시스템 프로그래밍 실습

Assignment3-2

Class	:	금 1, 2 분반
Professor	:	최상호 교수님
Student ID	:	2020202031
Name	:	김재현

Introduction

FTP 는 Client - Server architecture 를 사용하며 신뢰할 수 있는 데이터 전송 서비스를 기반으로 구축됩니다. 이 프로토콜은 클라이언트와 서버 간의 효율적이고 조직적인 통신을 보장하기 위해 control connection 과 data connection 두 개의 별도 연결을 사용하여 작동합니다.

Client 와 Server 는 control connection 을 통해 ftp 명령어 및, code 를 송수신하고, data connection 을 통해 해당 명령의 결과를 수신합니다.

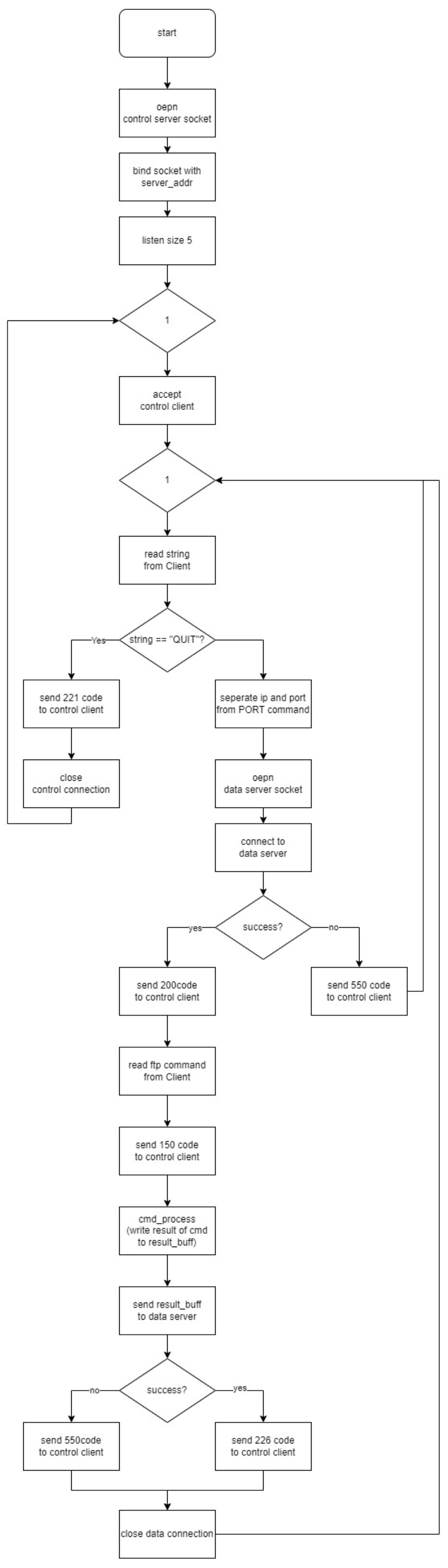
이번 실습을 통해 FTP 의 이중 연결 구조에 대해 더 자세히 이해할 것을 기대합니다.

Flow chart

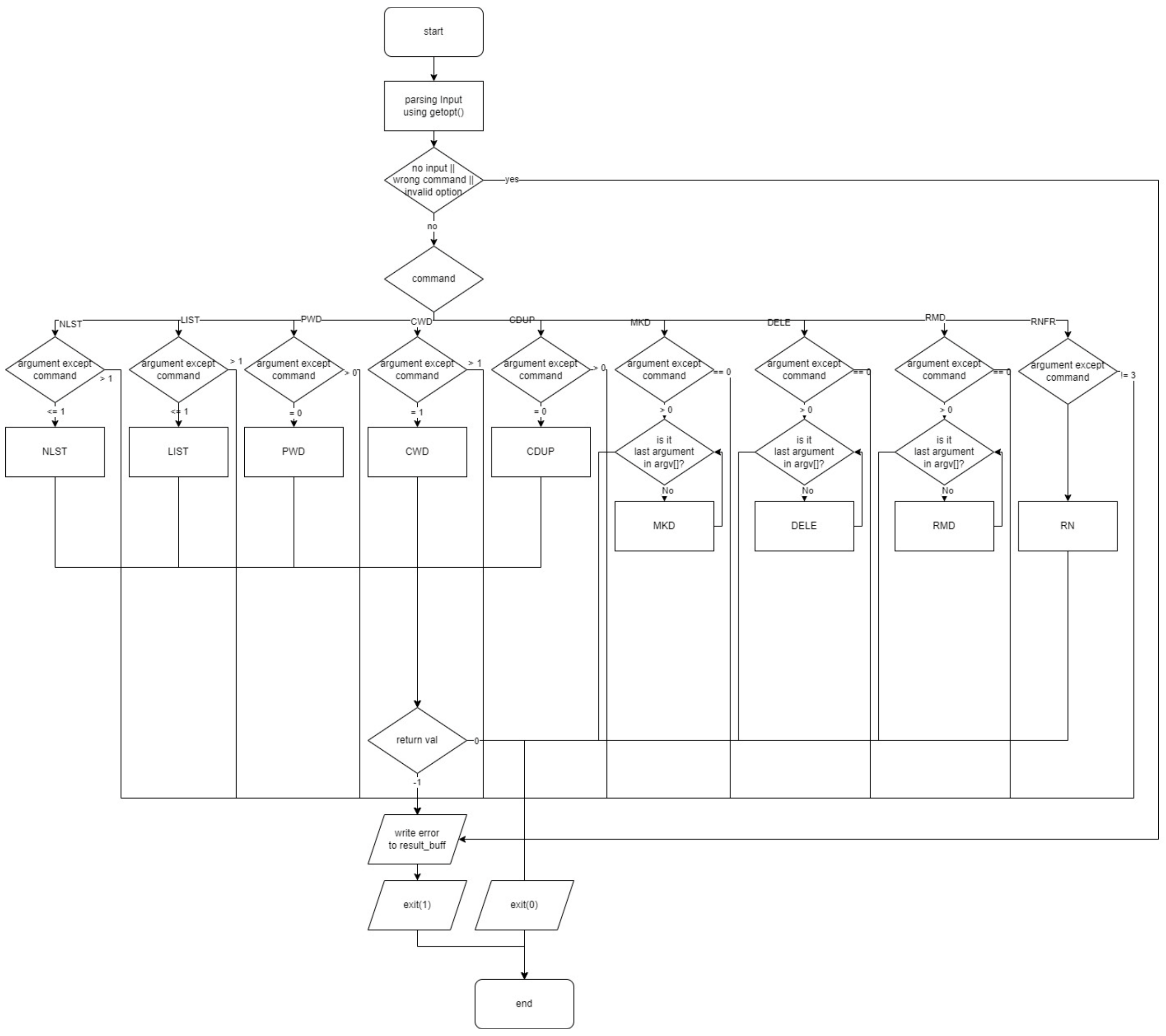
바로 아래에 있습니다.

srv.c

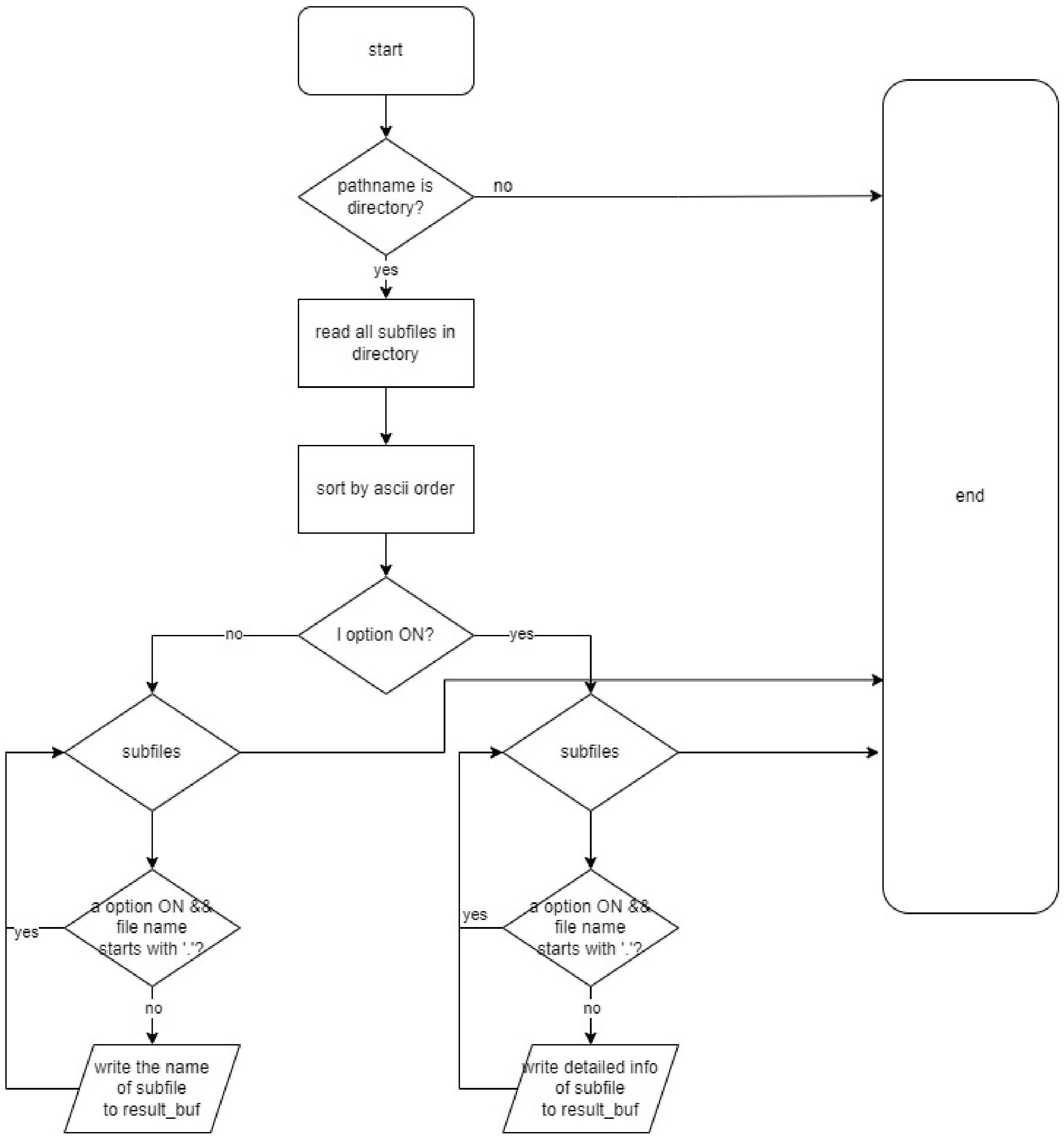
main


start
oepn
control server socket
bind socket with
server_addr
listen size 5
1
accept
control client
1
read string
from Client
string == "QUIT"?
Yes
send 221 code
to control client
close
control connection
seperate ip and port
from PORT command
oepn
data server socket
connect to
data server
success?
yes
no
send 200code
to control client
send 550 code
to control client
read ftp command
from Client
send 150 code
to control client
cmd_process
(write result of cmd
to result_buff)
send result_buff
to data server
success?
no
yes
send 550code
to control client
send 226 code
to control client
close data connection

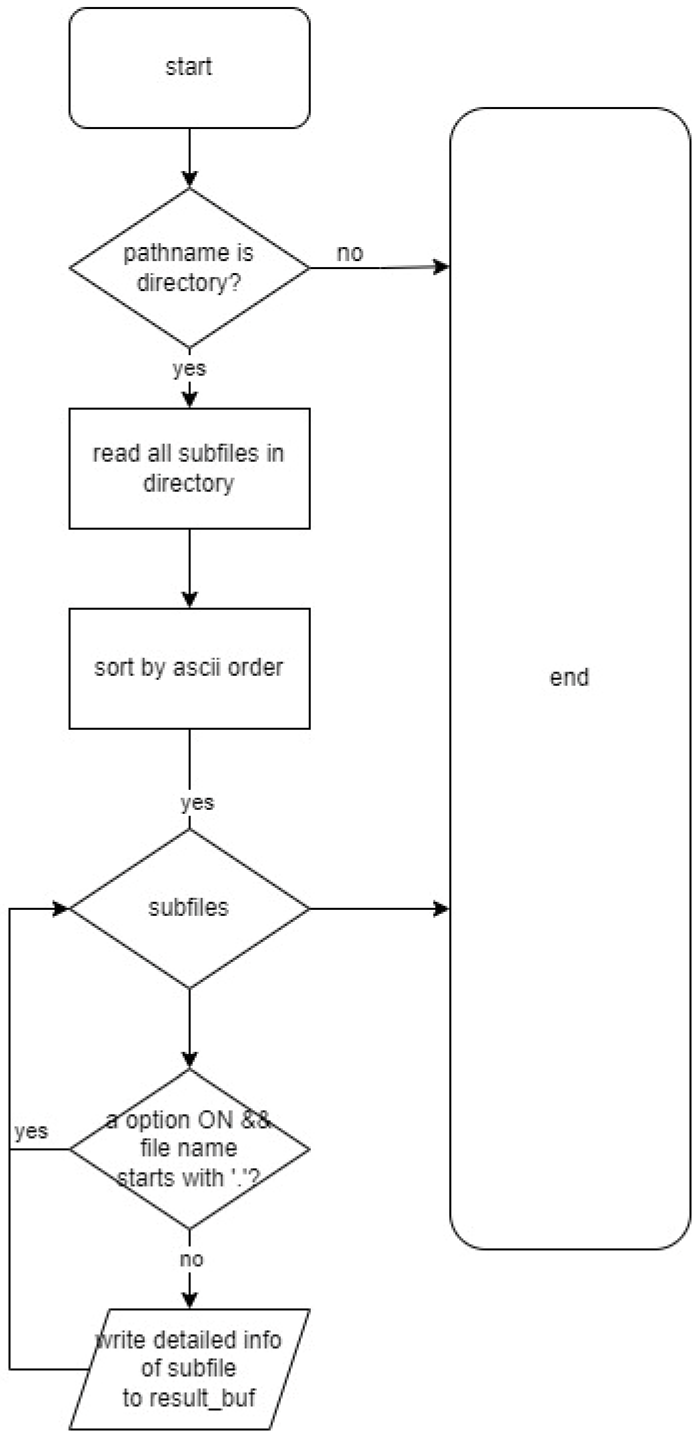
cmd_process


start
parsing Input using getopt()
no input || wrong command || invalid option
yes
no
command
NLST
LIST
PWD
CWD
CDUP
MKD
DELE
RMD
RNFR
argument except command
> 1
<= 1
> 0
= 0
= 1
== 0
!= 3
is it last argument in argv[]?
No
NLST
LIST
PWD
CWD
CDUP
MKD
DELE
RMD
RN
return val
0
-1
write error to result_buff
exit(1)
exit(0)
end

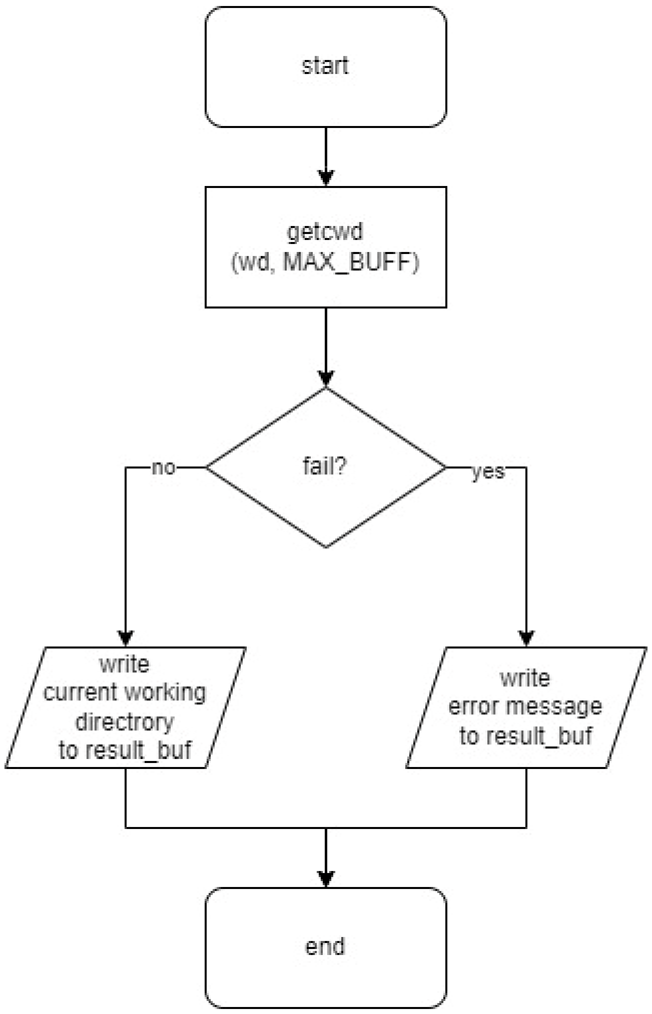
NLST


start
pathname is directory?
no
yes
read all subfiles in directory
sort by ascii order
l option ON?
no
yes
subfiles
subfiles
end
a option ON && file name starts with '.'?
yes
no
a option ON && file name starts with '.'?
yes
no
write the name of subfile to result_buf
write detailed info of subfile to result_buf

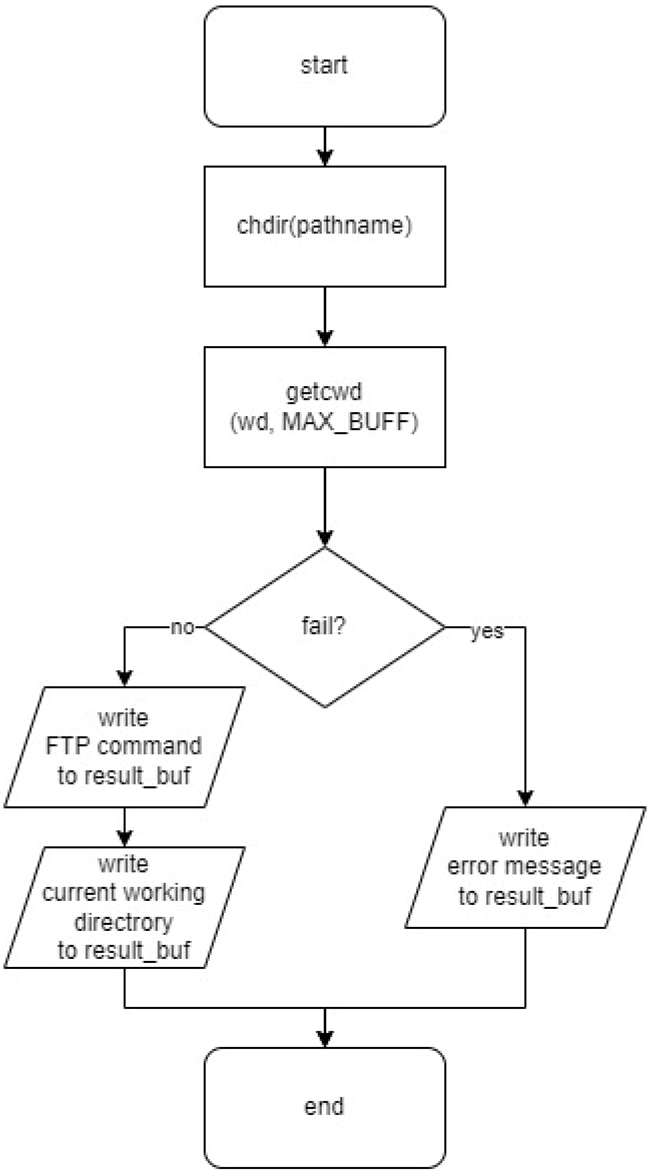
LIST


start
pathname is directory?
no
yes
read all subfiles in directory
sort by ascii order
yes
subfiles
a option ON && file name starts with '.'?
yes
no
write detailed info of subfile to result_buf
end

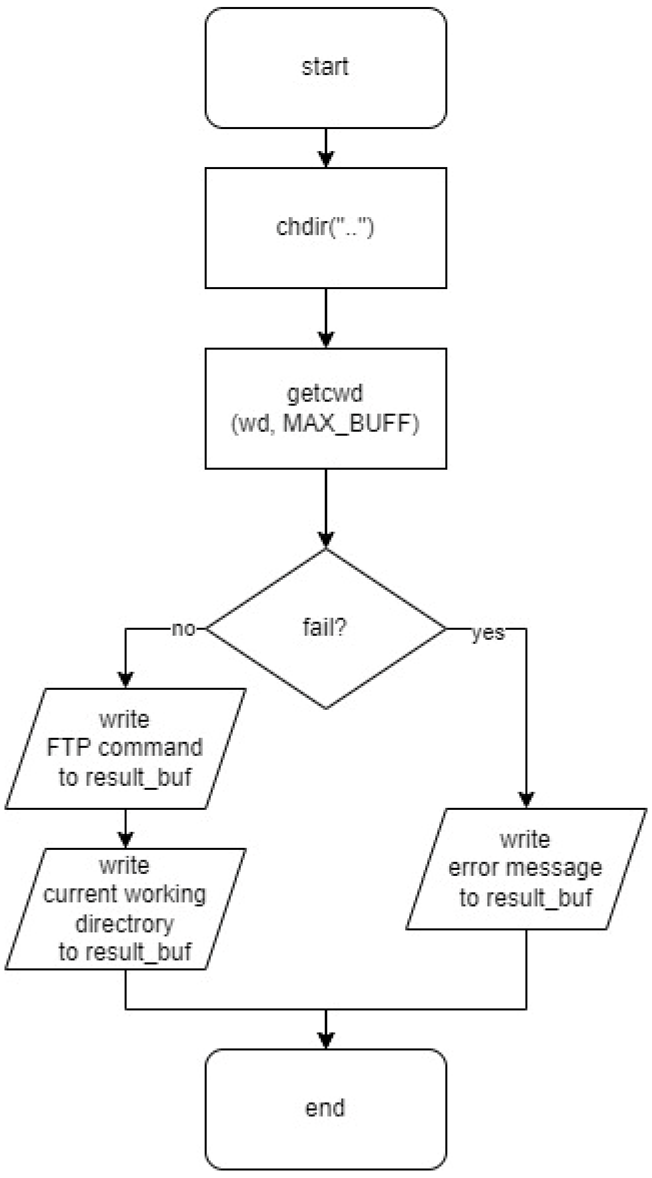
PWD


start
getcwd
(wd, MAX_BUFF)
fail?
no
yes
write
current working
directrory
to result_buf
write
error message
to result_buf
end

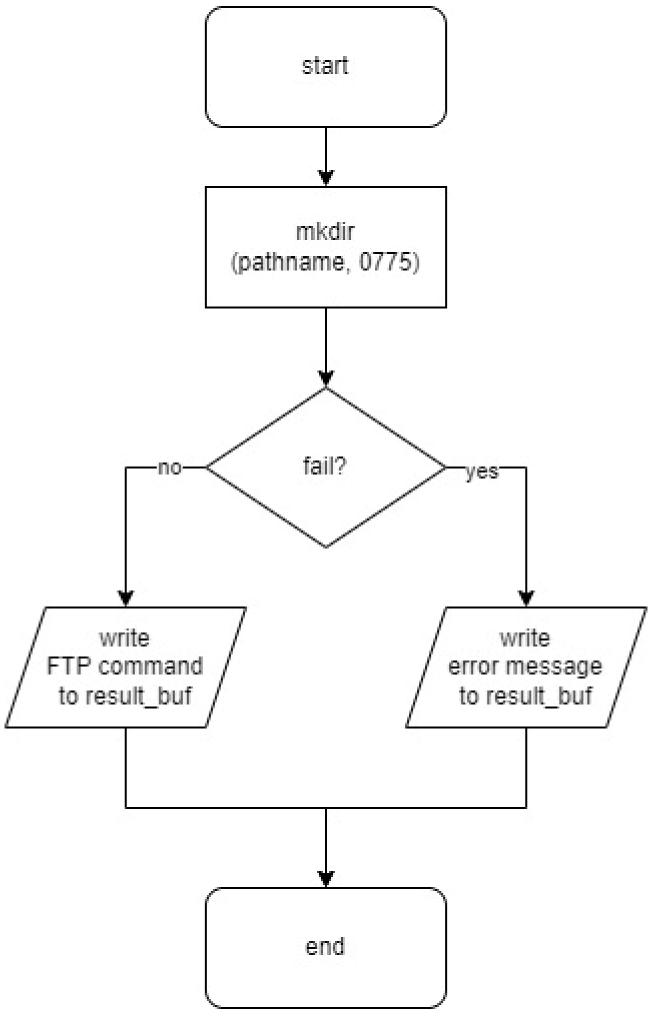
CWD


start
chdir(pathname)
getcwd
(wd, MAX_BUFF)
fail?
no
yes
write
FTP command
to result_buf
write
current working
directrory
to result_buf
write
error message
to result_buf
end

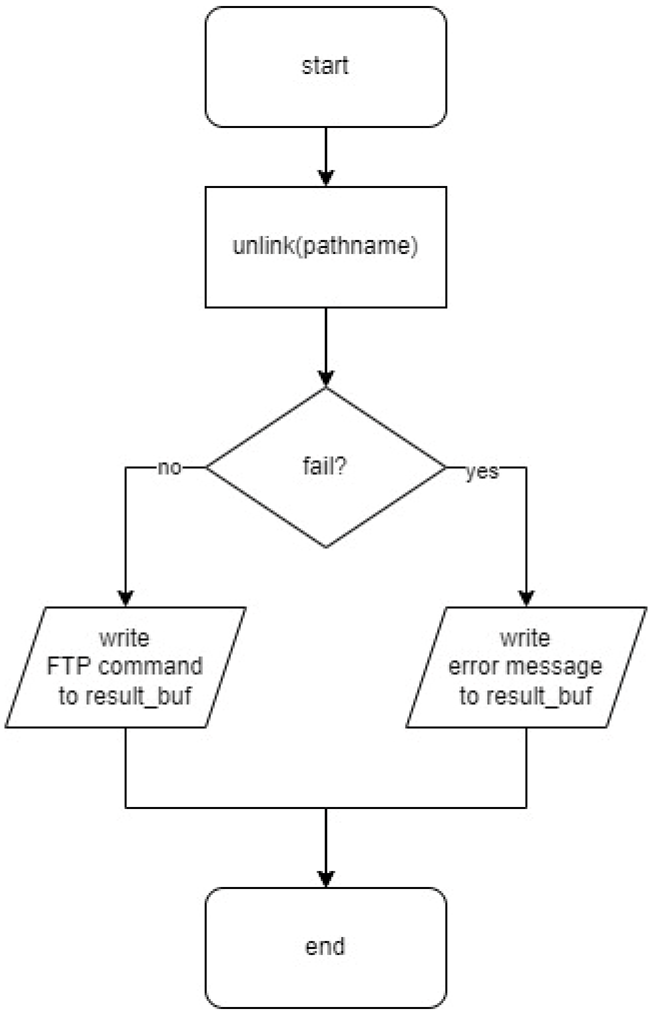
CDUP


start
chdir("..")
getcwd
(wd, MAX_BUFF)
fail?
no
yes
write
FTP command
to result_buf
write
current working
directrory
to result_buf
write
error message
to result_buf
end

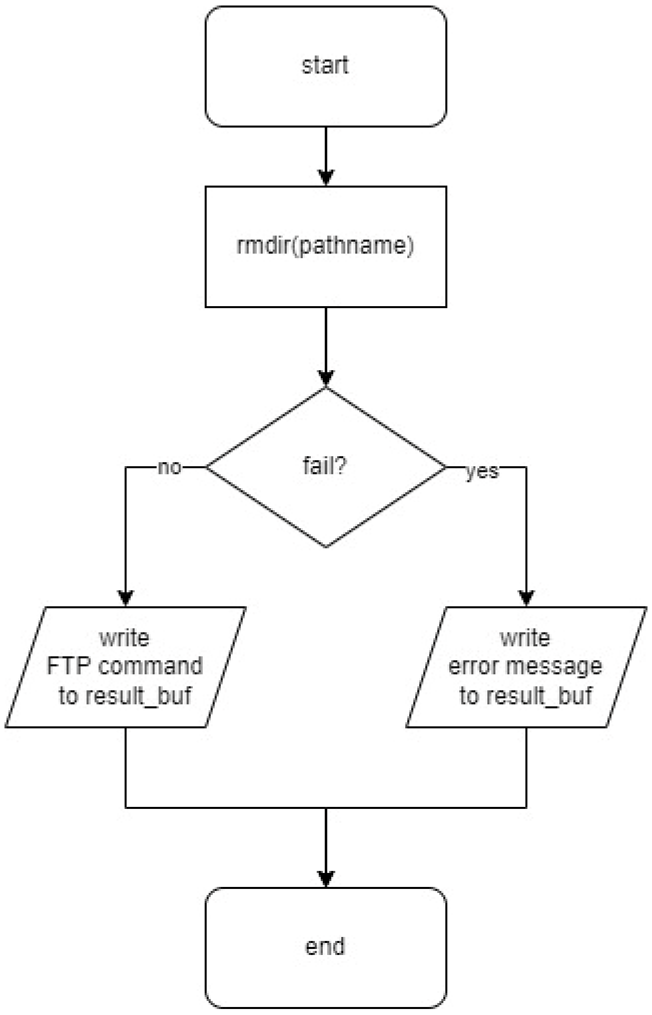
MKD


start
mkdir (pathname, 0775)
fail?
no
yes
write FTP command to result_buf
write error message to result_buf
end

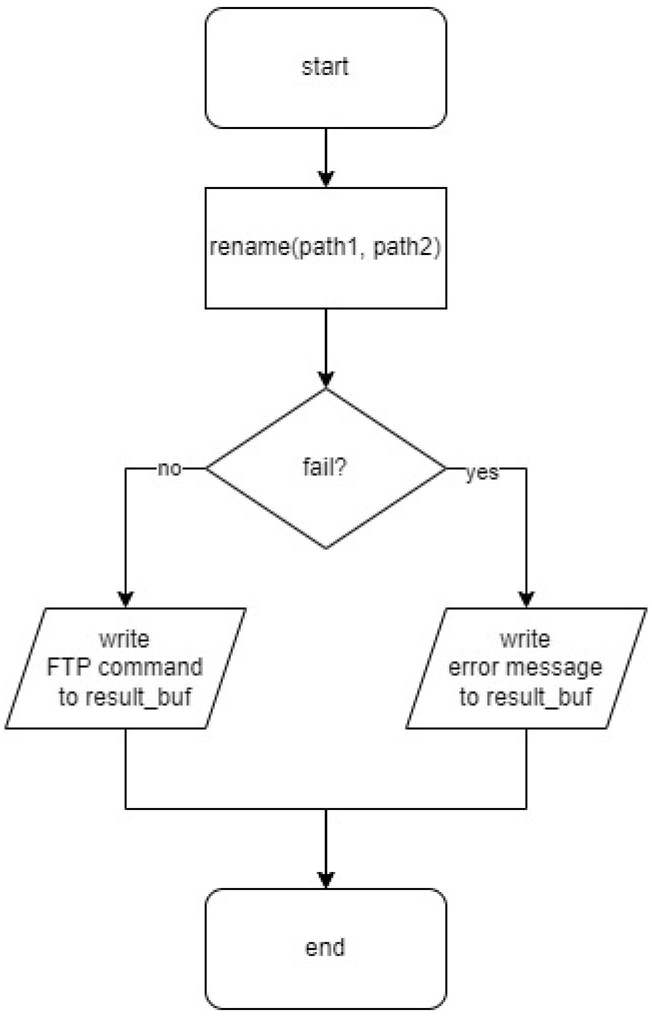
DELE


start
unlink(pathname)
fail?
no
yes
write FTP command to result_buf
write error message to result_buf
end

RMD

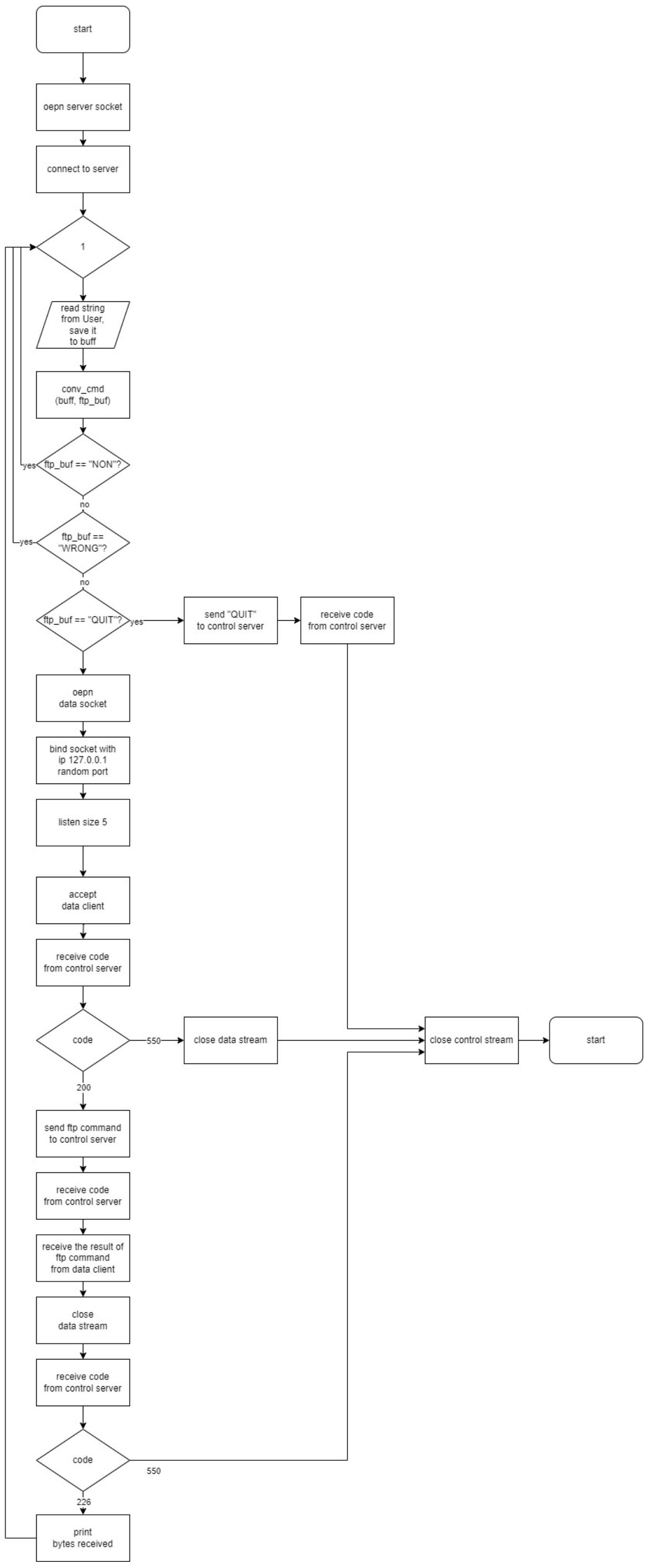

start
rmdir(pathname)
fail?
no
yes
write FTP command to result_buf
write error message to result_buf
end

RN


start
rename(path1, path2)
fail?
no
yes
write FTP command to result_buf
write error message to result_buf
end

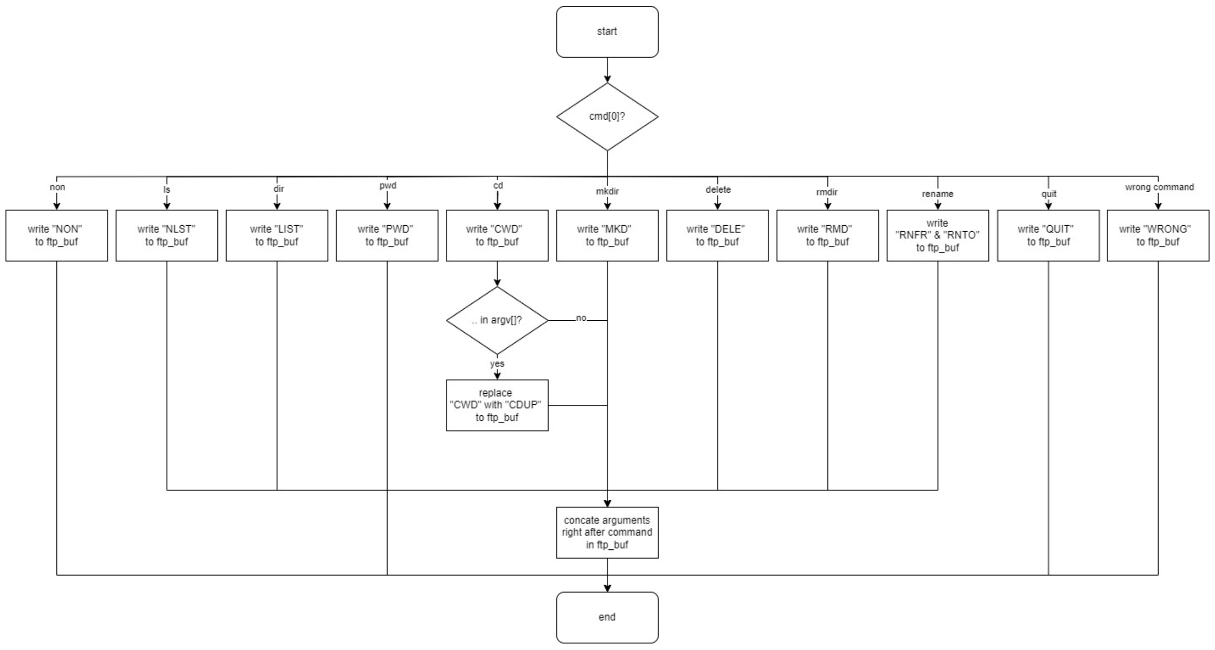
cli.c

main



start
oepn server socket
connect to server
1
read string from User, save it to buff
conv_cmd (buff, ftp_buf)
ftp_buf == "NON"?
yes
no
ftp_buf == "WRONG"?
yes
no
ftp_buf == "QUIT"?
yes
send "QUIT" to control server
receive code from control server
oepn data socket
bind socket with ip 127.0.0.1 random port
listen size 5
accept data client
receive code from control server
code
550
close data stream
close control stream
start
200
send ftp command to control server
receive code from control server
receive the result of ftp command from data client
close data stream
receive code from control server
code
550
226
print bytes received

conv_cmd


start
cmd[0]?
non
ls
dir
pwd
cd
mkdir
delete
rmdir
rename
quit
wrong command
write "NON" to ftp_buf
write "NLST" to ftp_buf
write "LIST" to ftp_buf
write "PWD" to ftp_buf
write "CWD" to ftp_buf
write "MKD" to ftp_buf
write "DELE" to ftp_buf
write "RMD" to ftp_buf
write "RNFR" & "RNTO" to ftp_buf
write "QUIT" to ftp_buf
write "WRONG" to ftp_buf
.. in argv[]?
no
yes
replace "CWD" with "CDUP" to ftp_buf
concate arguments right after command in ftp_buf
end

Pseudo code

srv.c

```
main(argc, argv):
    Initialize ptr, result_buff, ctrl_buff, data_buff, host_ip, port_num, ctrl_server_fd,
ctrl_client_fd, ctrl_server_addr, ctrl_client_addr, data_server_fd, data_server_addr, clilen


    if argc != 2:
        ptr = "enter two arguments!"
        write(STDERR_FILENO, ptr, strlen(ptr))
        return


    ctrl_server_fd = socket(AF_INET, SOCK_STREAM, 0)
    if ctrl_server_fd < 0:
        ptr = "Server: Can't open stream socket.₩n"
        write(STDERR_FILENO, ptr, strlen(ptr))
        return


    memset(ctrl_server_addr, 0, sizeof(ctrl_server_addr))
    ctrl_server_addr.sin_family = AF_INET
    ctrl_server_addr.sin_addr.s_addr = htonl(INADDR_ANY)
    ctrl_server_addr.sin_port = htons(atoi(argv[1]))


    if bind(ctrl_server_fd, (struct sockaddr *)&ctrl_server_addr, sizeof(ctrl_server_addr)) < 0:
        ptr = "Server: Can't bind₩n"
        write(STDERR_FILENO, ptr, strlen(ptr))
        return

listen(ctrl_server_fd, 5)

while True:
        clilen = sizeof(ctrl_client_addr)
        ctrl_client_fd = accept(ctrl_server_fd, (struct sockaddr *)&ctrl_client_addr, &clilen)

        while True:
                memset(ctrl_buff, 0, sizeof(ctrl_buff))
                memset(data_buff, 0, sizeof(data_buff))
                memset(result_buff, 0, sizeof(result_buff))

                read(ctrl_client_fd, ctrl_buff, MAX_BUFF)
                print(ctrl_buff)

                if ctrl_buff == "QUIT":
                        ptr = "221 Goodbye."
                        write(ctrl_client_fd, ptr, strlen(ptr) + 1)
                        print(ptr)
                        break

                host_ip = convert_str_to_addr(ctrl_buff, &port_num)
        data_server_fd = socket(AF_INET, SOCK_STREAM, 0)
        if data_server_fd < 0:
            ptr = "Server: Can't open stream socket.₩n"
            write(STDERR_FILENO, ptr, strlen(ptr))
            exit(1)

        memset(data_server_addr, 0, sizeof(data_server_addr))
        data_server_addr.sin_family = AF_INET
        data_server_addr.sin_addr.s_addr = inet_addr(host_ip)
        data_server_addr.sin_port = htons(port_num)
        free(host_ip)

        if connect(data_server_fd, (struct sockaddr *)&data_server_addr,
sizeof(data_server_addr)) < 0:
            ptr = "550 Failed to access."
            write(ctrl_client_fd, ptr, strlen(ptr) + 1)
            print(ptr)
            continue
        else:
            ptr = "200 Port command performed successfully."
            write(ctrl_client_fd, ptr, strlen(ptr) + 1)
            print(ptr)
        read(ctrl_client_fd, ctrl_buff, MAX_BUFF)
        print(ctrl_buff)

        ptr = "150 Opening data connection for directory list."
        write(ctrl_client_fd, ptr, strlen(ptr) + 1)
        print(ptr)

        cmd_process(ctrl_buff, result_buff)

        if write(data_server_fd, result_buff, MAX_BUFF) > 0:
            ptr = "226 Complete transmission."
            write(ctrl_client_fd, ptr, strlen(ptr) + 1)
            print(ptr)
        else:
            ptr = "550 Failed transmission."
            write(ctrl_client_fd, ptr, strlen(ptr) + 1)
            print(ptr)

        close(data_server_fd)

    close(ctrl_client_fd)
```

```
convert_str_to_addr(str, port):
    addr = str[5:] // Extract the part after "PORT " from the input string
    ip = allocate memory of size 30 characters
    j = 0

    // Tokenize the address string and construct the IP address
    token = strtok(addr, ",")
    j += sprintf(ip + j, "%s", token)
    for i = 0 to 2:
        token = strtok(NULL, ",")
        j += sprintf(ip + j, ".%s", token)

    // Parse the port number from the last two tokens
    port = 0
    token = strtok(NULL, ",")
    port += atoi(token) << 8 // Shift the first byte to the left by 8 bits
    token = strtok(NULL, ",")
    port += atoi(token) // Add the second byte to the port number

    return ip
```

```
int cmd_process(const char *buff, char *result_buff)
{
    parsing buf using getopt();

    if (input not fit in ftp command form)
        write error message to result_buf;
    else
    {
        if (command is "NLST")
        {
            if (there are too many arguments)
                write an error message to result_buf and return 0;
            if (NLST < 0)
                write an error message to result_buf and return -1;
        }
        else if (command is "LIST")
        {
            if (there are too many arguments)
                write an error message to result_buf and return 0;
            if (LIST < 0)
                print an error message and return -1;
        }
        else if (command is "PWD")
        {
            if (an argument is provided)
                write an error message to result_buf and return 0;
            if (PWD < 0)
                return -1;
        }
        else if (command is "CWD")
        {
            if (there are too many arguments)
                write an error message to result_buf and return 0;
            if (CWD < 0)
                return -1;
        }
        else if (command is "CDUP")
        {
            if (there are too many arguments)
                write an error message to result_buf and return 0;
            if (CDUP < 0)
                return -1;
        }
        else if (command is "MKD")
        {
            if (there is no arguments)
                write an error message to result_buf and return 0;
            for (argv[])
                MKD;
        }
        else if (command is "DELE")
        {
            if (there is no arguments)
                write an error message to result_buf and return 0;
            for (argv[])
                DELE;
        }
        else if (command is RMD)
        {
            if (there is no arguments)
                write an error message to result_buf and return 0;
            for (argv[])
                RMD;
        }
        else if (command is RNFR and RNTO)
        {
            if (the number of arguments != 2)
                write an error message to result_buf and return 0;
            if (filename already exists)
                write an error message to result_buf and return 0;
            RN;
        }
    }

    return 0;
}
```

```
int NLST(char *result_buff, const char *pathname, int opflag)
{
    if (pathname is not directory)
        return -1;

    read all subfiles in directory named pathname;
    sort subfiles by ascii order;

    if (l option ON)
    {
        while (subfiles)
        {
            if (a option off && filename starts with '.')
                continue;
            else
                write detailed information of subfile to result_buf;


        }
    }
    else // l option OFF
    {
        while (subfiles)
        {
            if (a option off && filename starts with '.')
                continue;
            else
                write name of subfile to result_buf;
        }
    }
}
```

```
int LIST(char *result_buff, const char *pathname)
{
    if (pathname is not directory)
        return -1;

    read all subfiles in directory named pathname;
    sort subfiles by ascii order;

    while (subfiles)
    {
        if (a option off && filename starts with '.')
            continue;
        else
            write detailed information of subfile to result_buf;
    }
}
```

```
int PWD(char *result_buff)
{
    char wd[MAX_BUFF];

    if (getcwd(wd, MAX_BUFF) == NULL)
    {
        write error to result_buf;
        return -1;
    }
    else
    {
        write current working directory to result_buf;
        return 0;
    }
}
```

```c
int CWD(char *result_buff, const char *pathname)
{
    char wd[MAX_BUFF];

    if (chdir(pathname) < 0 || getcwd(wd, MAX_BUFF) == NULL)
    {
        write error to result_buf;
        return -1;
    }
    else
    {
        write FTP command to result_buf;
        write current working directory to result_buf;
        return 0;
    }
}
```

```c
int CDUP(char *result_buff)
{
    char wd[MAX_BUFF];

    if (chdir("..") < 0 || getcwd(wd, MAX_BUFF) == NULL)
    {
        write error to result_buf;
        return -1;
    }
    else
    {
        write FTP command to result_buf;
        write current working directory to result_buf;
        return 0;
    }
}
```

```
int MKD(char *result_buff, const char *pathname)
{
    char str[MAX_BUFF];

    if (mkdir(pathname, 0775) == 0)
    {
        write FTP command to result_buf;
        return 0;
    }
    else
    {
        write error to result_buf;
        return -1;
    }
}
```

```
int DELE(char *result_buff, const char *pathname)
{
    char str[MAX_BUFF];

    if (unlink(pathname) == 0)
    {
        write FTP command to result_buf;
        return 0;
    }
    else
    {
        write error to result_buf;
        return -1;
    }
}
```

```
int RMD(char *result_buff, const char *pathname)
{
    char str[MAX_BUFF];

    if (rmdir(pathname) == 0)
    {
        write FTP command to result_buf;
        return 0;
    }
    else
    {
        write error to result_buf;
        return -1;
    }
}
```

```
int RN(char *result_buff, const char *pathname1, const char *pathname2)
{




    if (rename(pathname1, pathname2) == 0)
    {
        write FTP command to result_buf;
        return 0;
    }
    else
    {
        write error to result_buf;
        return -1;
    }
}
```

cli.c

```
main(argc, argv):

initialize ptr, hostport, port, ctrl_server_fd, data_server_fd, data_client_fd, ctrl_server_addr, data_server_addr, data_client_addr, clilen, buff, ftp_buff, ctrl_buff, data_buff


if argc is not 3:
    set ptr to "enter three arguments!"
    write ptr to STDERR_FILENO
    exit program


ctrl_server_fd = socket(AF_INET, SOCK_STREAM, 0)
if ctrl_server_fd is less than 0:
    set ptr to "Server: Can't open stream socket.₩n"
    write ptr to STDERR_FILENO
    exit program


set ctrl_server_addr to 0
ctrl_server_addr.sin_family = AF_INET
ctrl_server_addr.sin_addr.s_addr = inet_addr(argv[1])
ctrl_server_addr.sin_port = htons(atoi(argv[2]))


if connect(ctrl_server_fd, (struct sockaddr *)&ctrl_server_addr, sizeof(ctrl_server_addr)) is less than 0:
    set ptr to "control connection fails₩n"
    write ptr to STDERR_FILENO
    close(ctrl_server_fd)
    return

seed random number generator with time(NULL)

while True:
    set buff, ftp_buff, ctrl_buff, data_buff to 0

    write "> " to STDOUT_FILENO

    read from STDIN_FILENO into buff up to MAX_BUFF
    if read failed:
        set ptr to "read error!"
        write ptr to STDERR_FILENO
        close(ctrl_server_fd)
        exit program

    remove trailing newline from buff

    convert buff to ftp_buff using conv_cmd

    if ftp_buff is "NON":
        set ptr to "Non Command!₩n₩n"
        write ptr to STDERR_FILENO
        continue

    if ftp_buff is "WRONG":
        set ptr to "Invalid Command!₩n₩n"
        write ptr to STDERR_FILENO
        continue

    if ftp_buff is "QUIT":
        write "QUIT" to ctrl_server_fd
        read from ctrl_server_fd into ctrl_buff up to MAX_BUFF
        print ctrl_buff
        break

    port = random number between 10001 and 30000

    data_server_fd = socket(AF_INET, SOCK_STREAM, 0)
    if data_server_fd is less than 0:
        set ptr to "Server: Can't open stream socket.₩n"
        write ptr to STDERR_FILENO
        close(ctrl_server_fd)
        break

    set data_server_addr to 0
```

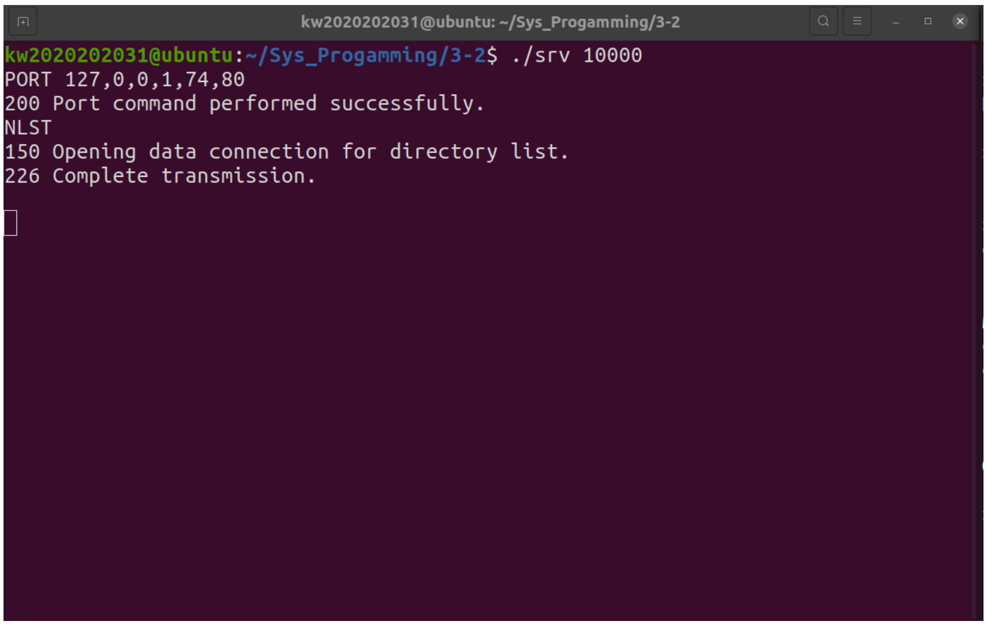
```
    data_server_addr.sin_family = AF_INET
    data_server_addr.sin_addr.s_addr = inet_addr("127.0.0.1")
    data_server_addr.sin_port = htons(port)

    if bind(data_server_fd, (struct sockaddr *)&data_server_addr, sizeof(data_server_addr))
is less than 0:
        set ptr to "Server: Can't bind₩n"
        write ptr to STDERR_FILENO
        close(ctrl_server_fd)
        close(data_server_fd)
        break

    listen(data_server_fd, 5)

    hostport          =          convert_addr_to_str(data_server_addr.sin_addr.s_addr,
data_server_addr.sin_port)
    print "converting to ", hostport
    write hostport to ctrl_server_fd up to MAX_BUFF
    free hostport

    clilen = sizeof(data_client_addr)
    data_client_fd = accept(data_server_fd, (struct sockaddr *)&data_client_addr, &clilen)

    read from ctrl_server_fd into ctrl_buff up to MAX_BUFF
    if ctrl_buff is "200 Port command performed successfully.":
```

```
print ct
```

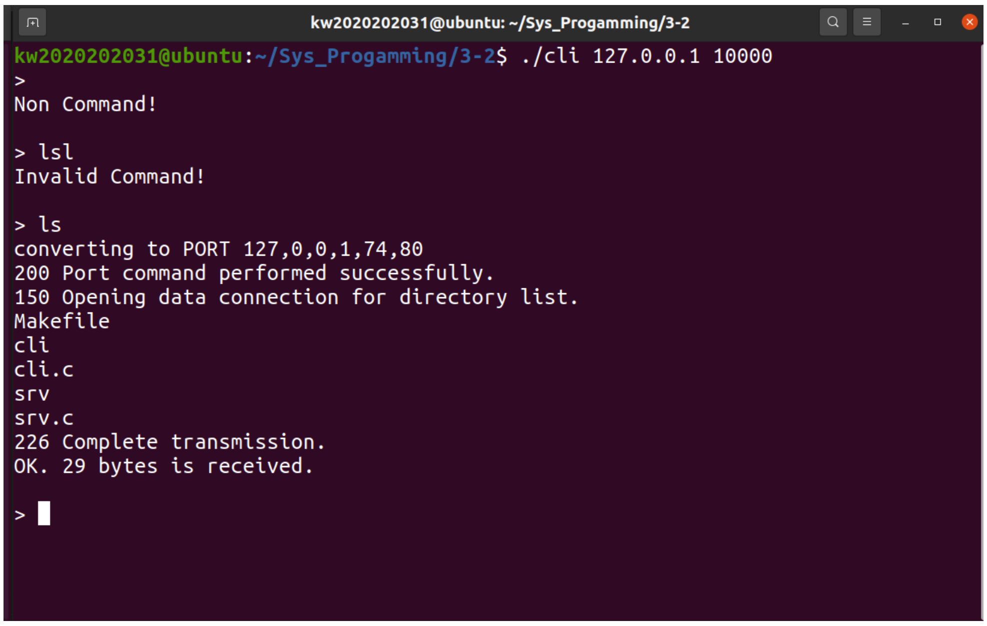
```
convert_addr_to_str(ip_addr, port):
    cmd_port = allocate memory of size 30 characters
    j = 0

    // Convert ip_addr to host byte order
    ip_addr = ntohl(ip_addr)

    // Convert port to host byte order
    port = ntohs(port)

    // Append "PORT " to cmd_port
    j += sprintf(cmd_port + j, "PORT ")

    // Convert ip_addr to string format
    for i = 3 down to 0:
        byte = (ip_addr & (0xFF << (8 * i))) >> (8 * i)
        j += sprintf(cmd_port + j, "%lu,", byte)

    // Convert port to string format
    high_byte = (port & 0xFF00) >> 8
    low_byte = port & 0x00FF
    j += sprintf(cmd_port + j, "%u,%u", high_byte, low_byte)

    return cmd_port
```

```
conv_cmd
{
    getopt(cmd_buf)
    if( the number of input arguments is 0)
        Copy the string "NON" to ftp_buf.
    else if (first input argument is "ls")
        Copy the string "NLST" to ftp_buf.
    else if (first input argument is "dir")
        Copy the string "LIST" to ftp_buf.
    else if (first input argument is "pwd")
        Copy the string "PWD" to ftp_buf.


    else if (first input argument is "cd")
        Copy the string "CWD" to ftp_buf.

    If additional argument is ".."
        Copy the string "CDUP" to ftp_buf.
    else
        append additional argument to ftp_buf.
    else if (first input argument is "mkdir")
        Copy the string "MKD" to ftp_buf.
    else if (first input argument is "delete")
        Copy the string "DELE" to ftp_buf.
    else if (first input argument is "rmdir")
        Copy the string "RMD" to ftp_buf.
    else if (first input argument is "rename")
        Copy the string "RNFR" and the second argument to ftp_buf.
        Copy the string "RNTO" and the third argument to ftp_buf.
    else if (first input argument is "quit")
        Copy the string "QUIT" to ftp_buf.
    else (incorrect command entered)
        Copy the string "WRONG" to ftp_buf.

    If there are additional arguments:
        Append a space to ftp_buf.
        Append the additional argument to ftp_buf.
}
```

결과화면

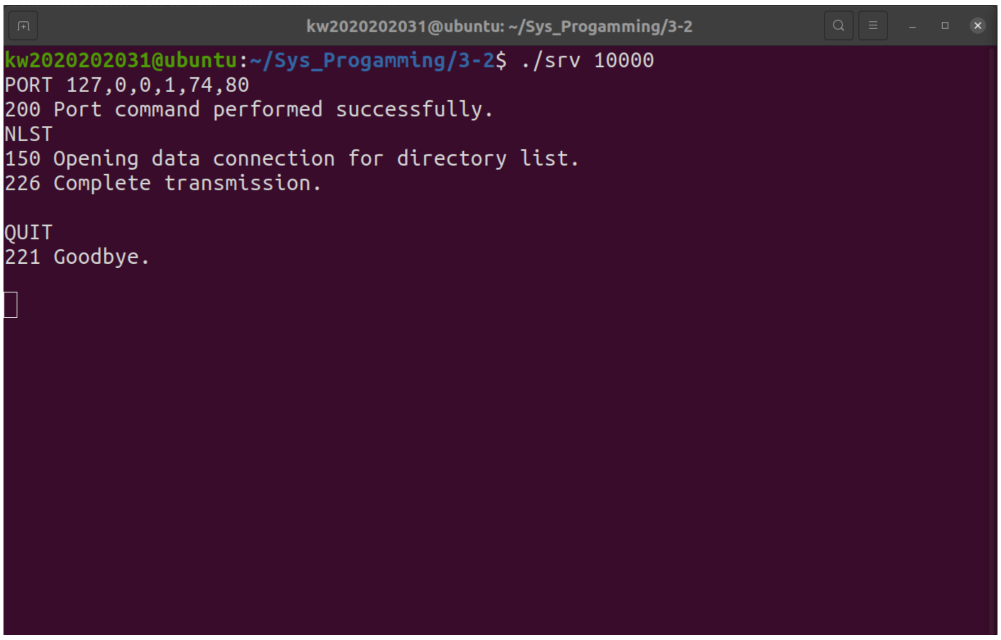


```
kw2020202031@ubuntu:~/Sys_Progamming/3-2$ ./srv 10000
PORT 127,0,0,1,74,80
200 Port command performed successfully.
NLST
150 Opening data connection for directory list.
226 Complete transmission.
```

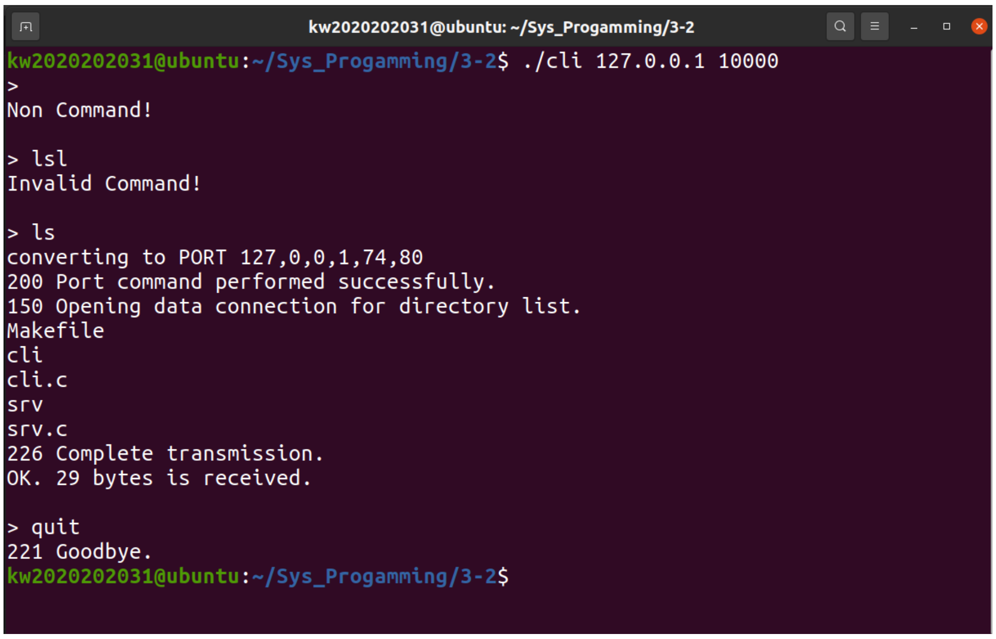


```
kw2020202031@ubuntu:~/Sys_Progamming/3-2$ ./cli 127.0.0.1 10000
>
Non Command!

> lsl
Invalid Command!

> ls
converting to PORT 127,0,0,1,74,80
200 Port command performed successfully.
150 Opening data connection for directory list.
Makefile
cli
cli.c
srv
srv.c
226 Complete transmission.
OK. 29 bytes is received.

>
```

USER 가 잘못된 명령어를 입력하거나, 아무런 입력도 하지 않으면, Invalid Command, Non Command 오류 메시지를 출력하는 것을 확인할 수 있고, ls 명령어를 입력하면 PORT command 송수신, codes 송수신, result data 송수신 여부 등을 출력결과를 통해 확인할 수 있습니다.



```
kw2020202031@ubuntu:~/Sys_Progamming/3-2$ ./srv 10000
PORT 127,0,0,1,74,80
200 Port command performed successfully.
NLST
150 Opening data connection for directory list.
226 Complete transmission.

QUIT
221 Goodbye.
```



```
kw2020202031@ubuntu:~/Sys_Progamming/3-2$ ./cli 127.0.0.1 10000
>
Non Command!

> lsl
Invalid Command!

> ls
converting to PORT 127,0,0,1,74,80
200 Port command performed successfully.
150 Opening data connection for directory list.
Makefile
cli
cli.c
srv
srv.c
226 Complete transmission.
OK. 29 bytes is received.

> quit
221 Goodbye.
kw2020202031@ubuntu:~/Sys_Progamming/3-2$
```

USER 가 quit 명령어를 입력하자 221 Goodbye 라는 코드를 송수신한 후 client 측 프로그램이 종료되는 것을 확인할 수 있고, server 측 프로그램은 client 와의 연결을 종료하고, 다른 client 의 연결요청을 대기하고 있음을 확인할 수 있습니다.

고찰

client 측에서 server 로 PORT command 를 줄 때, 본인의 ip 를 server 에 전달해 줘야하는데, USER 의 ip 를 알아낼 수 있는 방법을 찾을 수 없었습니다.

하지만 우리의 실습과제에서는 로컬 ip 를 사용하여 실습하므로, data connection socket ip address 를 "127.0.0.1"로 고정시켰습니다.

이는 같은 기기 내에서만 작동하는 Server - Client 모델이므로 본인이 접속해 있는 ip address 를 알 수 있는 방법을 찾아 적용시킨다면, 서로 다른 기기 간 Server - Client 모델도 구현 가능할 것임을 예상할 수 있습니다.

Reference

시스템프로그래밍실습 / 광운대학교 / 최상호 교수님 / 2024-1_SPLab_07_FTP3_1_v2

시스템프로그래밍실습 / 광운대학교 / 최상호 교수님 / 2024-1_SPLab_FTP_Assginment3_2_v2