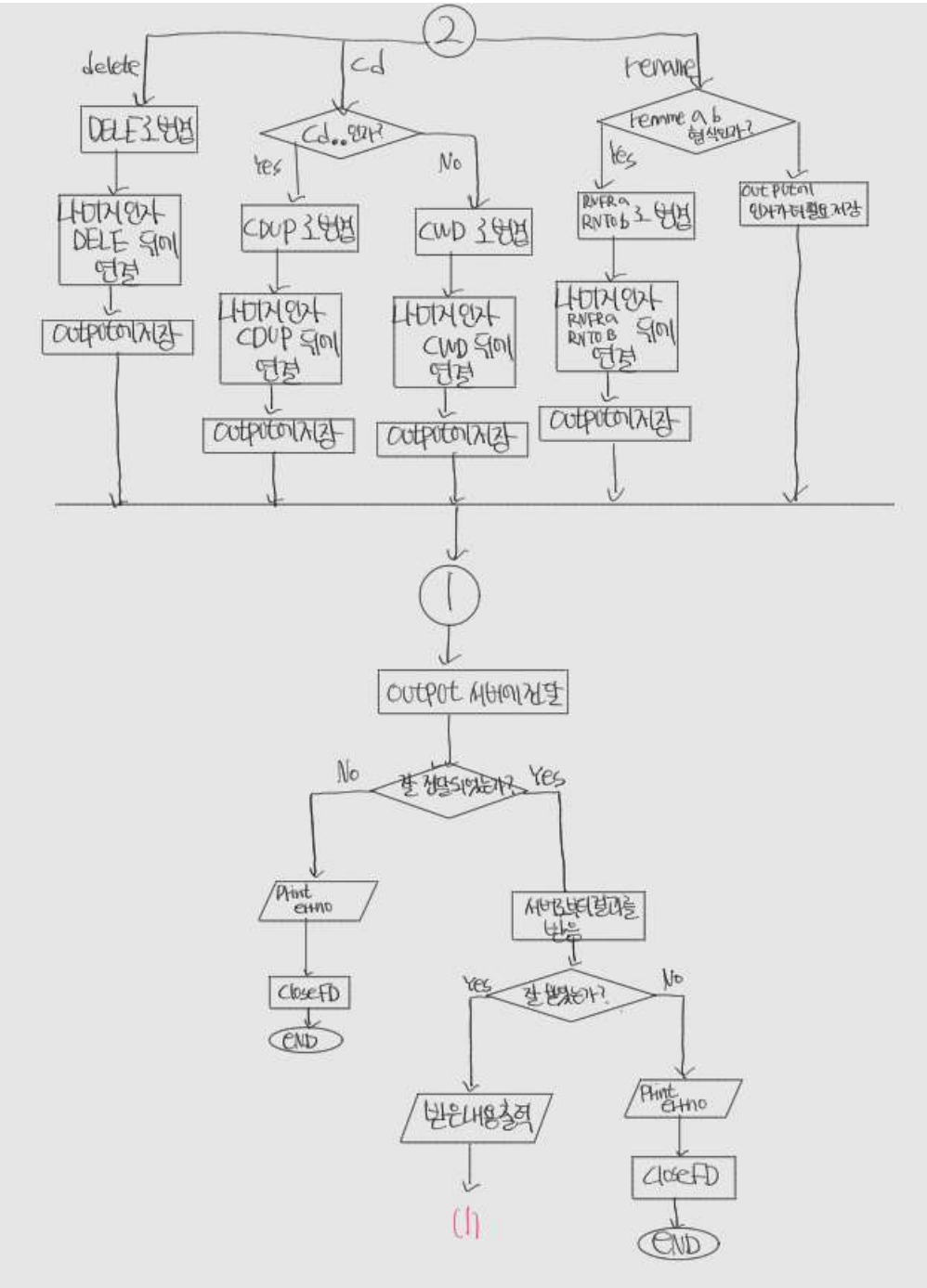
해당 프로젝트에서는 이전에 구현한 소켓 통신을 활용하여 FTP 서버 및 클라이언트를 구현한다. FTP 는 파일 전송 프로토콜로, 파일을 서버와 클라이언트 간에 전송하기 위한 표준 프로토콜이다. 이 프로젝트에서는 다중 클라이언트를 지원하기 위해 fork 함수를 사용하여 여러 요청에 대응하는 서버를 만들 것이다. fork 함수를 사용하면 각 클라이언트 요청마다 새로운 프로세스가 생성되어 해당 요청을 처리할 수 있는 장점이 있다. 이를 통해 동시에 여러 클라이언트 요청을 처리하여 효율적일 것이다.

또한, 서버는 주기적으로 alarm 함수를 통해 10 초마다 현재 연결된 클라이언트의 정보와 개수를 출력한다. 이는 서버의 상태를 모니터링하고 현재 연결된 클라이언트 수를 파악하는 데 도움이 되는 부분들이다. 클라이언트가 여러 명령어 응답을 받는 와중에 Quit 명령을 서버로 전송하면, 해당 서버는 종료되어 클라이언트는 read 를 못해 종료될 것이다. 이를 위해 시그널을 사용하여 클라이언트의 종료를 감지하고 처리해야 할 것이다. 이를 통해 클라이언트가 종료되었음을 부모 서버가 인식하고 효율적인 서버를 개발할 수 있다. 결국 이러한 기능들에 의해서 FTP 서버 및 클라이언트 간에 안정적인 통신 및 파일 전송이 가능해지는 것이다.

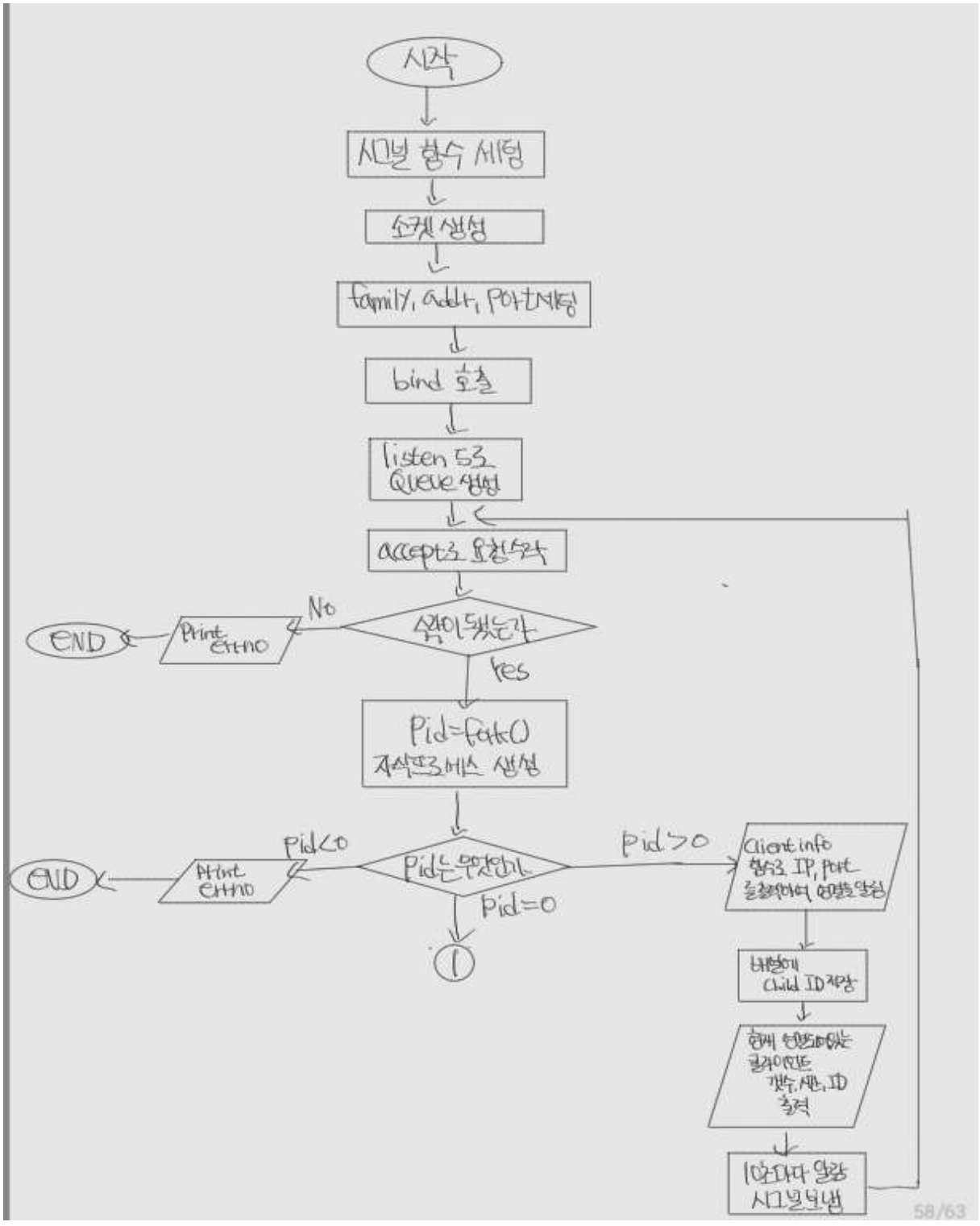
Flow chart

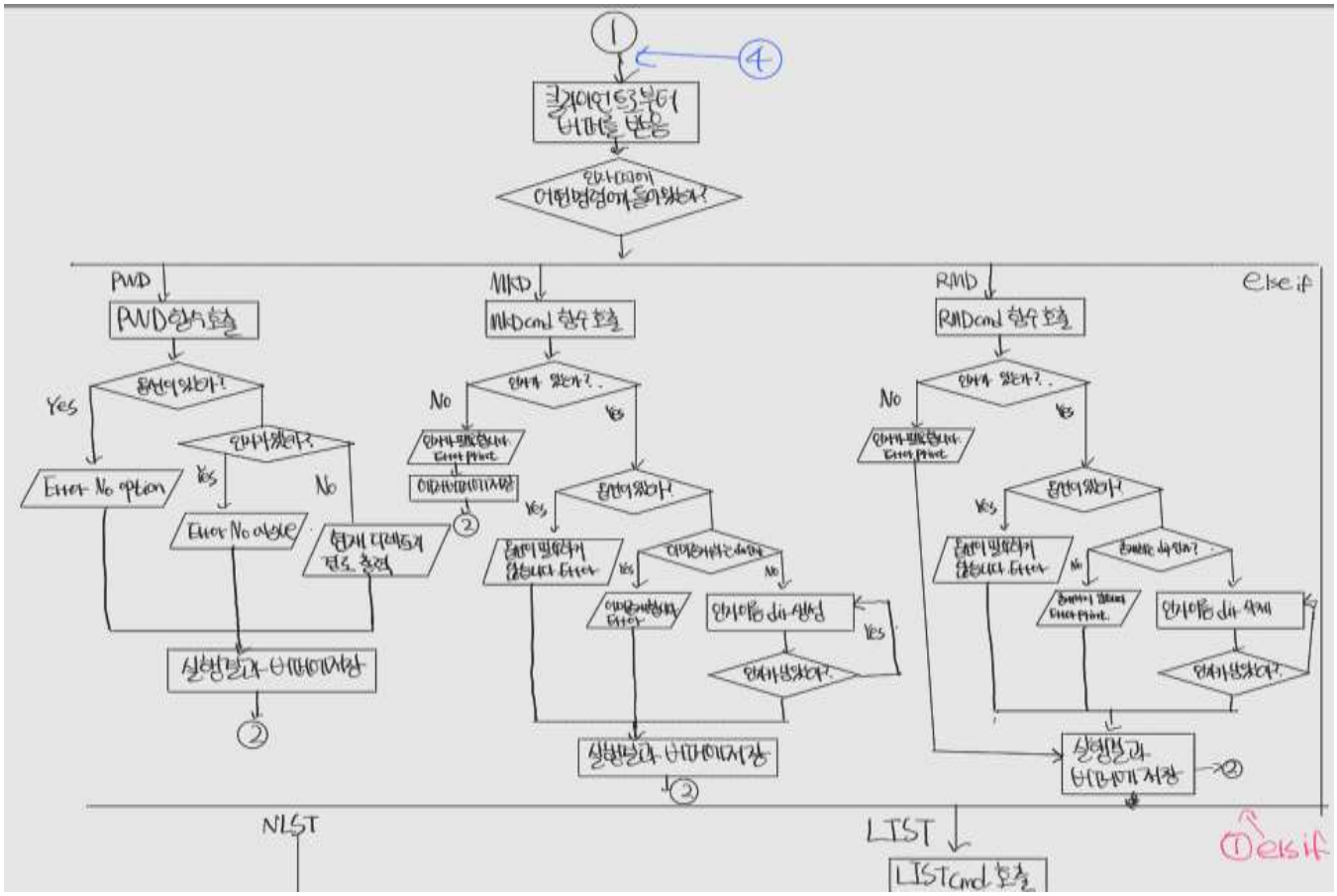
CLI.c

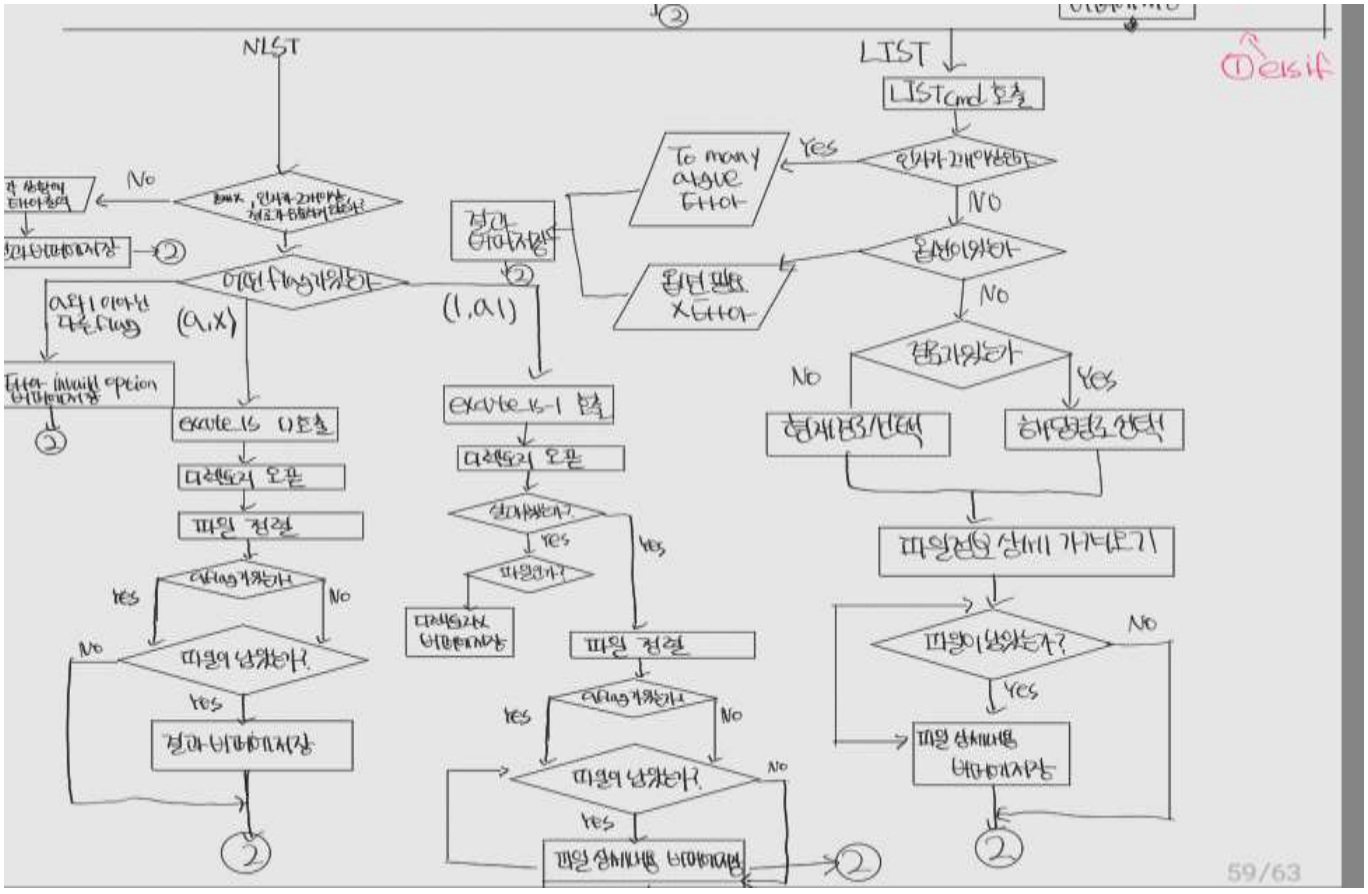


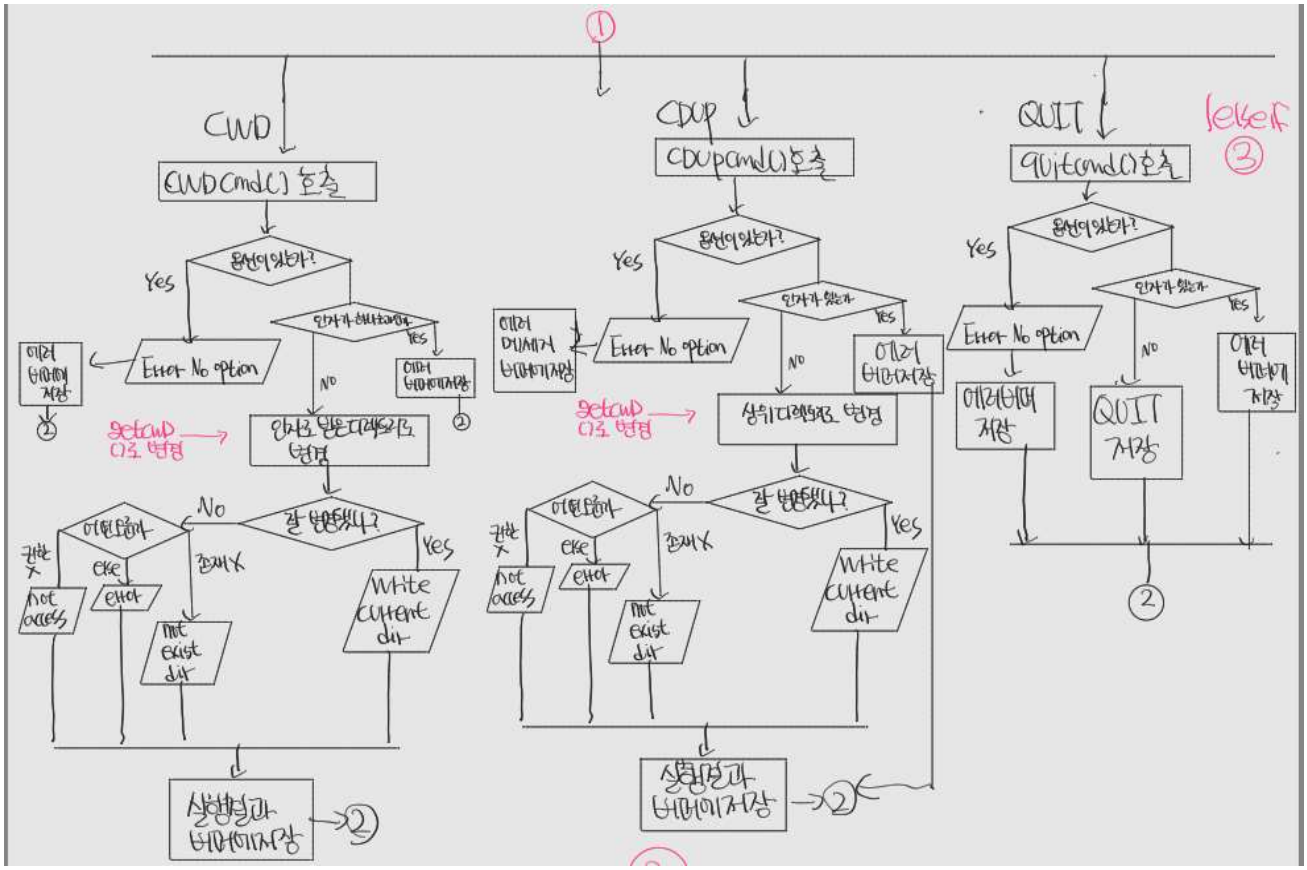

2
delete
cd
rename
DELE 프로토콜
cd.. 인가?
rename a b 형식인가?
Yes
No
Yes
CDUP 프로토콜
CWD 프로토콜
RNFR a
RNTO b 프로토콜
나머지인자 DELE 뒤에 연결
나머지인자 CDUP 뒤에 연결
나머지인자 CWD 뒤에 연결
나머지인자 RNFR a RNTO b 뒤에 연결
output이 인자가 허용됨 저장
output메시지 저장
output메시지 저장
output메시지 저장
output메시지 저장
1
output 서버에 전달
No
잘 실행되었는가?
Yes
Print errno
서버로부터 결과를 받음
closeFD
END
Yes
잘 실행되었는가?
No
받은 내용 출력
Print errno
closeFD
END
(1)

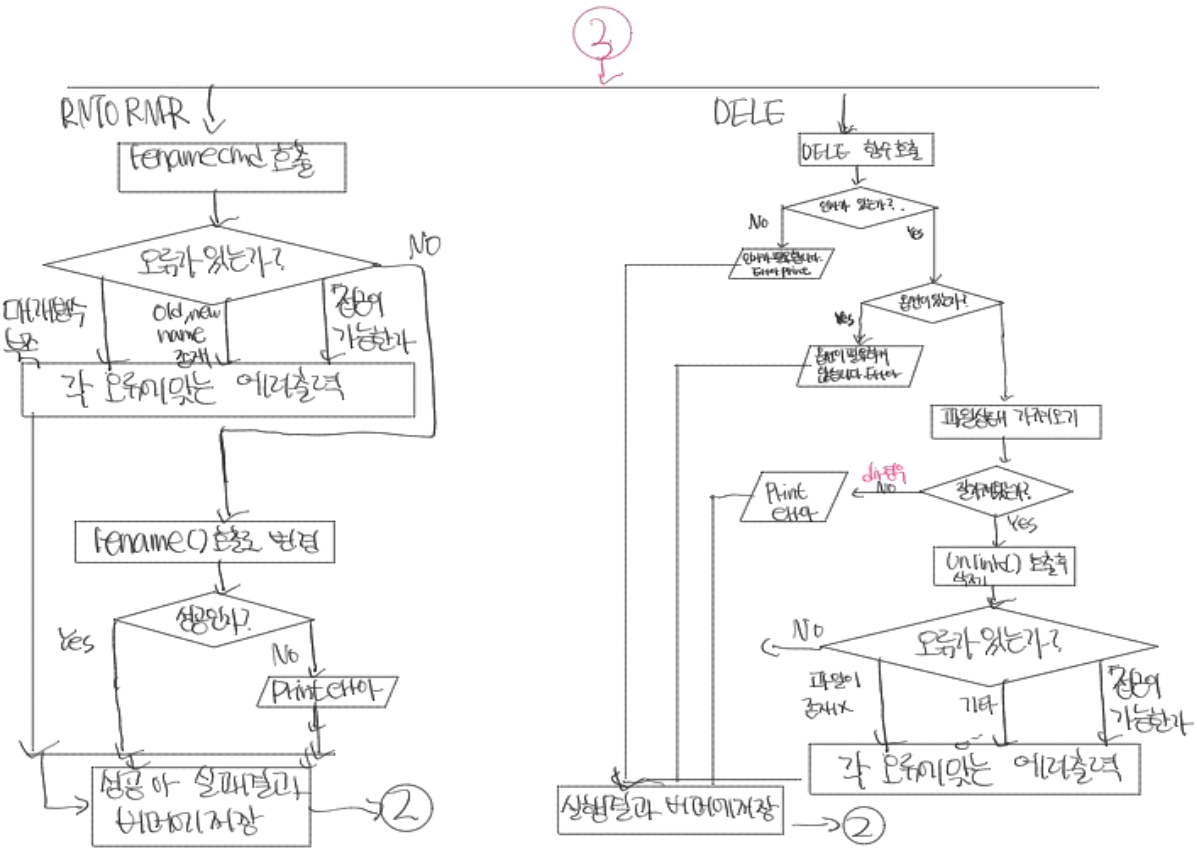
Srv.c

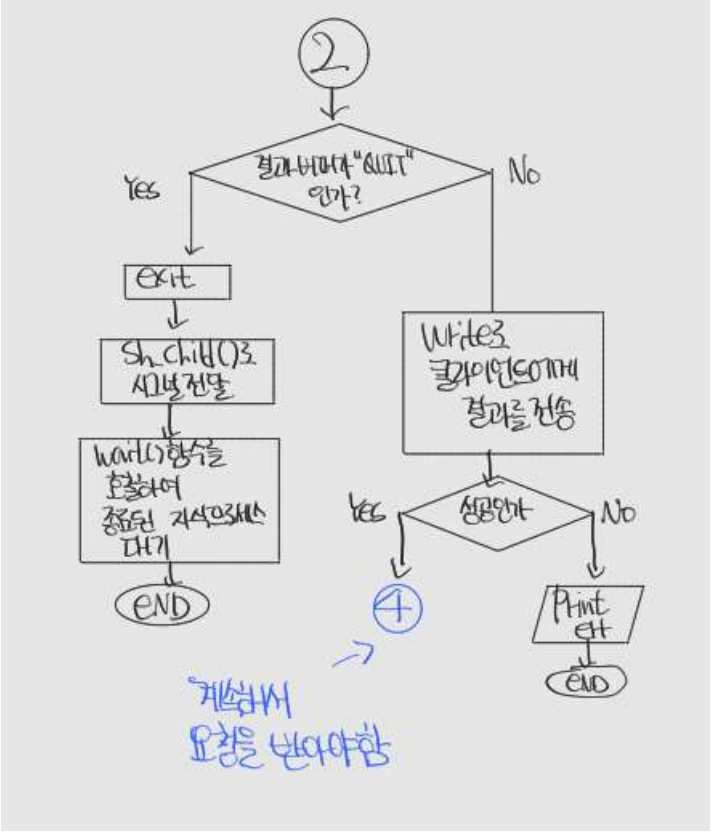

시작
시그널 함수 세팅
소켓 생성
family, addr, port 세팅
bind 호출
listen 통해 Queue 생성
accept를 통한 무한루프
성공이 됐는가
No
Print errno
END
Yes
Pid=fork()
자식프로세스 생성
Pid는 정상인가
Pid<0
Print errno
END
Pid=0
①
Pid>0
Client info 함수를 통해 IP, port 등을 얻어내고 연결을 알림
내부에 Child ID 저장
현재 연결되어있는 클라이언트 개수, 세션, ID 출력
10초마다 알람 시그널 보내기
58/63


PWD
NLST
RMD
LIST
Else if
Yes
No
Error No option
Error No value
실행결과 버퍼에저장
LISTcmd 호출
①Else if


NLST
LIST
LISTcmd 호출
Too many value Error
Error Invalid option
excute_ls 호출
디렉토리 오픈
파일 정보
디렉토리 닫기
실행결과 버퍼에저장
excute_ls-l 호출
①Else if
59/63


①
CWD
CWDCmd() 호출
옵션이있는가?
Yes
인자가 하나더있는가
Yes
No
에러 메세지 저장
Error No option
②
SetCwD 디폴 변경
인자로 받은 디렉토리로 변경
잘 변경됐나?
No
Yes
이미존재
권한 X
Error
존재X
not access
not exist dir
Write current dir
실행결과 메세지저장
CDUP
CDUPcmd() 호출
옵션이있는가?
Yes
인자가 있는가
Yes
No
에러 메세지 저장
Error No option
SetCwD 디폴 변경
상위디렉토리로 변경
잘 변경됐나?
No
Yes
이미존재
권한 X
Error
존재X
not exist dir
Write current dir
실행결과 메세지저장
②
QUIT
QUITcmd() 호출
옵션이있는가?
Yes
인자가 있는가
Yes
No
Error No option
에러메세지 저장
QUIT 저장
에러 메세지 저장
②
Select ③


③
RNTO RNFR
Renamecmd 호출
오류가 있는가?
NO
대기상태가 오면
Old,new name 존재
옵션이 가능한가
각 오류에맞는 에러출력
Rename() 호출로 변경
성공인가?
Yes
No
Print에러
성공 or 실패결과 메세지저장
②
DELE
DELE 함수 호출
인자가 있는가?
No
Yes
인자가 포함되어 Error Print
옵션이있는가?
Yes
옵션이 잘못되어 Error
파일상태 가져오기
Print 에러
실패
정상적?
No
Yes
Unlink() 호출
오류가 있는가?
No
파일이 존재X
기타
옵션이 가능한가
각 오류에맞는 에러출력
실행결과 메세지저장
②


2
"quit" 입력했는 가?
Yes
No
exit
sh_child()로 시그널 전달
wait() 함수를 호출하여 종료된 자식프로세스 대기
END
write로 클라이언트에게 결과를 전송
성공인가
Yes
No
4
Print err
END
서버에게 요청을 보내야함

Pseudo code

cli.c

```
void sh_int(int)
int sockfd

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] == '1s'){
    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] RNTO token[1]"
        }
        else output=err
    }
}

int main(int arg, char **argv){
    char buff[BUF_SIZE]
    ftp_cmd[BUF_SIZE]

    int n
    int sockfd
    struct sockaddr_in serv_addr
    sockfd = socket make this opt(AF_INET, SOCK_STREAM,0) ;

    signal(SIGINT, sh_int);

    family=AF_INET;
    addr= make binary addr // inet_addr(argv[1])
    port=host to network prot//thtons(atoi(argv[2]))
```

```
    while (1) {
        write ">"
        conv_cmd(buff, ftp_cmd)
        if(send buff to the server sucess){
            recive buff to the network
            if (recive buff success){
                print result buff
            }
            else{
                print errorno
                close sockfd
                exit
            }
        }
        else {
            print errorno
                close sockfd
                exit
        }

    }
        close socket discriptor
        return
}

void sh_int(int signum){
    if ctrl+c = sockfd=QUIT
    exit
}
```

Srv.c

```
#define MAX_BUFF 5012
#define SEND_BUFF 5012
#define BUF_SIZE 100
#define SA struct sockaddr

pid_t save_pid; // 전역 변수로 선언
int count_arguments=0;


void add_client(pid_t pid, int port, time_t connect_time) {
    child process id port 연결 시간을 받아
    그 정보를 배열에 저장.

    만약 정보를 저장하지 못했다면 에러 후 종료
}

void remove_client(pid_t pid) {
    pid로 받은 child process 정보를 배열에서 제거
    만약 오류가 해당 정보 id가 없는 경우 오류 출력


}

void print_connected_clients(){
    현재 저장되어있는 child process 정보들을 모두 출력

    "=====아래 형식과 같음===="
    Current Number of Client : 1
    PID     PORT    TIME
    118721  39954   0
}
```

```
int compare(const struct dirent **a, const struct dirent **b) {
    두 변수에 대한 이름을 비교
}

int client_info(struct sockaddr_in *client_addr) {
    ip=inet_ntoa(client_addr->sin_addr)
    prot=ntohs(client_addr->sin_port)
    write(ip and port) -> like this

    ==========Client info===========
    client IP: 127.0.0.1
    client port: ??????
    ================================

    return
}

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
}
```

```
void execute_NLST_a(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 숨김포파일을 포함하여
    해당 디렉토리를 순회하며 결과 결과 버퍼에 저장

    버퍼를 정렬
}

void execute_NLST_l(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
    해당 디렉토리를 순회하며 해당 파일 자세한 내용을
    결과 버퍼에 저장

    버퍼를 정렬
}
```

```
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 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
}
```

```
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 && RNTO != 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] == 'RNTO') go renamecmd(result_buff,args)
else if (argv[0] == 'QUIT') {옵션, 인자가 있는 경우는 에러를 버퍼에 저장,
|   |   |   |   |   |   |   |   |   |   |   그것이 아닌 경우에는 QUIT출력 }
}
```

메인 시작

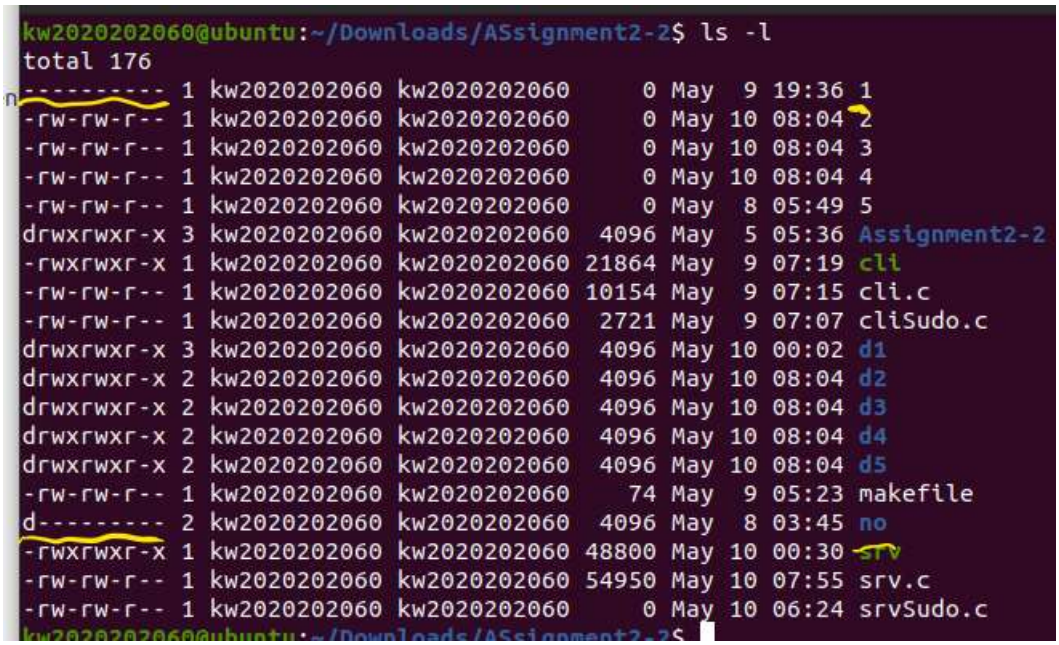
```
int main(int argc, char ** argv){

int server_fd, client_fd, n;
struct sockaddr_in server_addr, client_addr;
char buff[MAX_BUFF], result_buff[SEND_BUFF];
int len;

 if (argc == 1) {
     print "less argument"
     exit(1)
    }
 if(argc >2){
     print "too many argument"
     exit(1)
    }

    만약 포트 번호가 숫가자 아닌 경우 종료


if Call the sh_alrm function when the alarm goes off
if Call sh_alrm function at the end of the child process
if ctrl +c= call sh_int

server_fd = socket make this->(PF_INET, SOCK_STREAM, 0)

    family=AF_INET;
    s_addr=htonl(INADDR_ANY);
    port=htons(atoi(argv[1]));

    bind(setting)
    listen(server_fd, 5)->Queue make
```

a

```
while(1){

    pid_t pid;
    len=sizeof(client_addr);
    client_fd= Accepted connection request from server_fd
    if (client_fd < 0) {
        errno and exit
    }

    pid = fork();
    if (pid < 0) {
      errno and exit
    }

   if (pid == child process) {
        alarm(0)
        while (1) {
            Save to client request buff
            if (Save to client request buff fail){
                print errno and exit
            }


            if(buff="QUIT"){
                exit
            }
                cmd_process(buff, result_buff) 호출하여 명령어 수행
                child process id print
      }
      결과를 클라이언트에 전달
      실패시 에러 출력.
```

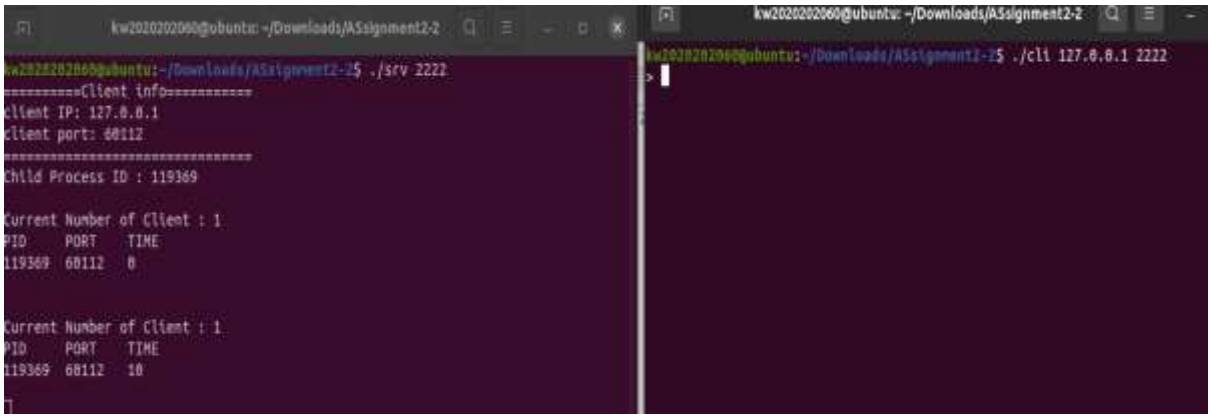
```
        }
        else {  // parent process
            client_info함수 호출
            child process id 출력
            add_client함수로 배열에 저장
            print_connect 함수 호출
            alarm(10)
        }
        close(client_fd)
    }
    close(server_fd)
    return 0
}

void sh_chld(int signum){
    wiat함수를 호출하여 종료된 자식 프로세스를 대기.
    remove_client로 제거함
}
void sh_alrm(int signum){
    print_connected_clients 함수 호출
    alarm(10)로 10초대기
}
void sh_int(int signum){
    exit(0)
}
```

결과화면

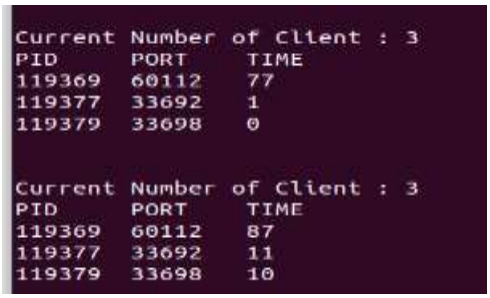


검증을 시작하기 전 위 그림은 현재 디렉토리에 대한 정보 출력이다.

먼저 이 프로젝트에서 다수에 클라이언트가 접속이 가능하기에 그 예시를 보여준다.

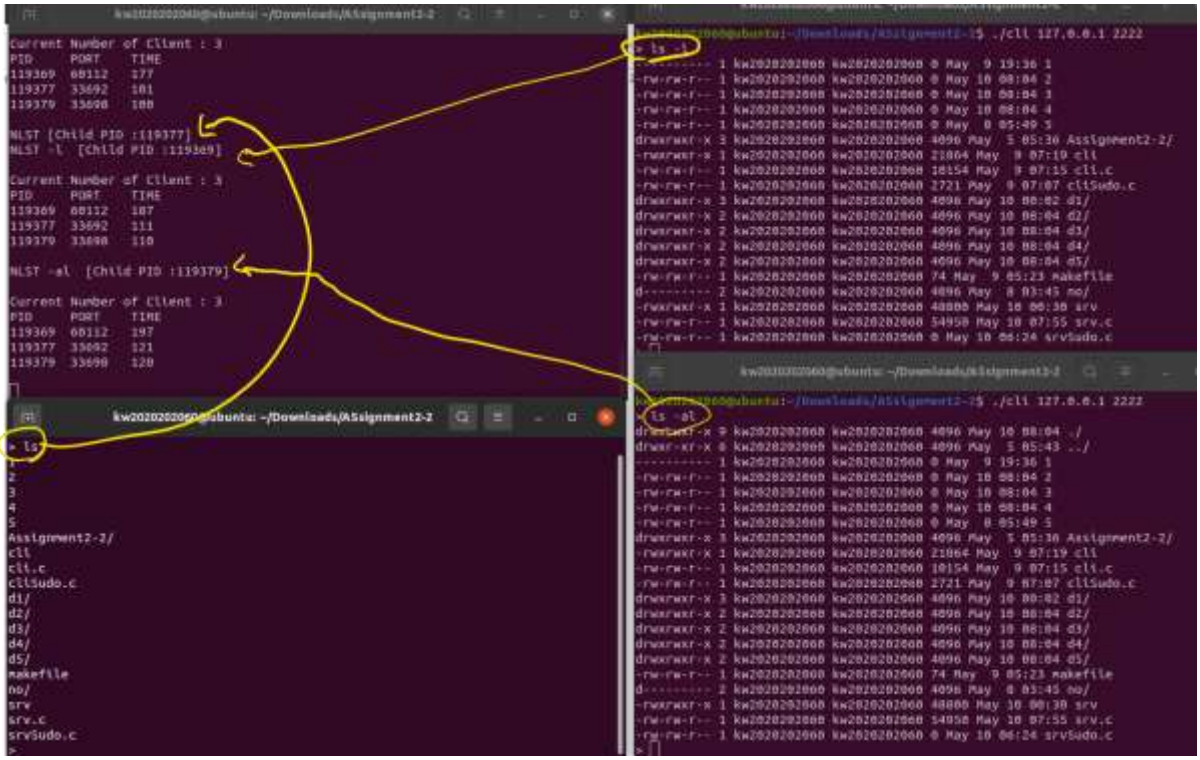


위는 한 클라이언트가 접속하였을 때 연결 형태 출력과 10 초 후에 현재 클라이언트 수와 정보가 출력되는 것을 확인할 수 있다.

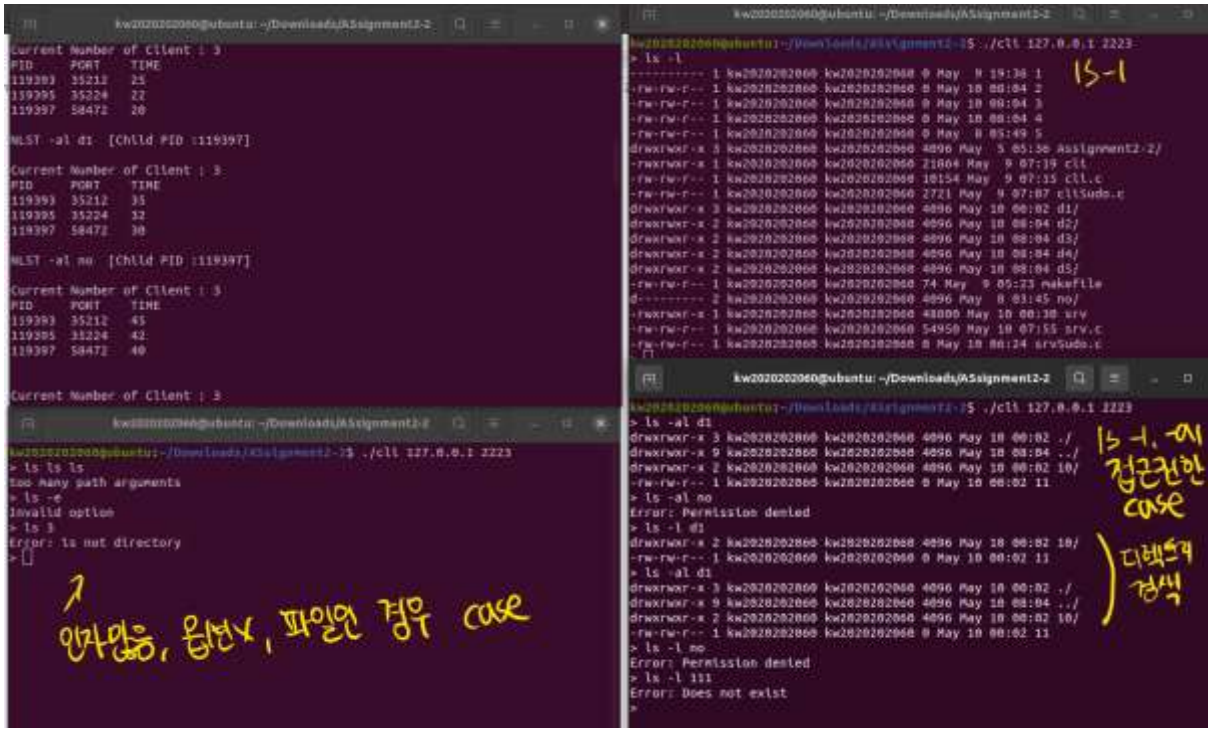


추가로 2 개의 클라이언트가 추가로 연결되었을 때 클라이언트 숫자가 증가되면서 모든 정보들이 출력되는 형태를 볼 수 있다. 이때 10 초는 지날 때마다 서버는 어떤 형태인지 알 수 있다.

다음은 **ls** 에대한 검증이다.



위 결과를 확인하면 다수의 클라이언트가 서버에 요청했을 때 해당 클라이언트 pid 를 통해 어떤 클라이언트가 명령어를 요청했는지 알 수 있게 된다. 이때 ls 명령어들은 모두 잘 작동되는 것을 확인할 수 있다. 다음은 검증하지 못한 ls -a 과 여러 예외 상황에 대한 검증이다.

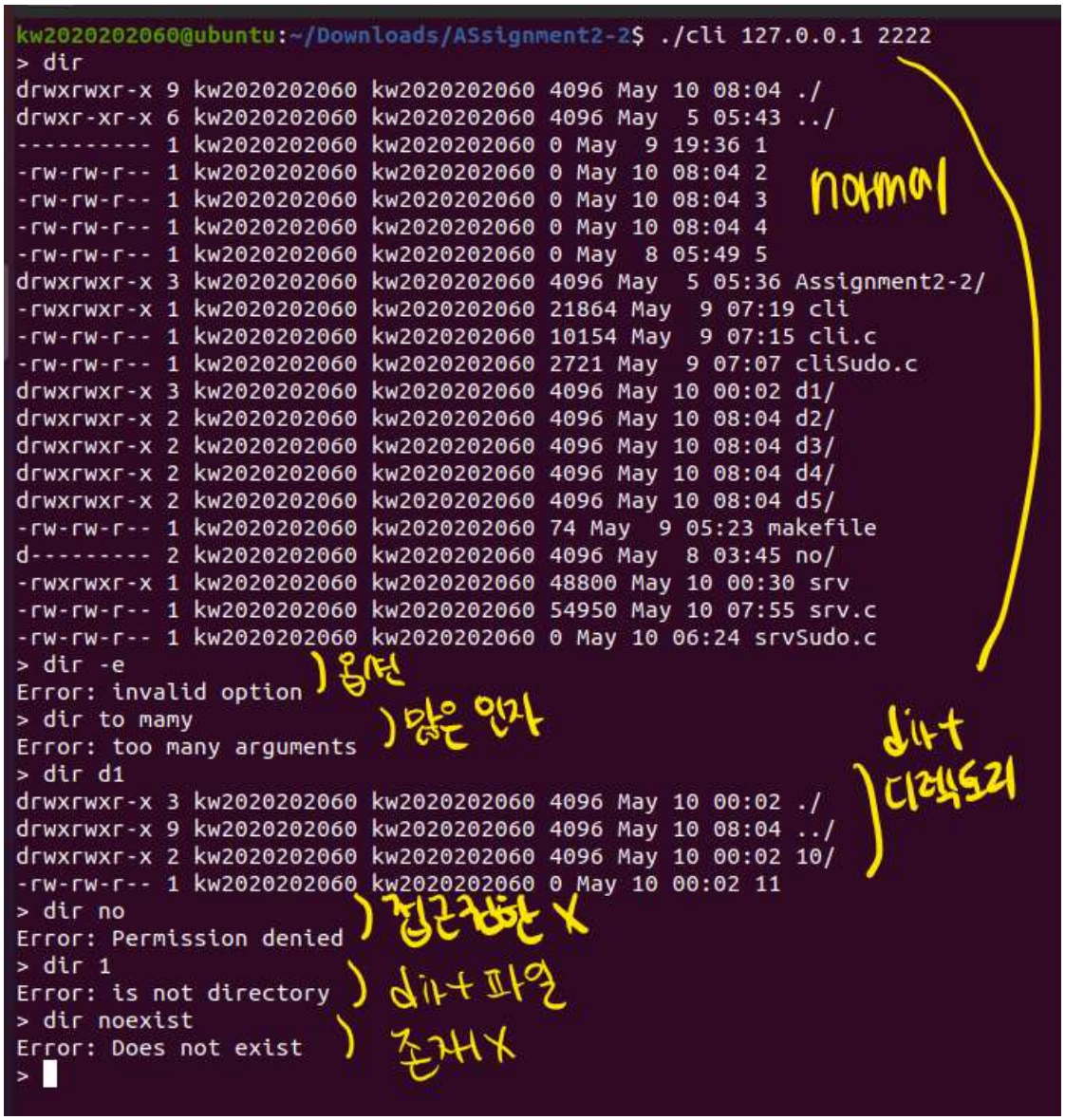


위 설명을 토대로 확인을 하면, 모든 ls 와 ls 옵션들에 대한 예외처리, 정상작동 함을 할 수 있다.

위를 통해 다수에 클라이언트가 접속하여 명령어 통해 응답을 받음을 확인했다.

나머지 명령어에 검증은 한 클라이언트로 사용하여 보기 쉽게 검증할 것이다.

LIST

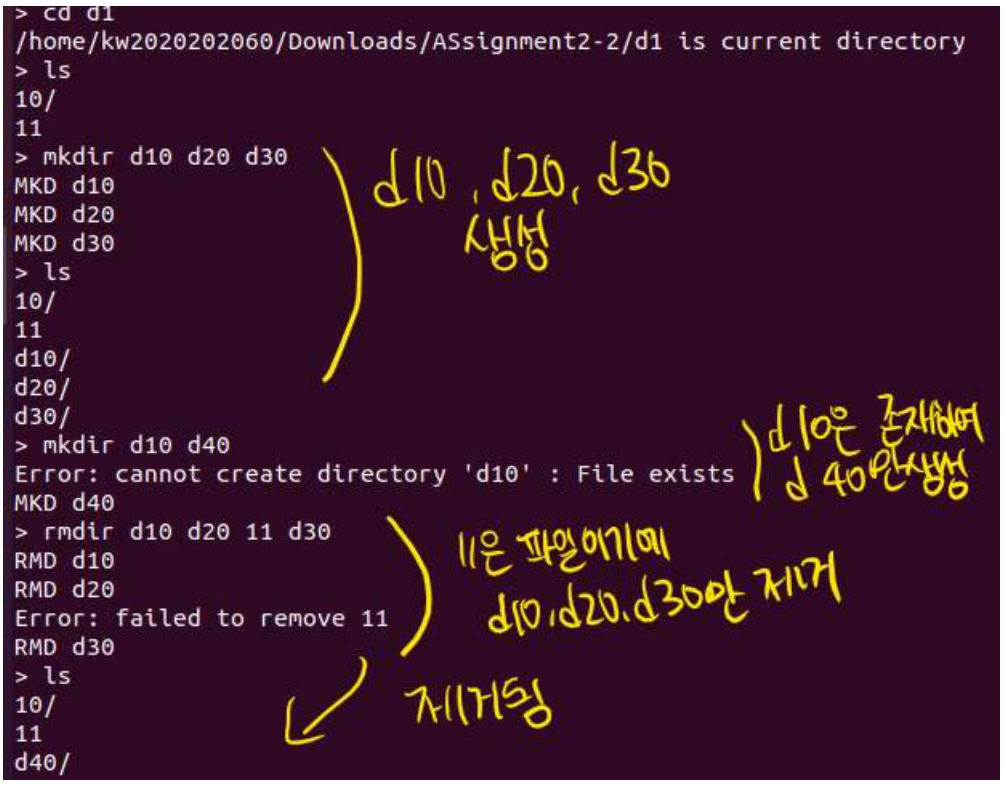


위를 확인하면 dir 에 대한 출력이 ls -al 과 동일하다는 것을 확인할 수 있다. 이때 많은 예외들도 위 설명과 함께 처리됨을 확인할 수 있다.

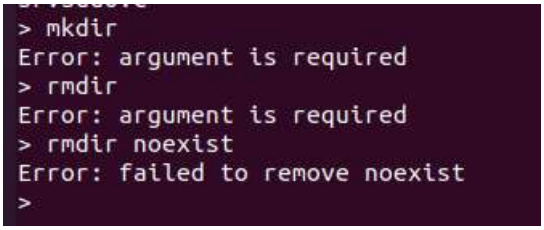
CWD, CDUP, PWD

위 설명과 함께 결과를 확인한다면 CWD, CDUP, PWD 작동이 모두 정상적임을 확인할 수 있다. 이때 옵션이나 많은 인자, 접근권한이 없을 경우에 예외도 정확하게 처리됨을 확인할 수 있다. Fork 를 사용하였기에 cd 로 바꾼 경로를 PWD 명령어로 확인한다면 즉각적으로 경로가 바뀌는 형태 또한 확인할 수 있다.

Mkdir Rmdir



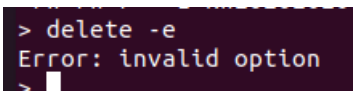
위 결과를 확인하면 모두 명령어가 정상 작동하는 것을 확인할 수 있다. 이때 이미 존재하거나 파일일 경우에는 모두 예외처리가 된 것을 확인할 수 있다.



추가적으로 인자가 부족한 경우와 존재하지 않는 디렉토리를 지우려고 할 때 또한 예외가 정상 처리됨을 확인할 수 있다.

DELE

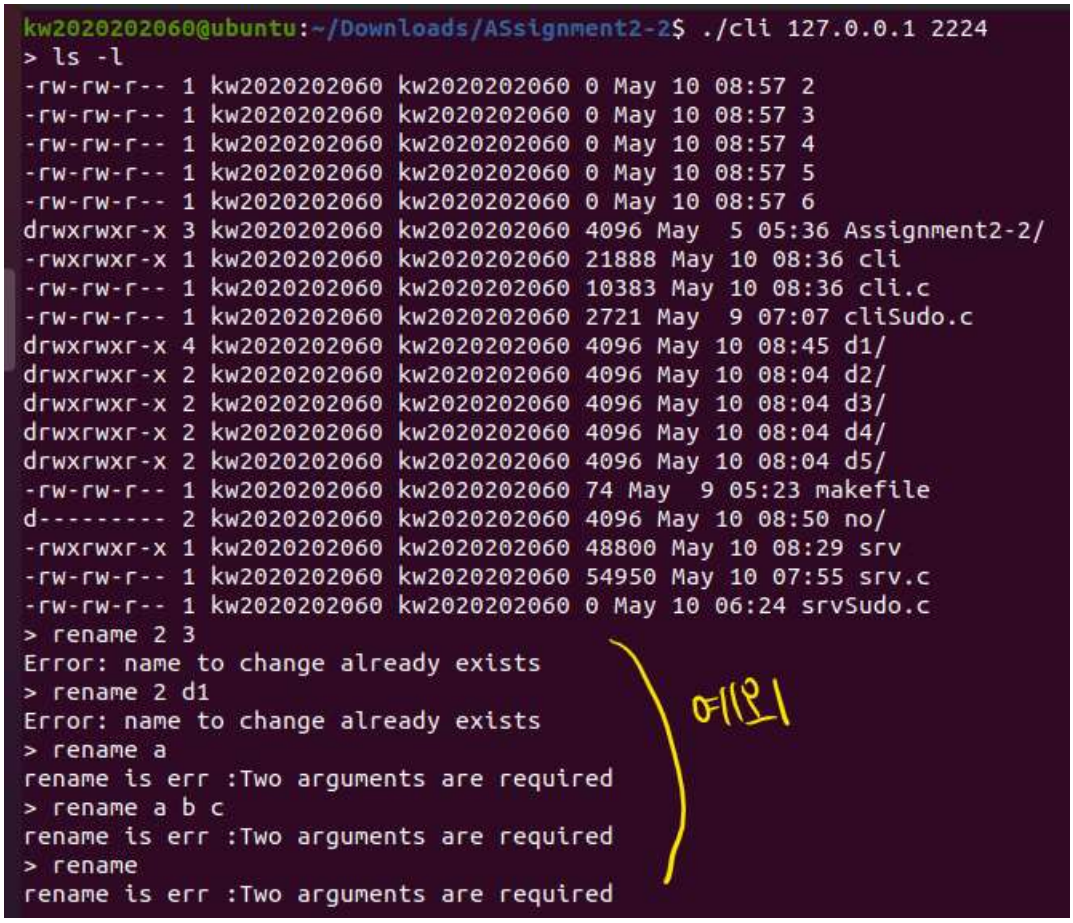
위 결과를 확인하면 인자가 없는 경우에 예외가 처리되었다. 또한 파일 제거는 모두 정상적으로 수행되지만 디렉토리를 제거할 경우에는 제거가 되지 않는 것을 확인할 수 있다.



추가로 옵션이 있는 경우 또한 예외가 처리됨을 확인할 수 있다.

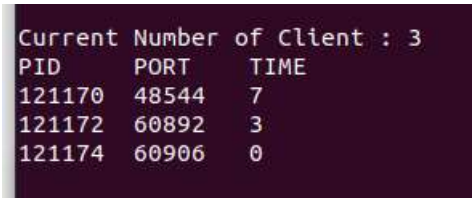
Rename


> ls -l
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 10 08:57 1
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 10 08:57 2
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 10 08:57 3
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 10 08:57 4
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 10 08:57 5
drwxrwxr-x 3 kw2020202060 kw2020202060 4096 May 5 05:36 Assignment2-2/
-rwxrwxr-x 1 kw2020202060 kw2020202060 21888 May 10 08:36 cli
-rw-rw-r-- 1 kw2020202060 kw2020202060 10383 May 10 08:36 cli.c
-rw-rw-r-- 1 kw2020202060 kw2020202060 2721 May 9 07:07 cliSudo.c
drwxrwxr-x 4 kw2020202060 kw2020202060 4096 May 10 08:45 d1/
drwxrwxr-x 2 kw2020202060 kw2020202060 4096 May 10 08:04 d2/
drwxrwxr-x 2 kw2020202060 kw2020202060 4096 May 10 08:04 d3/
drwxrwxr-x 2 kw2020202060 kw2020202060 4096 May 10 08:04 d4/
drwxrwxr-x 2 kw2020202060 kw2020202060 4096 May 10 08:04 d5/
-rw-rw-r-- 1 kw2020202060 kw2020202060 74 May 9 05:23 makefile
d--------- 2 kw2020202060 kw2020202060 4096 May 10 08:50 no/
-rwxrwxr-x 1 kw2020202060 kw2020202060 48800 May 10 08:29 srv
-rw-rw-r-- 1 kw2020202060 kw2020202060 54950 May 10 07:55 srv.c
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 10 06:24 srvSudo.c
> rename 1 6
RNFR 1
RNTO 6
1->6으로
> ls -l
바뀜!
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 10 08:57 2
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 10 08:57 3
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 10 08:57 4
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 10 08:57 5
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 10 08:57 6
drwxrwxr-x 3 kw2020202060 kw2020202060 4096 May 5 05:36 Assignment2-2/
-rwxrwxr-x 1 kw2020202060 kw2020202060 21888 May 10 08:36 cli
-rw-rw-r-- 1 kw2020202060 kw2020202060 10383 May 10 08:36 cli.c
-rw-rw-r-- 1 kw2020202060 kw2020202060 2721 May 9 07:07 cliSudo.c
drwxrwxr-x 4 kw2020202060 kw2020202060 4096 May 10 08:45 d1/
drwxrwxr-x 2 kw2020202060 kw2020202060 4096 May 10 08:04 d2/
drwxrwxr-x 2 kw2020202060 kw2020202060 4096 May 10 08:04 d3/
drwxrwxr-x 2 kw2020202060 kw2020202060 4096 May 10 08:04 d4/
drwxrwxr-x 2 kw2020202060 kw2020202060 4096 May 10 08:04 d5/
-rw-rw-r-- 1 kw2020202060 kw2020202060 74 May 9 05:23 makefile
d--------- 2 kw2020202060 kw2020202060 4096 May 10 08:50 no/
-rwxrwxr-x 1 kw2020202060 kw2020202060 48800 May 10 08:29 srv
-rw-rw-r-- 1 kw2020202060 kw2020202060 54950 May 10 07:55 srv.c
-rw-rw-r-- 1 kw2020202060 kw2020202060 0 May 10 06:24 srvSudo.c
>



위 설명과 함께 결과를 확인한다면 정상적으로 이름이 바뀌는 형태를 확인할 수 있다. 이때 인자 개수가 맞지 않는다면 모두 예외처리가 된 것을 확인할 수 있다.

위를 통해 모든 명령어가 정상 작동함을 확인하였다. 만약 QUIT 가 들어올 경우 서버가 종료되어 소통하던 클라이언트는 Read 를 못하여 종료될 것이다. 이를 통해 부모 프로세스에 서버는 현재 연결중인 클라이언트를 프린트에서 제거하여야 한다. 또한 ctrl+를 입력하였을 때 서버에 QUIT 를 시그널 함수를 통해 전달하였다. 이에 대한 검증은 다음과 같다.

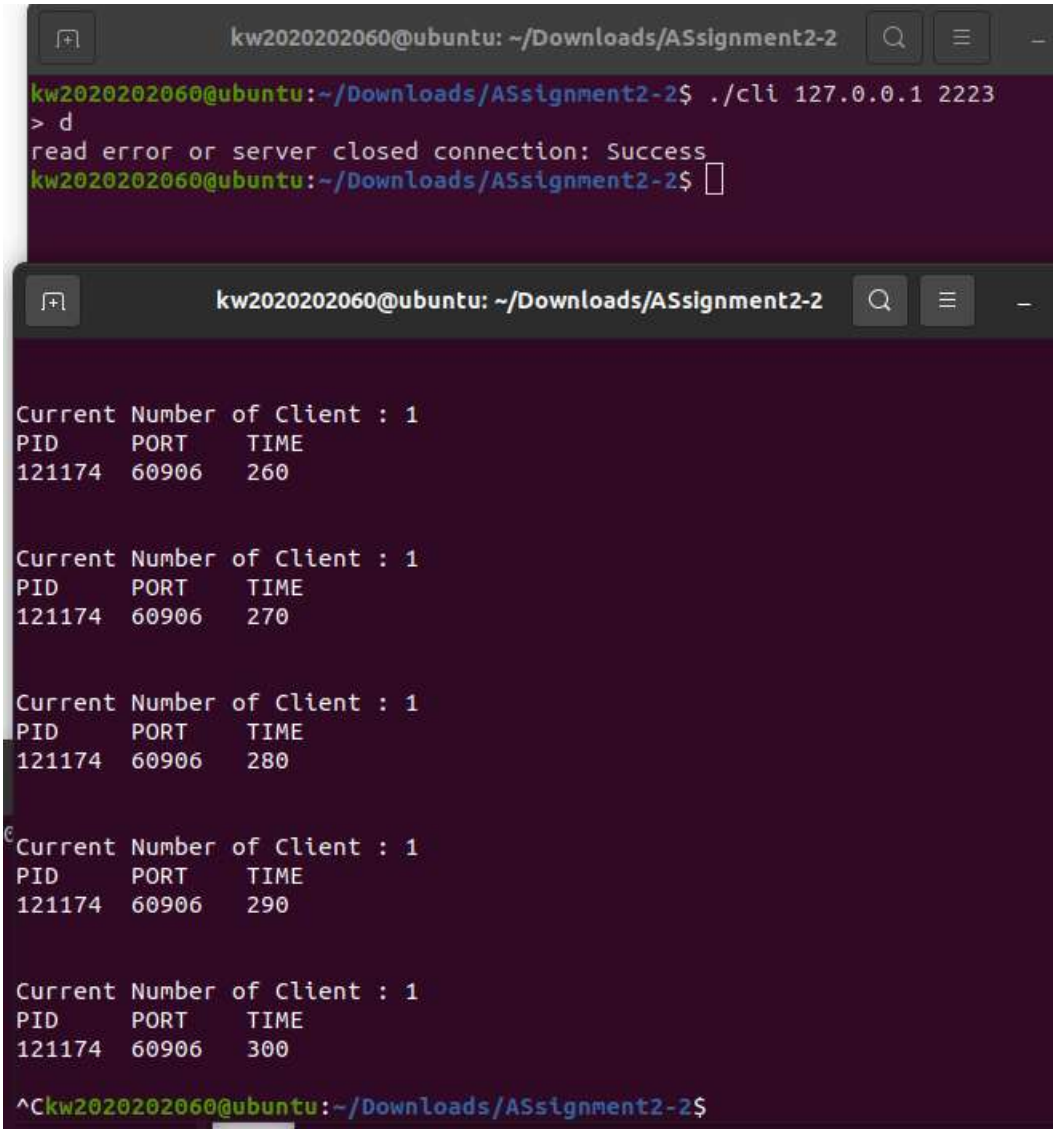


현재는 3 개의 클라이언트가 연결중인 상태이다.

₩

위를 설명과 함께 확인한다면 quit 를 입력하였을 때 서버는 이 프로세스를 종료시키는 것을 확인할 수 있다. 종료가 되었기에 요청을 하던 클라이언트는 read 를 못하여 종료되는 것을 볼 수 있다. 이때 서버에서 해당 프로세스 ID 가 사라지고 연결된 클라이언트 개수가 감소하는 형태를 볼 수 있다.

만약 ctrl+c 로 종료를 했다면 해당 child process 는 종료되는 것을 확인할 수 있고, 목록에서도 제거되는 것을 볼 수 있다.



이때 반대로 서버를 ctrl+c 로 종료시킨다면, 해당 클라이언트에서 입력을 했을 때 응답을 받지 못하기에 클라이언트가 종료되는 것을 확인할 수 있다.

고찰

처음에는 각각의 단계를 분리하여 구현하는 것은 어렵지 않았다. 소켓 통신을 구현하고, 명령어 처리를 위한 함수를 작성하고, FTP 프로토콜에 따른 명령어를 처리하는 등의 작업은 모두 단계적으로 잘 성공했다. 하지만 이러한 각각의 작업을 모두 하나의 프로젝트로 통합하는 것은 조금 혼란스러웠다.

첫째로, 코드의 크기가 커지면서 전체 구조를 이해하고 구현하기 어려웠다. 각각의 단계에서 작성한 코드들을 통합하면서 함수와 변수들이 많아지고, 코드의 흐름을 파악하기도 난잡했다. 이로 인해 버그를 찾는 것이 어려워졌고, 수정하거나 유지 보수하는 것에 힘이 많이 들었던 프로젝트이다.

둘째로, 다양한 부분에서 발생하는 에러를 처리하는 것이 복잡했다. 네트워크 통신에서 발생할 수 있는 다양한 예외 상황들과 오류들을 처리해야 했는데, 이를 모두 고려하고 적절하게 처리하는 것은 쉽지 않았다. 어쩌면 처리 못한 예외도 존재할 수 있을 것이라고 생각했다.

많은 것을 공부하고 지속적으로 코드를 구현해 가면서 이러한 노력들 덕분에 프로젝트를 성공적으로 완료할 수 있었다. 이를 통해 앞으로의 프로젝트에서 더 나은 방법으로 작업할 수 있을 것이라고 생각했다.