시스템 프로그래밍 실습

[Assignment3_3]

Class : D

Professor : 이혁준 교수님

Student ID : 2020202095

Name : 김준원

Introduction

이번 프로젝트는 그동안 ftp 1 , 2 ,3 에서 해왔던 프로젝트의 결과들을 모으고 모아 최종 ftp project 를 완성하는 과제이다. 기본적인 명령어인 ls, pwd , cd , mkdir , rmdir , rename , quit 와 추가적인 명령어인 type , get , put 을 구현하는 것이 목적이다. 일반 명령어는 기존의 프로젝트와 동일하고, 나머지 명령어 들에 대해서 설명해보자면 다음과 같다.

Type : binary mode 와 ascii mode 로 type 을 변경하는 command 인데, binary mode 로 type 을 변경할 때에는 type binary 나 bin command 를 입력하면 되고, ascii mode 로 type 을 변경할 때에는 type ascii 나 ascii command 를 입력해서 type 을 바꾸면 된다.

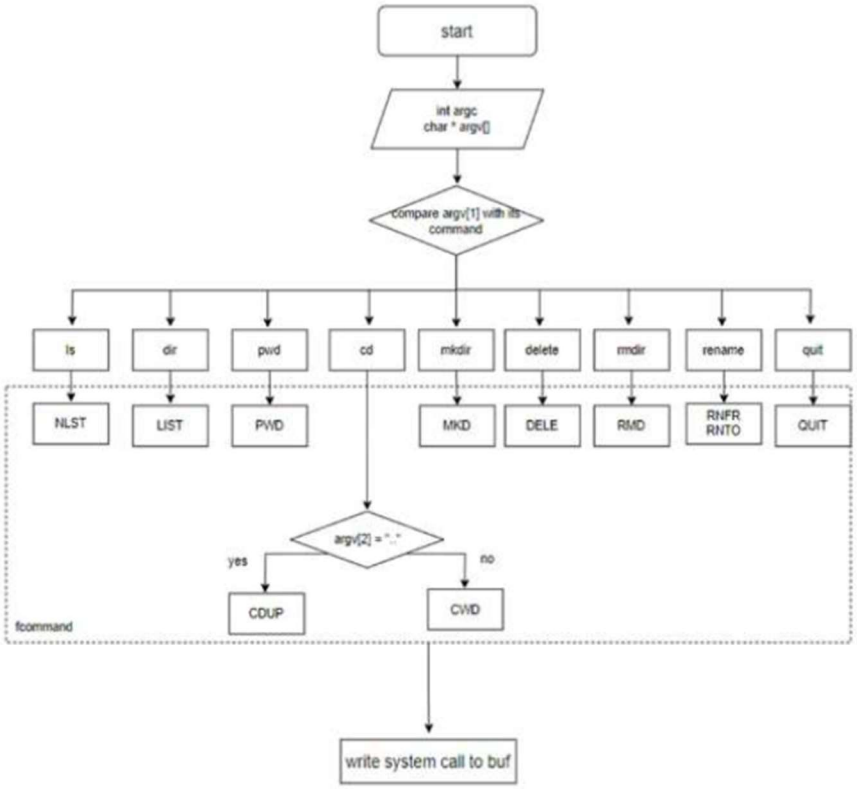
다음으로 get 과 put 은 파일을 주고 받을 때 사용하는 command 인데, 먼저 get 에 대해서 알아보자면, get 은 파일을 받을 때 사용하는 명령어 인데 이번 프로젝트에서는 server 쪽에 있는 파일을 get 명령어를 통해서 client 쪽으로 가져오는 역할을 수행함으로써 검증하게 되고, put 파일은 파일을 보낼 때 사용하는 명령어 인데 이번 프로제트에서는 client 에 있는 파일을 server 쪽으로 보내서 동작을 수행하는 방식으로 검증을 수행하게 된다.

Get 과 put 의 동작과정을 살펴보면 다음과 같다. Ls command 와 같이 먼저 PORT command 를 client 쪽에서 연결해 , port 를 연결한다. 이때 port 를 연결할 때에는 ls 에서 했던 것과 마찬가지로, client 에서 임의로 난수를 만들어서 address ip 와 port number 를 생성하게 되며, 이렇게 만들어진, address 와 port number 를 바탕으로 server 에서는 이를 토대로 data connection 을 연결하게 된다. Data connection 에 담길 정보로는 파일의 내용으로 put command 수행 시에는 파일 내용을 data connection 을 통해서 보내는 과정을 수행하게 되고, get command 수행 시에는 받을 파일 내용을 data connection 을 통해서 가져오는 과정을 수행하게 된다.

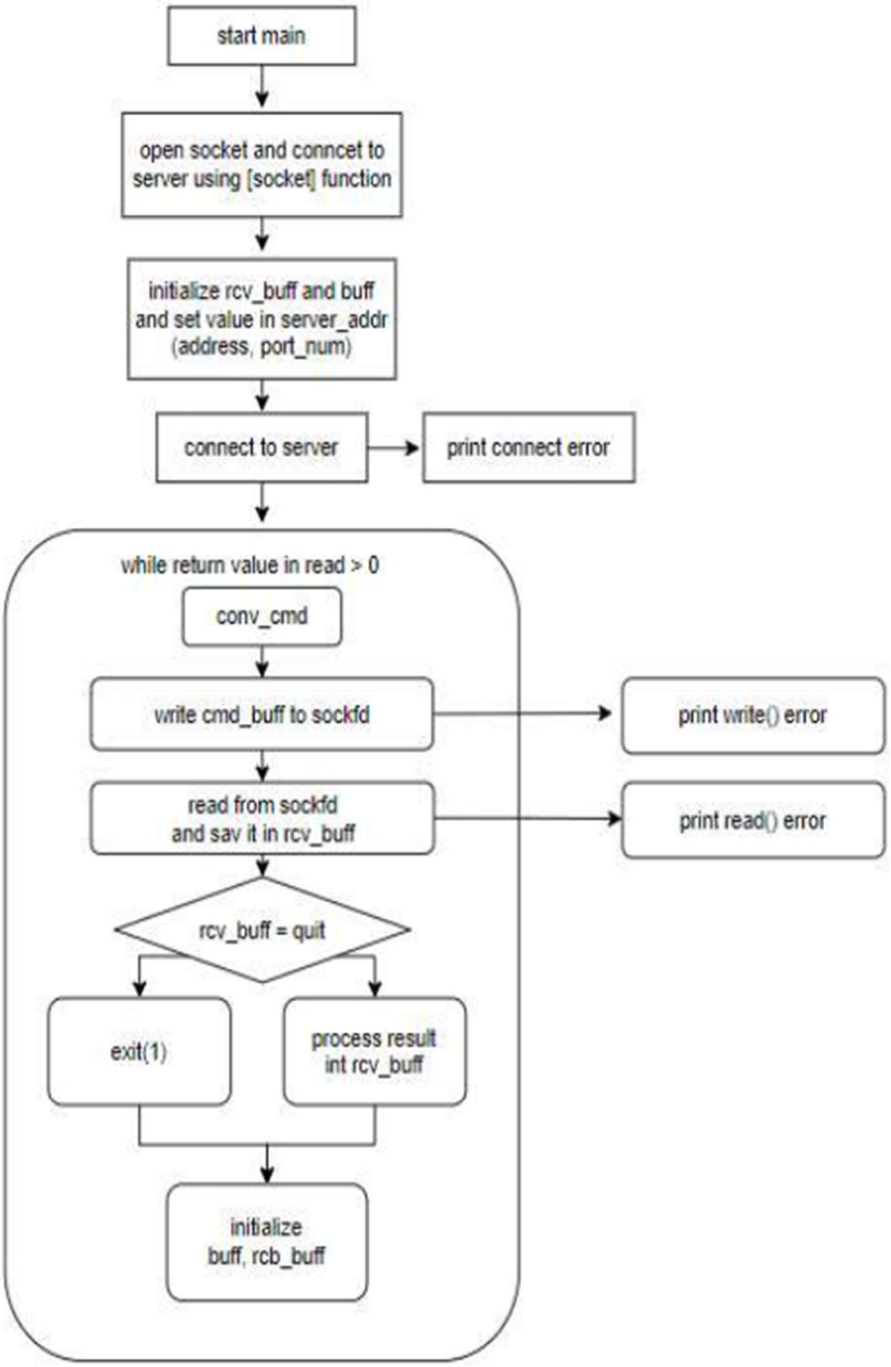
Flow chart

< cli.c >

Convert_command

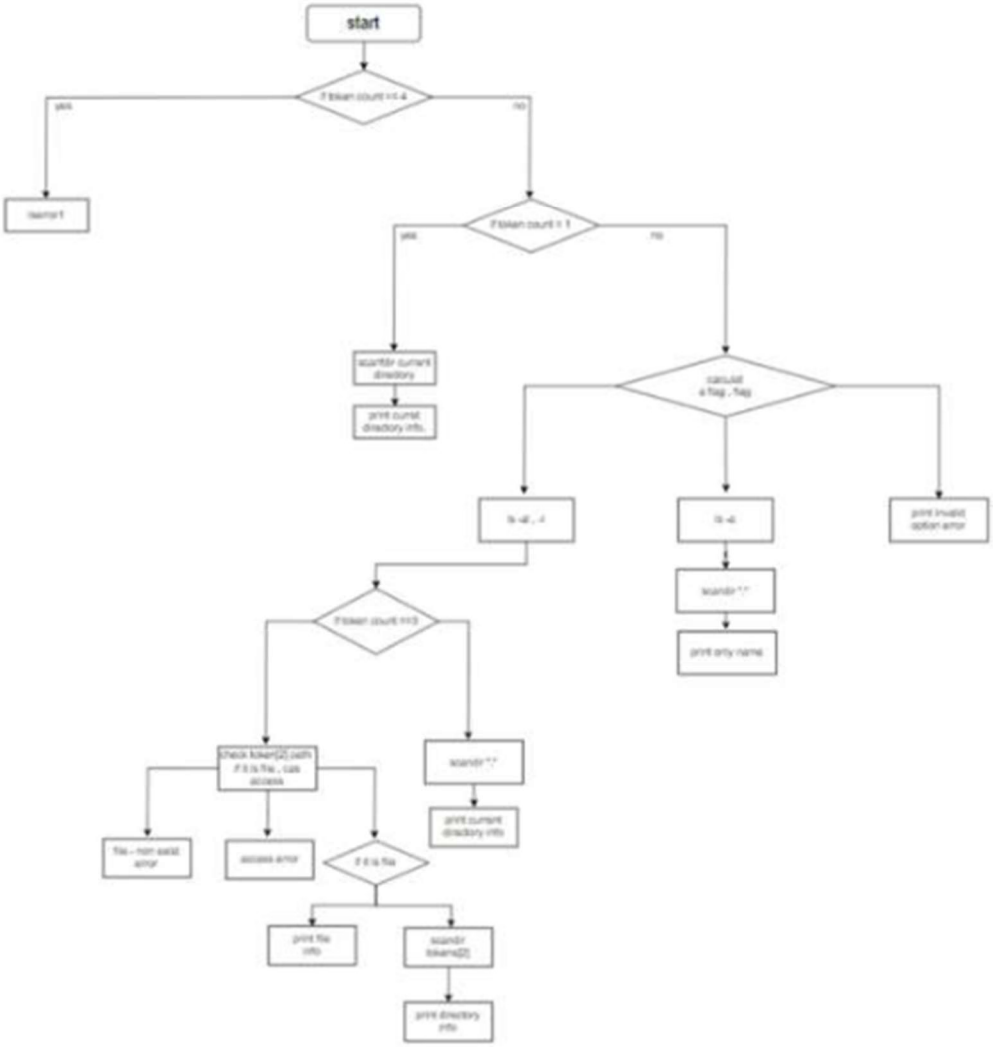


[main]

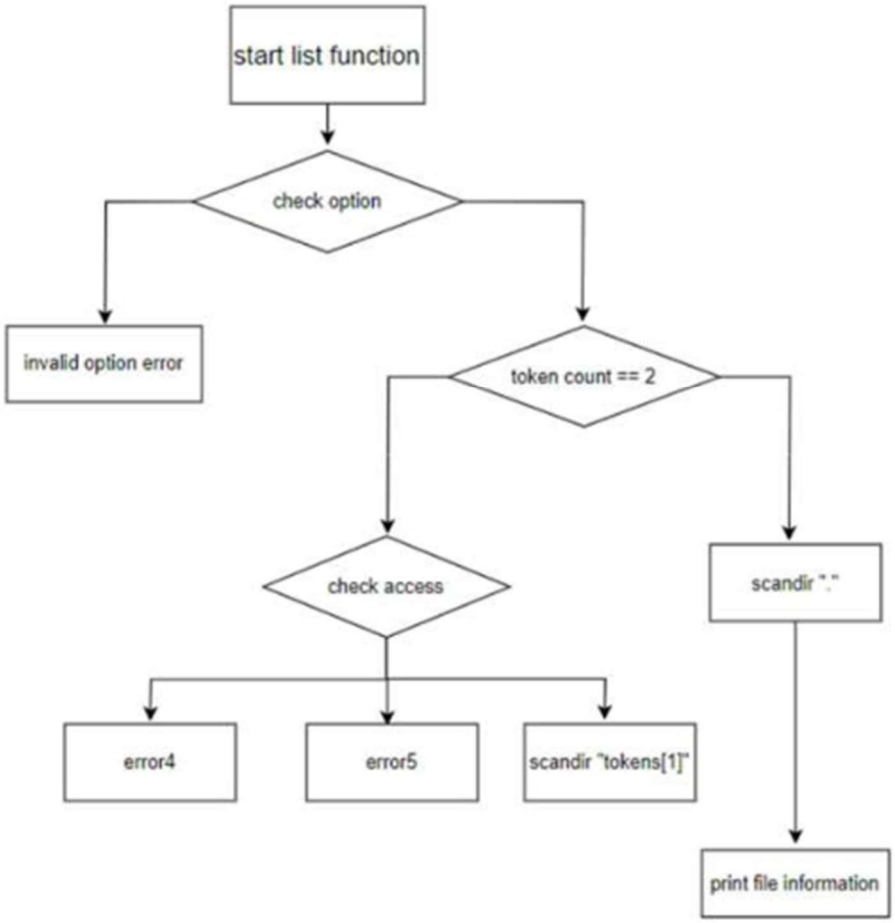

start main
open socket and conncet to server using [socket] function
initialize rcv_buff and buff and set value in server_addr (address, port_num)
connect to server
print connect error
while return value in read > 0
conv_cmd
write cmd_buff to sockfd
print write() error
read from sockfd and sav it in rcv_buff
print read() error
rcv_buff = quit
exit(1)
process result int rcv_buff
initialize buff, rcb_buff

< srv.c >

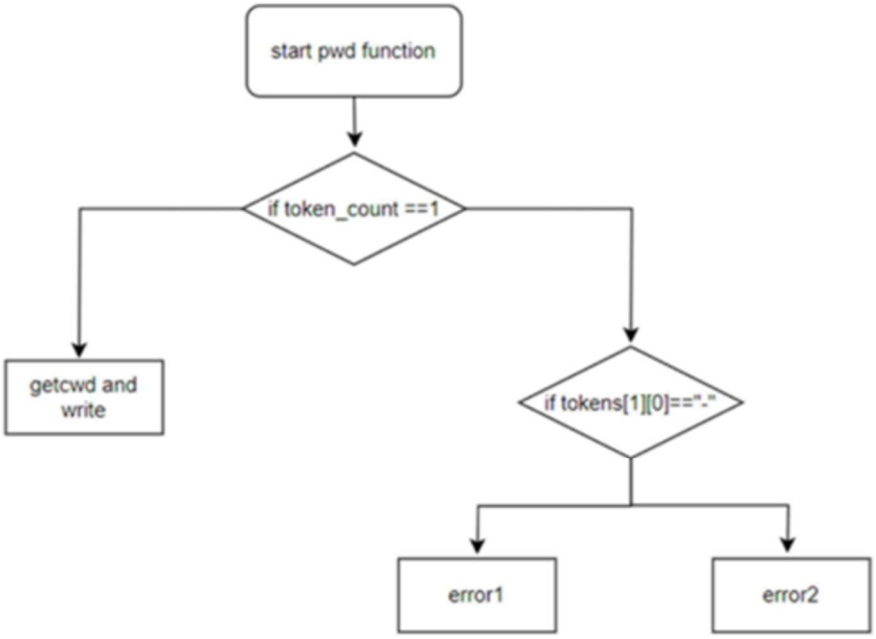
[NLST]


start
yes
no
scandir current directory
print current directory info.
print invalid option error
scandir "."
print only name
scandir "."
print current directory info
access error
print file info
print directory info

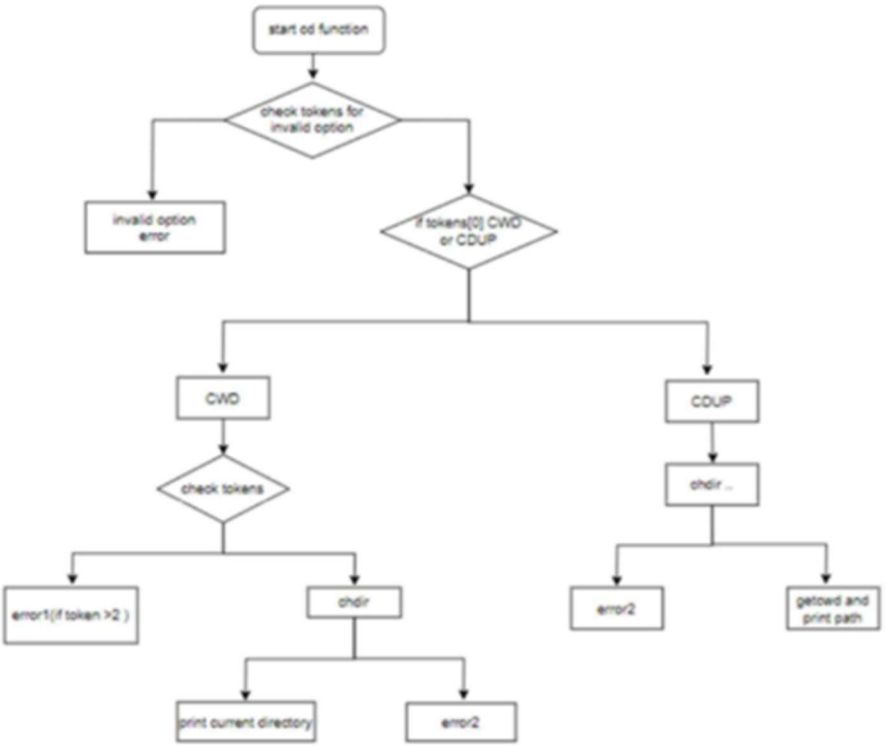
[DIR]


start list function
check option
invalid option error
token count == 2
check access
scandir "."
error4
error5
scandir "tokens[1]"
print file information

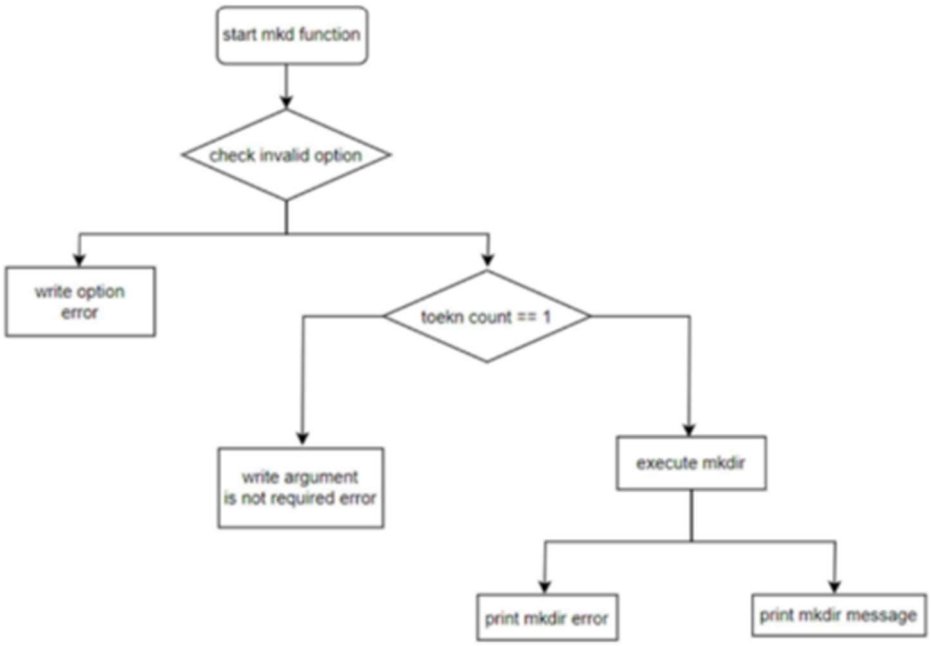
[PWD]


start pwd function
if token_count ==1
getcwd and write
if tokens[1][0]=="-"
error1
error2

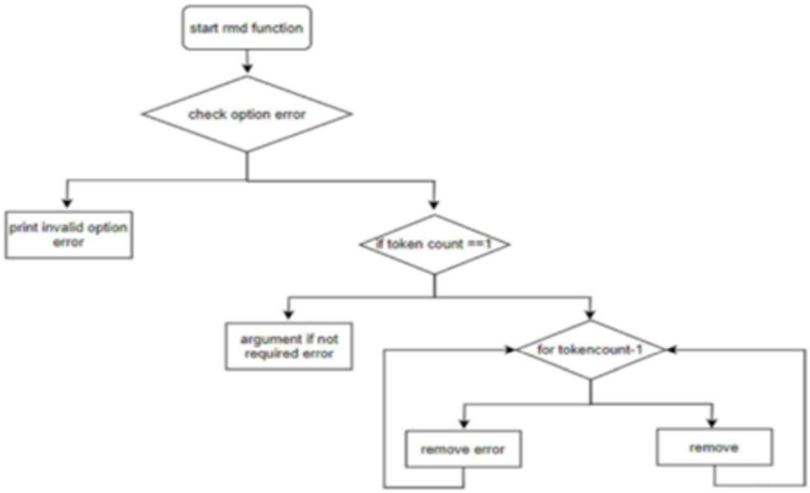
[CWD & CDUP]


start od function
check tokens for invalid option
invalid option error
if tokens[0] CWD or CDUP
CWD
CDUP
check tokens
chdir ..
error1(if token >2)
chdir
error2
getcwd and print path
print current directory
error2

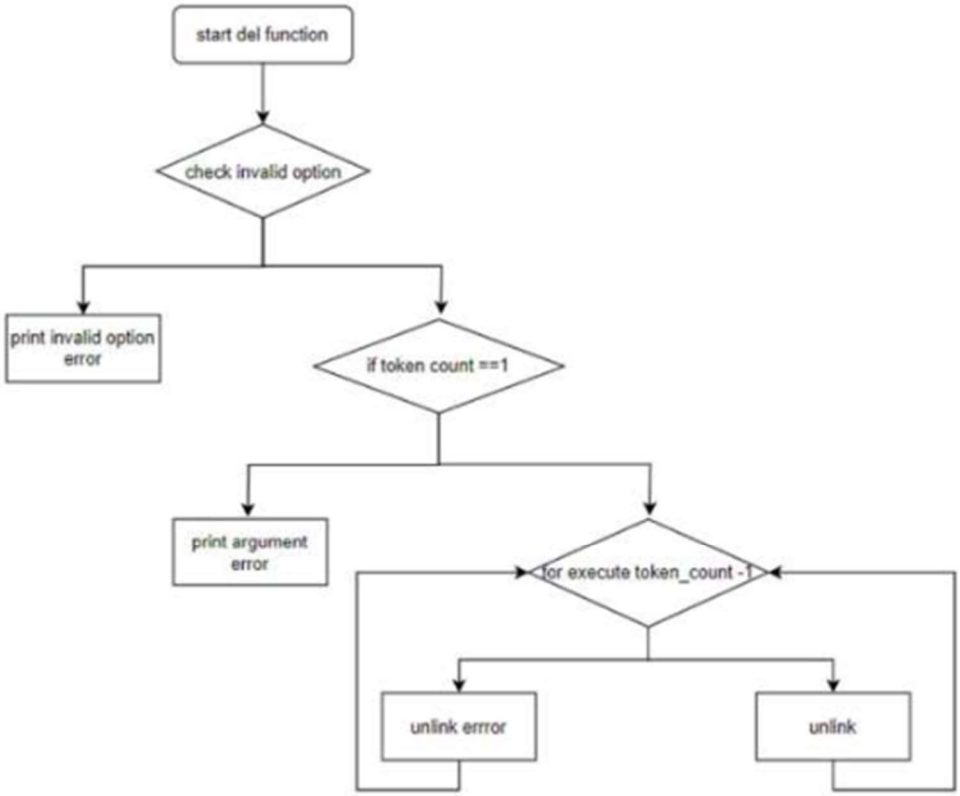
[MKDIR]


start mkd function
check invalid option
write option error
toekn count == 1
write argument is not required error
execute mkdir
print mkdir error
print mkdir message

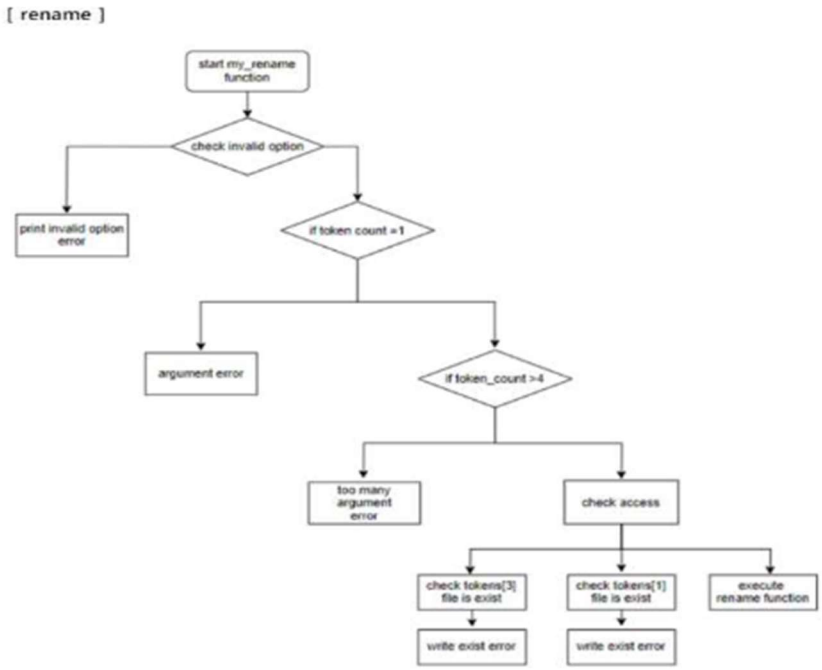
[RMD]


start rmd function
check option error
print invalid option error
if token count ==1
argument if not required error
for tokencount-1
remove error
remove

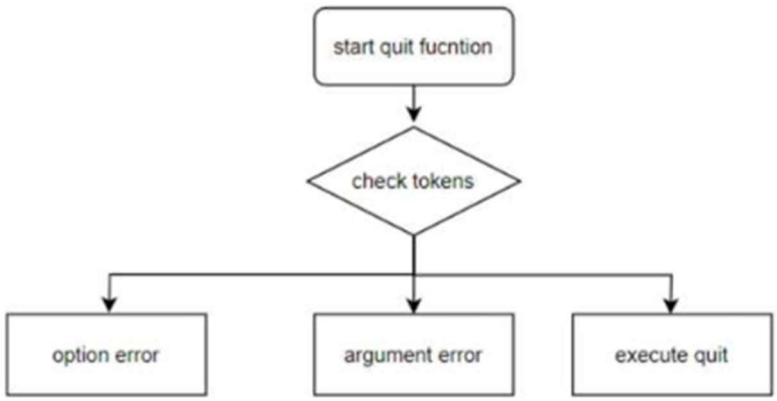
[DEL]


start del function
check invalid option
print invalid option error
if token count ==1
print argument error
for execute token_count -1
unlink errror
unlink

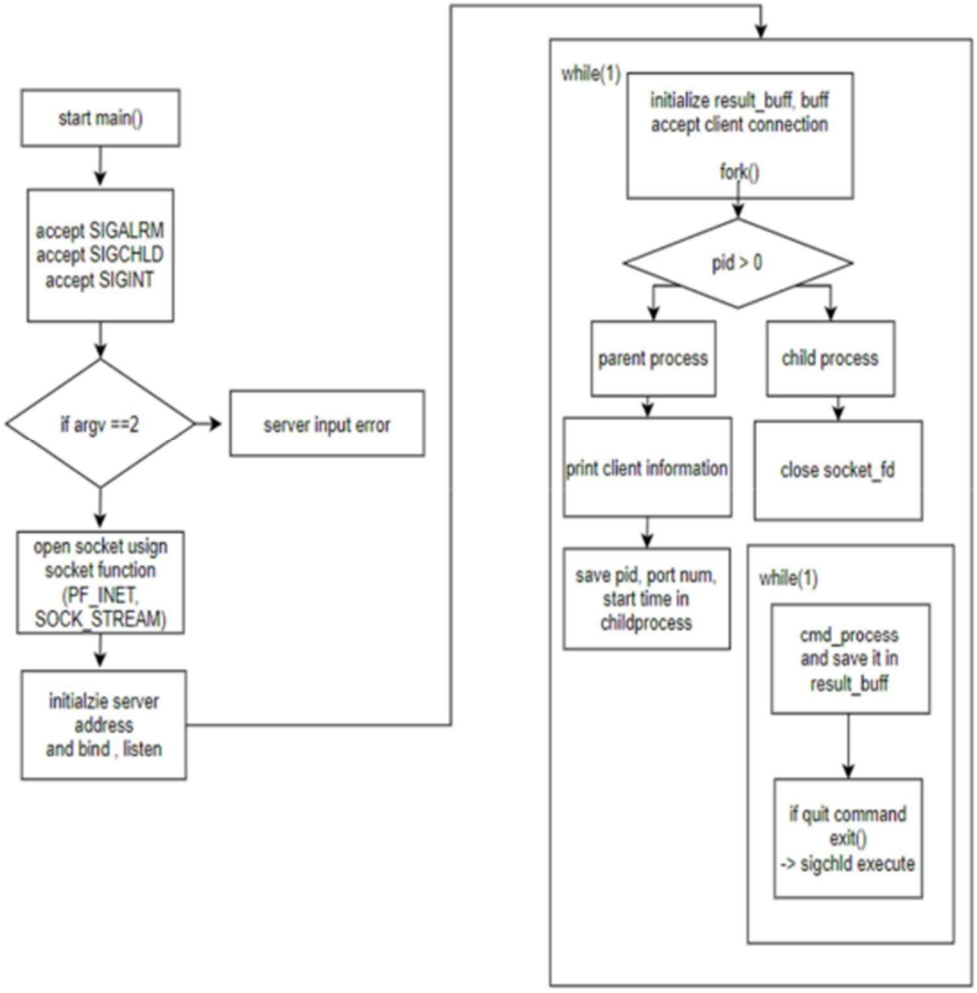
[RENAME]


[rename]
start my_rename function
check invalid option
print invalid option error
if token count =1
argument error
if token_count >4
too many argument error
check access
check tokens[3] file is exist
check tokens[1] file is exist
execute rename function
write exist error
write exist error

[QUIT]


start quit fucntion
check tokens
option error
argument error
execute quit

[main]


start main()
accept SIGALRM
accept SIGCHLD
accept SIGINT
if argv ==2
server input error
open socket usign
socket function
(PF_INET,
SOCK_STREAM)
initialzie server
address
and bind , listen
while(1)
initialize result_buff, buff
accept client connection
fork()
pid > 0
parent process
child process
print client information
close socket_fd
save pid, port num,
start time in
childprocess
while(1)
cmd_process
and save it in
result_buff
if quit command
exit()
-> sigchld execute

Pseudo code

[cli.c]

```
Function log_in(sockfd):


    Declare variables:

        n, user[MAX_BUFF], passwd[MAX_BUFF], buf[MAX_BUFF]

        user_command[MAX_BUFF], pass_command[MAX_BUFF]


    Initialize variables:

        Set all elements of user, passwd, buf to 0


    Read from socket:

        Read data into buf


    If buf starts with "431": // Rejection

        Print buf

        Exit the program


    Else if buf starts with "220": // Accepted

        Print buf


        Loop indefinitely:

            Prompt user for input: "Input ID : "

            Read user input into user
```

```
Construct USER command: "USER <user>"
Write USER command to socket

Read response from socket into buf

If buf starts with "530": // Username not accepted, exit
    Print buf
    Exit the program

If buf starts with "331": // Password required
    Print buf
    Loop indefinitely:
        Prompt user for input: "Input Password : "
        Read user input into passwd

        Construct PASS command: "PASS <passwd>"
        Write PASS command to socket

        Read response from socket into buf

        If buf starts with "530": // Password not accepted, exit
            Print buf
            Exit the program
```

Else if buf starts with "230": // Login successful

Print buf

Return from the function

Else if buf starts with "430": // Invalid username or password

Print buf

End of Function

Function main(argc, argv):

Initialize random number generator with current time

Declare variables:

buff[MAX_BUFF], cmd_buff[MAX_BUFF], rcv_buff[RCV_BUFF], control_buff[RCV_BUFF], command[MAX_BUFF]

server_addr

control_sock, n, len, len_out

data_port, data_sock, data_addr

port_command[MAX_BUFF], convert_command[MAX_BUFF]

addr_str, ack[MAX_BUFF]

Create a TCP socket:

```
    control_sock = socket(PF_INET, SOCK_STREAM, 0)
    If control_sock is -1, display error and exit

Initialize server address structure:
    Set server_addr fields: sin_family, sin_addr.s_addr, sin_port
    Convert address and port to network byte order

Connect to the server:
    If connection fails, display error and exit
    Print message: "Connected to Server"

Call log_in(control_sock) // Perform login

Prompt user for input:
    Write "ftp> " to standard output
    Loop while reading input from user:

        Read user input into buff

        If buff starts with "quit":
            Write "QUIT" to control_sock
            Read server response into control_buff
            Write control_buff to standard output
            Close control_sock
```

Exit program

Else if buff starts with "ls":

Make data connection

Accept data connection

Convert command from buff to cmd_buff

Write cmd_buff to control_sock

Read server response into control_buff

Write control_buff to standard output

Read data from data_connection into rcv_buff

Process and print received data

Read final server response into control_buff

Write control_buff to standard output

Print number of bytes received

Close data_connection

Close data_sock

// Similar logic for other commands like "pwd", "cd", "delete", "rename", "mkdir", "rmdir", "get", "put", "bin", "ascii"

End of Function

Function make_data_connection(control_sock, port, ip_addr):

Declare variables:

data_sock, data_addr, port_command[MAX_BUFF], convert_command[MAX_BUFF], addr_str, ack[MAX_BUFF]

Generate random port number between 10001 and 30000

Create data socket:

data_sock = socket(PF_INET, SOCK_STREAM, 0)

If data_sock is -1, display error and exit

Initialize data address structure:

Set data_addr fields: sin_family, sin_addr.s_addr, sin_port

Bind data socket to data address:

If bind fails, display error and exit

Listen for incoming connections on data socket:

If listen fails, display error and exit

Convert data address and port to string for PORT command:

Convert address and port to string format

Construct PORT command

Send PORT command to server

```
        Read acknowledgment from server
        Print acknowledgment

    Return data_sock

End of Function

Function conv_cmd(buf, cmd_buf):

    Declare variables:
        origin, token

    Extract command from buf:
        Tokenize buf by spaces
        Determine command type based on first token
        Convert command to FTP command and store in cmd_buf

        While there are more tokens:
            Concatenate token to cmd_buf with space delimiter

End of Function
```

[srv.c]

```
DEFINE CONSTANTS AND ERROR MESSAGES

DEFINE GLOBAL VARIABLE is_binary INITIALIZE TO 1


FUNCTION error_handling(message)

    PRINT message TO STDERR

    EXIT(1)


FUNCTION main(argc, argv)

    DECLARE VARIABLES curTime, pLocal, control_sock, client_sock, control_addr,
client_addr, client_addr_size

    SET curTime TO CURRENT TIME

    SET pLocal TO LOCAL TIME OF curTime


    CREATE control_sock

    IF control_sock IS -1

        CALL error_handling("socket() error!")


    INITIALIZE control_addr

    SET control_addr FAMILY TO AF_INET

    SET control_addr ADDRESS TO INADDR_ANY

    SET control_addr PORT TO argv[1]


    BIND control_sock TO control_addr
```

```
IF BIND FAILS
    CALL error_handling("bind() error")

LISTEN ON control_sock
IF LISTEN FAILS
    CALL error_handling("listen() error")

WHILE TRUE
    ACCEPT CLIENT CONNECTION ON control_sock
    IF accept() RETURNS -1
        CALL error_handling("accept() error")

    PRINT "** Client is connected **"

    CONVERT client_addr TO STRING AND STORE IN client_ip

    IF checkAccess(client_ip) > 0
        SEND welcome_message TO client_sock
    ELSE
        SEND "431 This client can't access. Close the session" TO client_sock
        CLOSE client_sock
        EXIT(1)

    CALL log_auth(client_sock)
```

```
        IF fork() RETURNS 0
            CHILD PROCESS:
            CLOSE control_sock
            CALL handle_client(client_sock)
            CLOSE client_sock
            EXIT(0)
        ELSE
            PARENT PROCESS:
            CLOSE client_sock
            WAIT FOR CHILD PROCESS TO EXIT

    CLOSE control_sock

FUNCTION checkAccess(client_ip)
    PARSE client_ip INTO origin ARRAY

    OPEN "access.txt" FOR READING
    IF FILE CANNOT BE OPENED
        PRINT "** File open error!! **"
        EXIT(1)

    WHILE READ LINE FROM FILE
        PARSE line INTO compare ARRAY
```

```
        COMPARE origin AND compare ARRAYS
        IF MATCH FOUND
            RETURN 1

    RETURN -1

FUNCTION log_auth(connfd)
    DECLARE VARIABLES information, user, passwd, count SET TO 1

    WHILE TRUE
        RECEIVE information FROM connfd

        IF information STARTS WITH "USER"
            PARSE information INTO user
            IF count >= 4
                SEND "530 Failed to log-in." TO connfd
                EXIT(1)
            IF user_match(user, 1) IS 1
                SEND "331 Password is required for user name." TO connfd
            ELSE
                SEND "430 Invalid username or password." TO connfd
                INCREMENT count

        ELSE IF information STARTS WITH "PASS"
```

```
            PARSE information INTO passwd
            IF count >= 4
                SEND "530 Failed to log-in." TO connfd
                EXIT(1)
            IF user_match(passwd, 0) IS 1
                SEND "230 User [user] logged in." TO connfd
                RETURN 0
            ELSE
                SEND "430 Invalid username or password." TO connfd
                INCREMENT count

FUNCTION user_match(information, is_username)
    OPEN "passwd" FOR READING
    IF FILE CANNOT BE OPENED
        PRINT "file open error!"
        EXIT(1)

    IF is_username IS 1
        WHILE READ LINE FROM FILE
            IF information MATCHES USERNAME IN LINE
                RETURN 1
    ELSE
        WHILE READ LINE FROM FILE
            IF information MATCHES PASSWORD IN LINE
```

```
            RETURN 1

    CLOSE FILE
    RETURN 0

FUNCTION handle_client(control_sock)
    DECLARE VARIABLES buffer, control_buffer, result_buffer, port_command, len, len_out

    WHILE READ COMMAND INTO buffer FROM control_sock
        IF buffer STARTS WITH "QUIT"
            SEND msg5 TO control_sock
            CLOSE control_sock
            EXIT(1)

        IF buffer STARTS WITH "PORT"
            PARSE buffer INTO client_ip AND client_port
            CREATE data_sock
            IF data_sock IS -1
                CALL error_handling("data socket() error")

            INITIALIZE client_addr WITH client_ip AND client_port

            CONNECT data_sock TO client_addr
            IF CONNECT FAILS
```

```
        SEND msg1 TO control_sock
        EXIT(1)

    SEND msg0 TO control_sock
    RECEIVE control_buffer FROM control_sock

    IF control_buffer STARTS WITH "NLST"
        SEND msg2 TO control_sock
        CALL cmd_process(control_buffer, result_buffer, control_sock)
        IF cmd_process FAILS
            SEND msg4 TO control_sock
            CLOSE data_sock
        ELSE
            SEND result_buffer TO data_sock
            SEND msg3 TO control_sock
            CLOSE data_sock

    IF control_buffer STARTS WITH "RETR"
        PARSE control_buffer INTO filename
        OPEN filename
        SEND OPENING MESSAGE TO control_sock
        WHILE READ FROM FILE INTO result_buffer
            SEND result_buffer TO data_sock
        CLOSE FILE
```

```
            CLOSE data_sock
            SEND msg3 TO control_sock
            PRINT "OK. total_bytes_sent bytes sent."

        IF control_buffer STARTS WITH "STOR"
            PARSE control_buffer INTO filename
            RECEIVE SERVER RESPONSE TO RETR COMMAND
            IF RESPONSE IS "150"
                OPEN filename
                WHILE READ FROM data_sock INTO result_buffer
                    WRITE result_buffer TO FILE
                CLOSE FILE
                CLOSE data_sock
                PRINT "OK. total_bytes_received bytes received."

    IF buffer STARTS WITH "PWD"
        CALL cmd_process(buffer, result_buffer, control_sock)
        SEND result_buffer TO control_sock

    IF buffer STARTS WITH "CWD"
        CALL cmd_process(buffer, result_buffer, control_sock)
        SEND result_buffer TO control_sock

    IF buffer STARTS WITH "DELE"
```

```
            CALL cmd_process(buffer, result_buffer, control_sock)
            SEND result_buffer TO control_sock

        IF buffer STARTS WITH "RNFR"
            CALL cmd_process(buffer, result_buffer, control_sock)

        IF buffer STARTS WITH "MKD"
            CALL cmd_process(buffer, result_buffer, control_sock)
            SEND result_buffer TO control_sock

        IF buffer STARTS WITH "RMD"
            CALL cmd_process(buffer, result_buffer, control_sock)
            SEND result_buffer TO control_sock

        IF buffer STARTS WITH "TYPE I"
            SEND "201 Type set to I." TO control_sock
            SET is_binary TO 1

        IF buffer STARTS WITH "TYPE A"
            SEND "201 Type set to A." TO control_sock
            SET is_binary TO 0

FUNCTION cmd_process(buff, result_buff, control_sock)
    PARSE buff INTO tokens ARRAY
```

```
    IF tokens[0] IS "NLST"
        CALL nlst(tokens, token_count, result_buff)
    IF tokens[0] IS "PWD"
        CALL pwd(tokens, token_count, result_buff)
    IF tokens[0] IS "LIST"
        CALL list(tokens, token_count, result_buff)
    IF tokens[0] IS "CWD" OR "CDUP"
        CALL cd(tokens, token_count, result_buff)
    IF tokens[0] IS "MKD"
        CALL mkd(tokens, token_count, result_buff)
    IF tokens[0] IS "DELE"
        CALL del(tokens, token_count, result_buff)
    IF tokens[0] IS "RMD"
        CALL rmd(tokens, token_count, result_buff)
    IF tokens[0] IS "RNFR"
        CALL my_rename(tokens, token_count, result_buff, control_sock)
    IF tokens[0] IS "QUIT"
        SET result_buff TO "program quit!!"
    ELSE
        SET result_buff TO "invalid command"
        RETURN -1

FUNCTION print_file_info(file_path, result_buff)
    GET FILE STATS FOR file_path
```

```
    IF FILE STATS SUCCESSFUL
        FORMAT FILE PERMISSIONS INTO result_buff
        FORMAT NUMBER OF LINKS INTO result_buff
        FORMAT OWNER NAME INTO result_buff
        FORMAT GROUP NAME INTO result_buff
        FORMAT FILE SIZE INTO result_buff
        FORMAT DATE AND TIME INTO result_buff
        APPEND FILE NAME TO result_buff
    RETURN 1

FUNCTION nlst(tokens, token_count, result_buff)
    SET aflag AND lflag TO 0
    IF token_count >= 4
        SET result_buff TO "Error: Too many arguments"
        RETURN -1
    ELSE IF token_count == 1
        SCAN CURRENT DIRECTORY INTO name_list
        FOR EACH name IN name_list
            IF name DOES NOT START WITH '.'
                APPEND name TO result_buff
                IF name IS DIRECTORY
                    APPEND '/' TO result_buff
                APPEND '₩n' TO result_buff
        RETURN 1
```

```
DETERMINE OPTIONS FROM tokens
IF INVALID OPTION
    SET result_buff TO "Error: invalid option"
    RETURN -1

IF lflag IS 1
    IF token_count == 3
        CHECK ACCESS FOR tokens[2]
        IF NO ACCESS
            SET result_buff TO "Error: cannot access"
            RETURN -1
        IF FILE
            CALL print_file_info(tokens[2], result_buff)
            RETURN 1
        SCAN tokens[2] DIRECTORY INTO name_list
    ELSE
        SCAN CURRENT DIRECTORY INTO name_list

    FOR EACH name IN name_list
        IF aflag IS 0 AND name STARTS WITH '.'
            CONTINUE
        CALL print_file_info(name, result_buff)
    RETURN 1
```

```
    ELSE IF aflag IS 1
        SCAN CURRENT DIRECTORY INTO name_list
        FOR EACH name IN name_list
            APPEND name TO result_buff
        RETURN 1

FUNCTION pwd(tokens, token_count, result_buff)
    IF token_count == 1
        GET CURRENT WORKING DIRECTORY INTO cwd
        SET result_buff TO "257 cwd is current directory"
        RETURN 1
    ELSE IF tokens[1] IS OPTION
        SET result_buff TO "Error: invalid option"
        RETURN -1
    ELSE
        SET result_buff TO "Error: argument is not required"
        RETURN -1

FUNCTION list(tokens, token_count, result_buff)
    SCAN DIRECTORY INTO name_list
    FOR EACH name IN name_list
        CALL print_file_info(name, result_buff)
    RETURN 1
```

```
FUNCTION cd(tokens, token_count, result_buff)
    IF token_count == 1
        SET result_buff TO "Error: argument is required"
        RETURN -1
    IF tokens CONTAINS OPTION
        SET result_buff TO "Error: invalid option"
        RETURN -1
    IF tokens[0] IS "CWD"
        IF token_count > 2
            SET result_buff TO "Error: invalid option"
            RETURN -1
        IF chdir(tokens[1]) FAILS
            SET result_buff TO "Error: cannot access"
            RETURN -1
        ELSE
            SET result_buff TO "250 CWD command succeeds"
    ELSE IF tokens[0] IS "CDUP"
        IF chdir("..") FAILS
            SET result_buff TO "Error: cannot access"
            RETURN -1
        ELSE
            SET result_buff TO "250 CWD command succeeds"
    RETURN 1
```

```
FUNCTION mkd(tokens, token_count, result_buff)
    IF token_count == 1
        SET result_buff TO "Error: argument is required"
        RETURN -1
    FOR EACH token IN tokens
        IF token IS OPTION
            SET result_buff TO "Error: invalid option"
            RETURN -1
    FOR EACH token IN tokens[1:]
        CREATE DIRECTORY token
        IF FAILS
            SET result_buff TO "Error: cannot create directory"
        ELSE
            SET result_buff TO "250 MKD command performed successfully"
    RETURN 1

FUNCTION del(tokens, token_count, result_buff)
    IF token_count == 1
        SET result_buff TO "Error: argument is required"
        RETURN -1
    FOR EACH token IN tokens
        IF token IS OPTION
            SET result_buff TO "Error: invalid option"
```

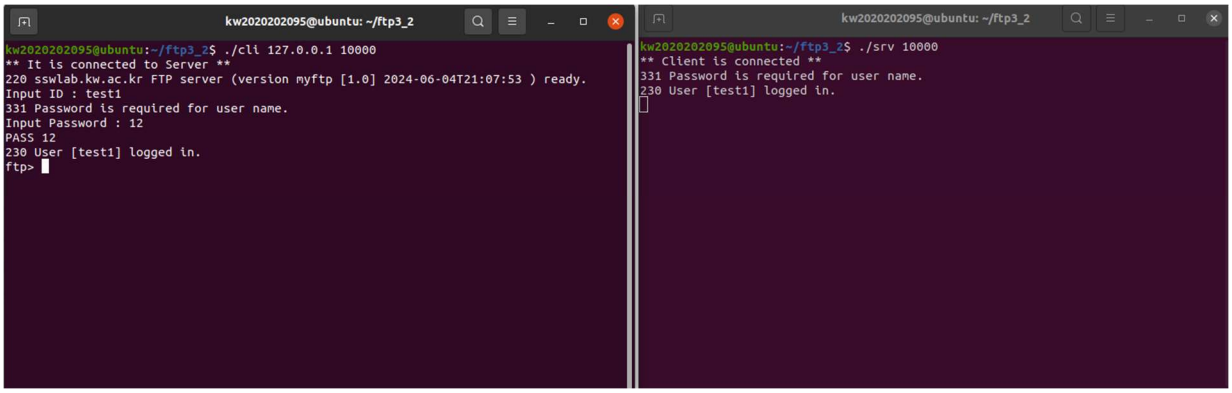
```
            RETURN -1
    FOR EACH token IN tokens[1:]
        DELETE FILE token
        IF FAILS
            SET result_buff TO "Error: failed to delete"
        ELSE
            SET result_buff TO "250 DELE command performed successfully"
    RETURN 1


FUNCTION rmd(tokens, token_count, result_buff)
    IF token_count == 1
        SET result_buff TO "Error: argument is required"
        RETURN -1
    FOR EACH token IN tokens
        IF token IS OPTION
            SET result_buff TO "Error: invalid option"
            RETURN -1
    FOR EACH token IN tokens[1:]
        DELETE DIRECTORY token
        IF FAILS
            SET result_buff TO "Error: failed to delete"
        ELSE
            SET result_buff TO "250 RMD command performed successfully"
    RETURN 1
```

```
FUNCTION my_rename(tokens, token_count, result_buff, control_sock)
    IF token_count == 1
        SET result_buff TO "Error: two arguments are required"
        RETURN -1
    IF token_count > 4
        SET result_buff TO "Error: too many arguments"
        RETURN -1
    CHECK ACCESS FOR tokens[1]
    IF NO ACCESS
        SET result_buff TO "Error: cannot access"
        RETURN -1
    SEND "350 File exists, ready to rename" TO control_sock
    IF NEW NAME EXISTS
        SET result_buff TO "Error: name to change already exists"
        RETURN -1
    RENAME tokens[1] TO tokens[3]
    SET result_buff TO "250 RNTO command succeeds"
    RETURN 1
```

결과화면

[login]



server 입력 설정 후 , client 에서 연결 요청 시 Welcome message 를 출력 하고 사용자로부터 ID 와 password 를 입력 받는다. 먼저 ID 를 제대로 입력 받았을 경우에는 password 가 필요하다고 client 에게 알리고, 사용자로부터 password 를 입력 받는다. 로그인에 성공했을 때에는 로그인에 성공했다고 사용자의 ID 와 함께 login 성공 message 를 출력한다.

[pwd]

“print working directory”

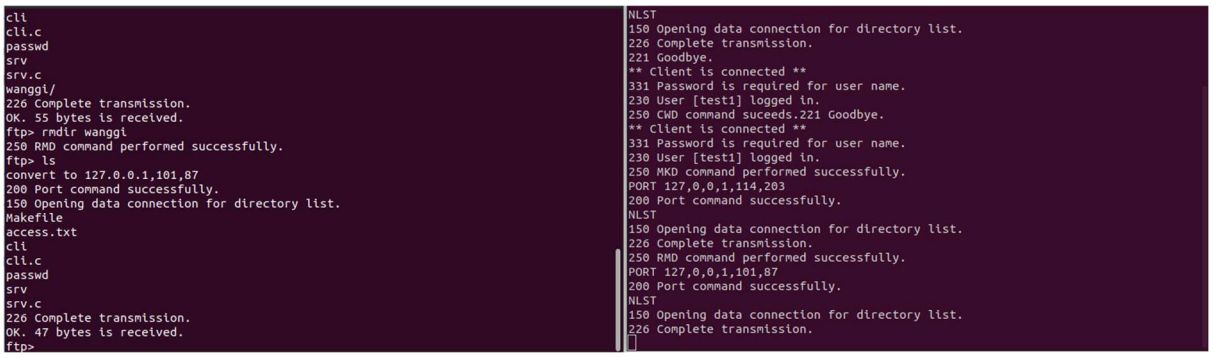
현재 directory 를 출력해주는 명령어로

/home/kw2020202095/ftp3_2 라는 현재 directory 를 출력해준다

[cd]

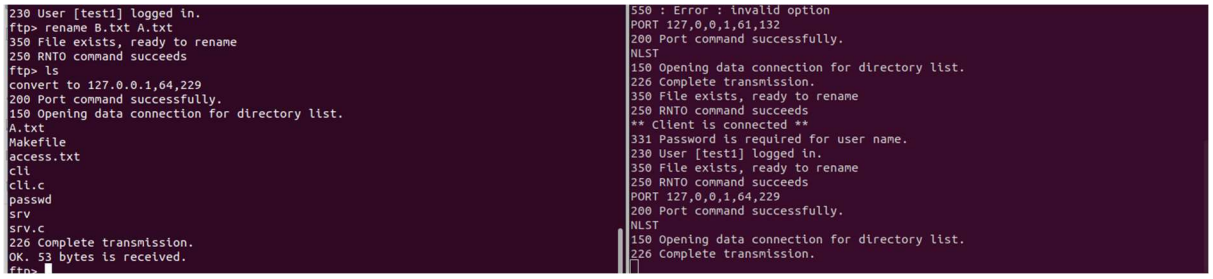
directory 의 위치를 변경하는 명령어로 cd 를 입력했을 경우 이전 directory 로 이동하는 것을 확인할 수 있고, cd + 경로 입력 시 입력한 경로로 이동하는 것을 확인 할 수 있다.

[mkdir , rmdir]



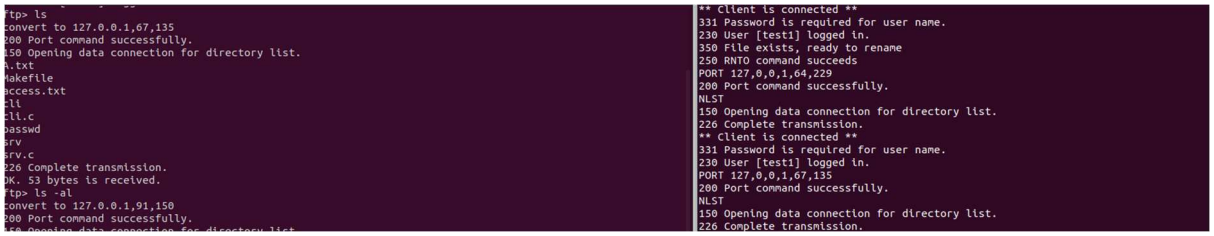
Mkdir 로 wanggi directory 를 만들고 rmdir 로 wanggi directory 로 지우는 과정을 확인할 수 있는데, ls 로 확인해본 결과, wanggi 라는 directory 가 만들어지고 , rmdir 수행 후 wanggi 라는 directory 가 지워진 것을 확인 할 수 있다.

[rename]



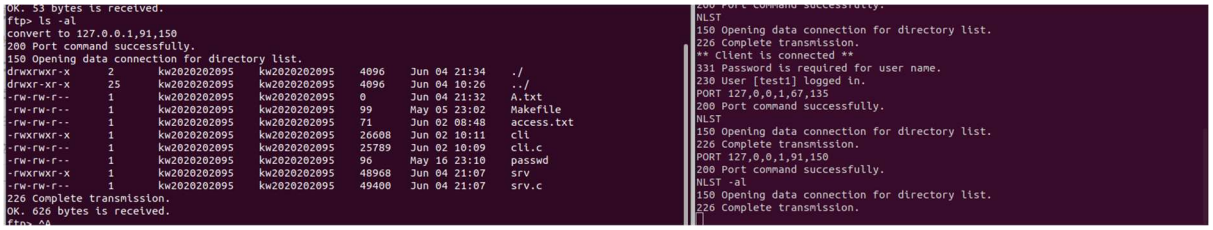
먼저 cmd 창에서 touch 명령어로 B.txt 를 만들고, client 연결을 통해서 rename command 의 동작과정을 검증하는 과정을 수행하였다. Rename B.txt A.txt 를 수행하면, 먼저 먼저 들어온 첫번째 인자로 들어온 파일이 있는지를 확인하고 , 350 File exists , ready to rename 이라는 command 를 전달하게 되고, 두 번째 인자로 들어온 file name 으로 파일 명을 변경하는 과정을 수행하게 된다. 이 결과, B.txt 파일이 A.txt 로 바뀐 것을 ls 명령어로 확인할 수 있다.

[ls]



Ls command 에 대한 테스트를 수행한 결과이다. Ls 명령어 입력 시 먼저 client 에서 PORT command 를 이용해 , server 에 전달하고, server 에서는 client 에서 만들어진 PORT command 를 받아, data connection 연결을 수행하는데 server 에서는 client 에서 랜덤으로 생성된 ip address 를 바탕으로 data connection 연결을 수행해, ls 의 결과를 data connection 을 이용해 client 로 보내게 된다. client 에서는 data connection 을 통해 받은 결과를 콘솔 창에 출력하게 되고, 전달받은 byte 의 수를 콘솔창에 출력해주게 된다.

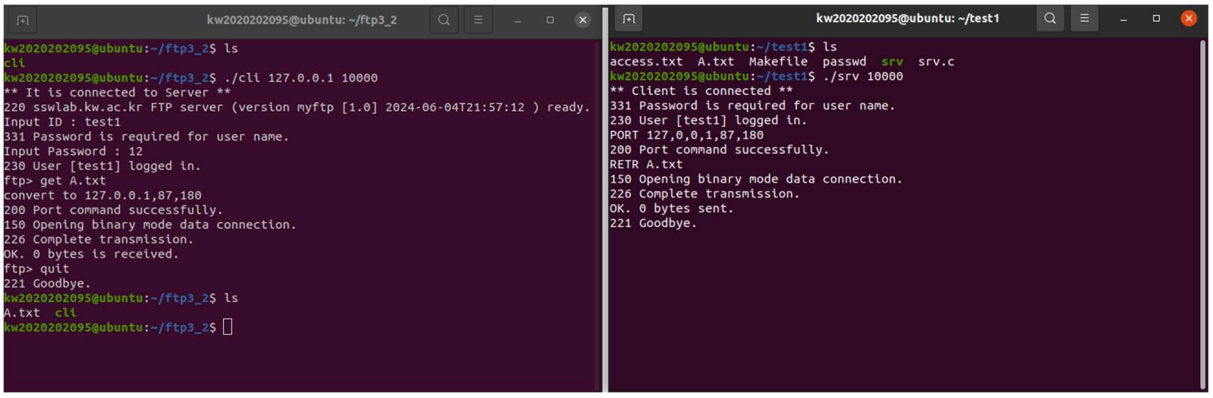
[ls -al]



Ls 명령어에 option 을 추가해 동작하는 결과를 확인 할 수 있다.

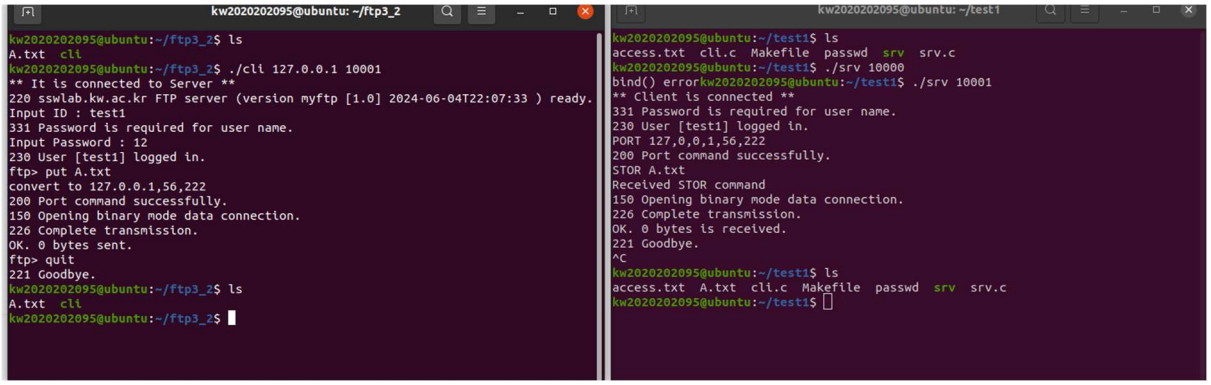
[ls – l]

[get]



Get command 를 확인하기 위해서 먼저 cli 파일과 srv 파일을 다른 경로에 위치해 두고, 테스트를 수행했다. Server 쪽의 path 에 있는 파일인 A.txt 를 get A.txt 명령어를 통해서, 가져오는 것을 확인 할 수 있다. (ls 명령어를 통해서 ~/ftp3_2 경로에 A.txt 가 있는 것을 확인 할 수 있다.)

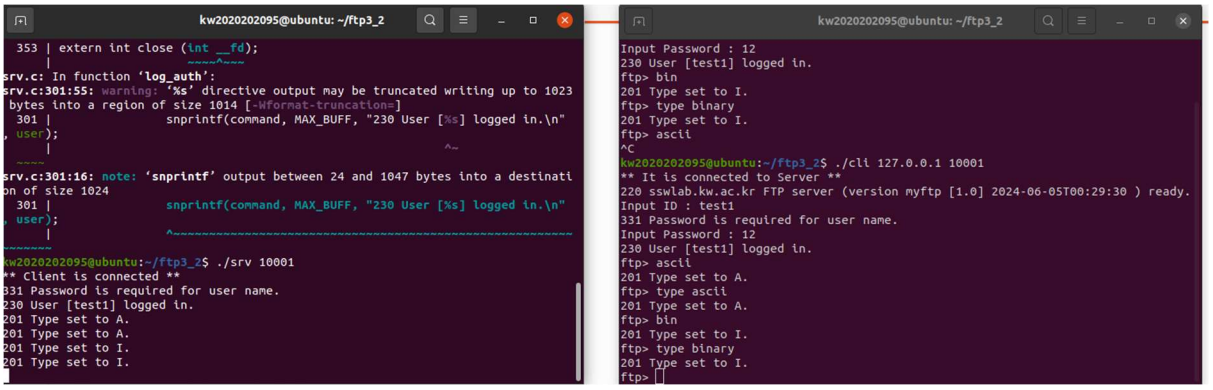
[put]



Put 명령어의 동작과정을 확인하기 위해서 cli 파일과 srv 파일을 다른 경로에 위치시키고,

Cli 실행파일이 있는 쪽에만, A.txt 파일을 넣어주고 put 명령어를 사용하여 srv 가 있는 곳에 A.txt 를 transmission 하는 과정을 검증하면 된다. client 쪽에서 put A.txt 명령어를 입력해, 검증을 진행하였다. Command 수행 후 , client 와 server 쪽에서 ls 명령어를 입력해본 결과 server 족에서 A.txt 파일이 제대로 전달 된 것을 확인 할 수 있다.

[type]



type command 를 확인하기 위해서 is_binary 변수를 전역변수로 설정하고, type binary 또는 bin 을 입력하면 binary 값이 1 이 바뀌도록 하였고, type ascii 또는 ascii 를 입력하면 binary 값이 0 으로 바뀌게 하였다.

[quit]

Quit 명령어 입력시 goodbye 메시지와 함께 client 가 종료되도록 구현하였다.

고찰

이번 프로젝트를 수행하면서, ftp server 와 유사하게, 실제 쓰이는 command 를 실제로 구현하고 검증해 봄으로써, 명령들에 대한 더 정확한 이해와 실제 명령어들을 어떻게 만드는지, 명령어들을 수행하게 하는 함수들은 어떤 함수들이 있는지 알 수 있게 되었다.

먼저 login 함수를 구현함으로 써 check 해야 할 요소인 accessable 한 client id 인지와 accesable 한 user name , accessable 한 user name 을 받았으면, 비밀번호를 사용자로부터 입력 받아 access.txt 파일에 있는 비밀번호화 비교해, 맞으면 로그인을 수행하는 과정을 통해 보안 및 접근 제어를 구현할 수 있었다.

다음으로 control_connection , data_connection 을 구현하여서 제어 연결 , 즉 각각의 command 마다 알맞은 번호를 보내면서 성공과 실패 시 그리고 중간중간 연결과정의 동작과정을 담을 제어 신호를 보냄으로써 client 와 server 가 상호작용하면서 동작할 수 있도록 하였고, 데이터 연결을 통해서 특정 command 가 수행한 결과 들은 data connection 을 통해서 연결함으로 써 구현할 수 있었다

Reference

-