시스템 프로그래밍 실습

Assignment2-1

Class : 금 1, 2 분반

Professor : 최상호 교수님

Student ID : 2020202031

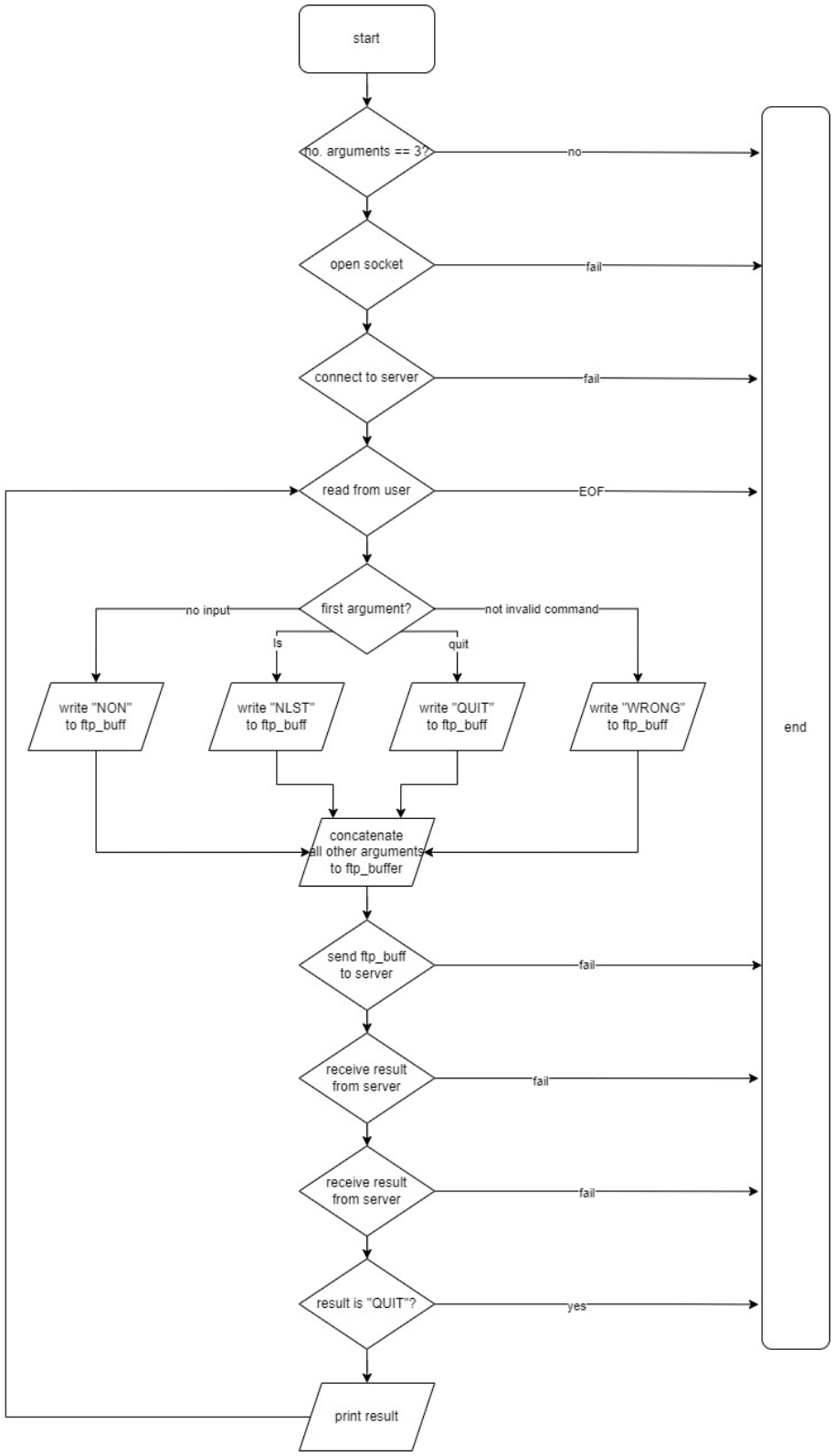
Name : 김재현

Introduction

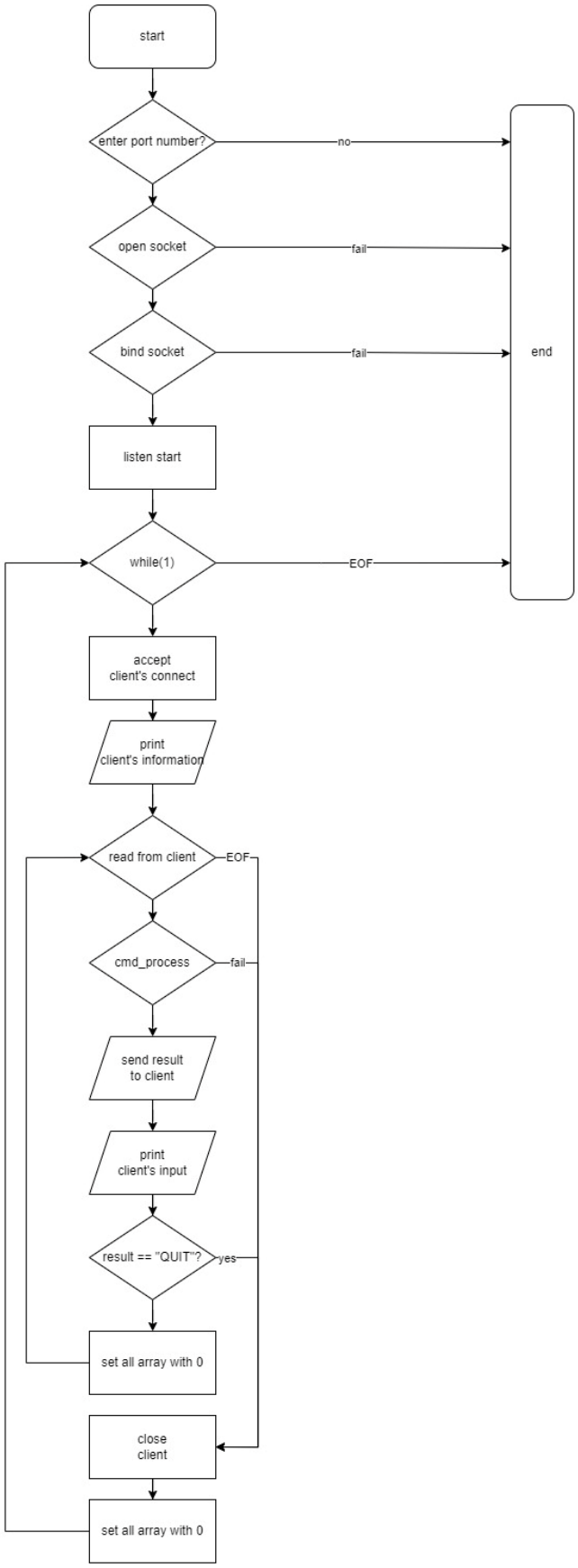
Socket 은 서로 다른 기기 간 네트워크 통신을 도구입니다. 이를 통해 클라이언트와 서버 간에 데이터를 주고받을 수 있으며, 이를 응용하여 다양한 기능을 구현할 수 있습니다. 이번 과제에서는 Socket Programming 을 활용하여, Client 에서 User command 를 FTP 명령어로 변환하여 Server 로 보내고, Server 에서 command 에 따른 연산을 처리 후 결과를 다시 Client 로 보내는 기능을 구현합니다. 리눅스의 기본 명령어 중 하나인 'ls' 명령어를 구현하는 것을 목표로 합니다.

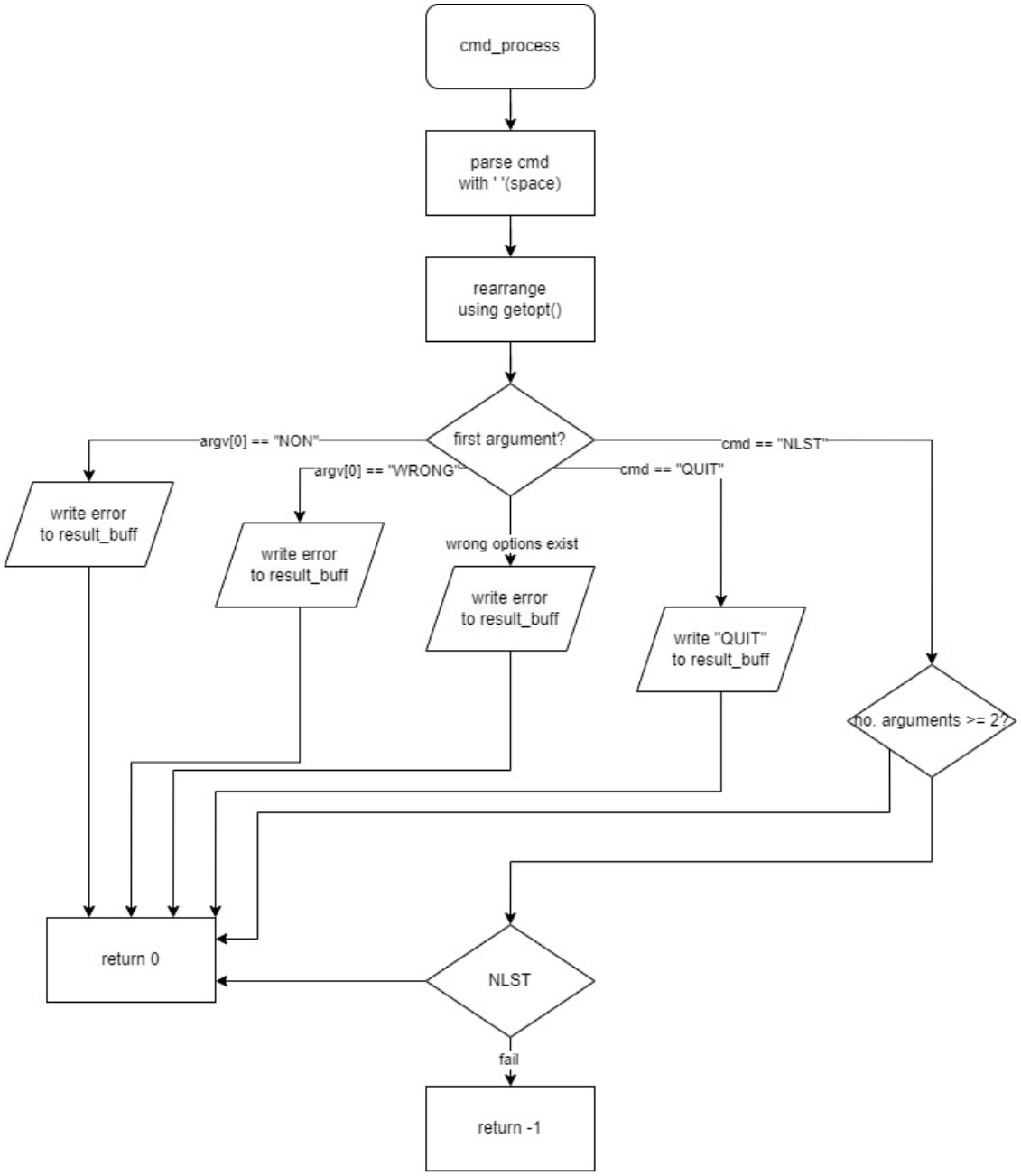
Flow chart

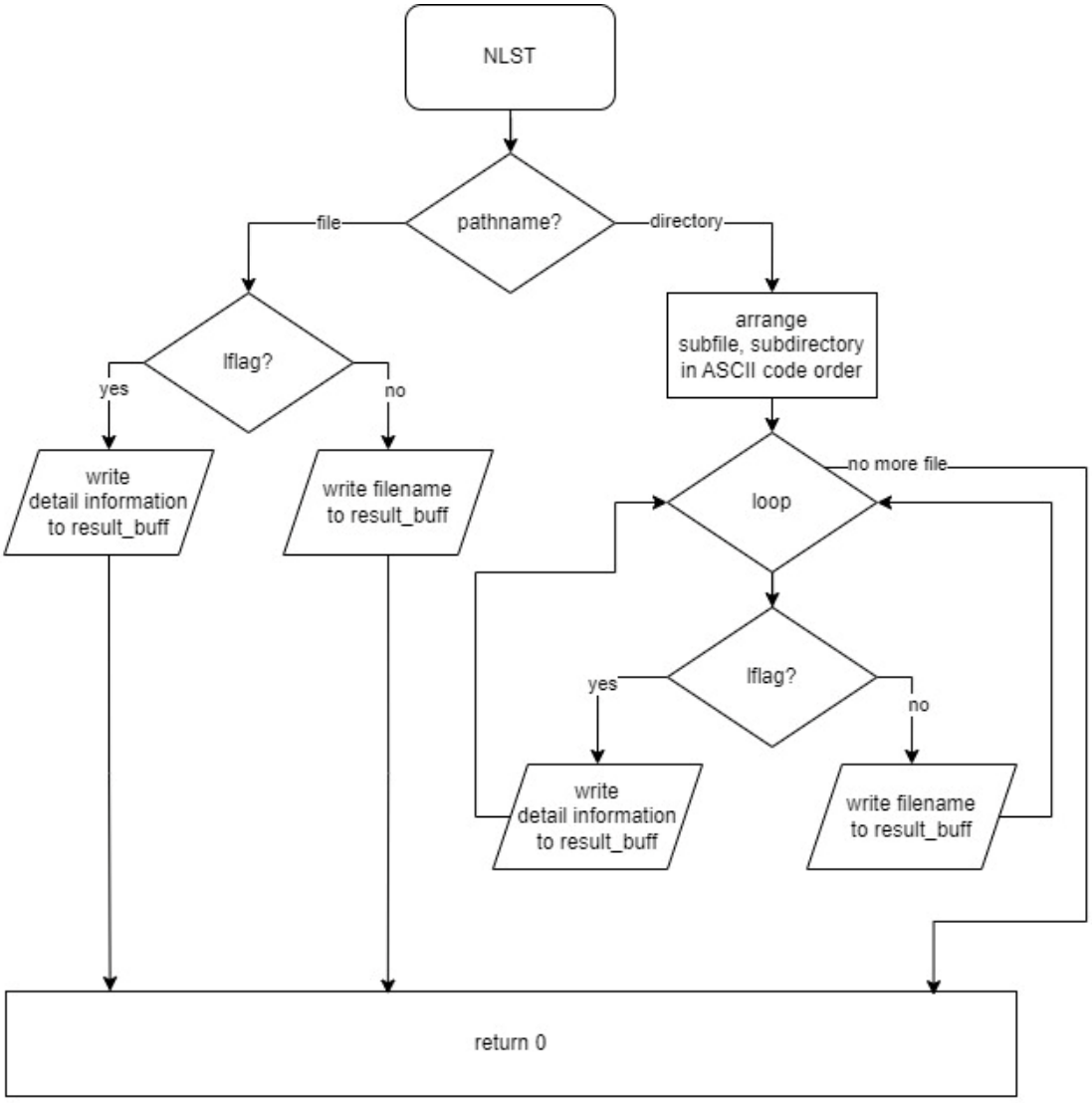
cli.c



srv.c


start
enter port number?
no
open socket
fail
bind socket
fail
end
listen start
while(1)
EOF
accept
client's connect
print
client's information
read from client
EOF
cmd_process
fail
send result
to client
print
client's input
result == "QUIT"?
yes
set all array with 0
close
client
set all array with 0


cmd_process
parse cmd
with ' '(space)
rearrange
using getopt()
first argument?
argv[0] == "NON"
argv[0] == "WRONG"
wrong options exist
cmd == "QUIT"
cmd == "NLST"
write error
to result_buff
write error
to result_buff
write error
to result_buff
write "QUIT"
to result_buff
no. arguments >= 2?
return 0
NLST
fail
return -1


NLST
pathname?
file
directory
lflag?
yes
no
write
detail information
to result_buff
write filename
to result_buff
arrange
subfile, subdirectory
in ASCII code order
loop
no more file
lflag?
yes
no
write
detail information
to result_buff
write filename
to result_buff
return 0

Pseudo code

cli.c

```
main
{
    check input arguments
    if(arguments don't fit in condition)
            print error;        exit(1)
    set arrays with 0
    open socket
    if(opening socket fails)
            print error;        exit(1)
    connect socket to server
    if(connecting fails)
            print error;        exit(1)
    while(read != EOF)
    {
            if(conv_cmd fails)
                    print error;        exit(1)
            write FTP command to server
            if(writing fails)
                    print error;        exit(1)
            read command's result from server
            if(reading fails)
                    print error;        exit(1)
```

```
        if(result is "QUIT")
                print "Program quit!!";   exit(1)
        print result
        set arrays with 0 again
    }
}

conv_cmd
{
        if(argument not entered)
                write "NON" to ftp buffer
        else if(first argument is ls)
                write "NLST" to ftp buffer
        else if(first argument is quit)
                write "QUIT" to ftp buffer
        else
                write "WRONG" to ftp buffer
        concatenate all other arguments to ftp buffer
}
```

srv.c

main

{

set arrays with 0

if(don't enter port number)

print error; return 0

open socket

if(opening fails)

print error; return 0

bind socket

if(binding fails)

print error; return 0

listen starts

while(1)

{

accept client's connect

print client's information

while(read from client)

{

if(cmd_process fails)

print error; break

print(client's input)

write the result of cmd_process to client

if(result is "QUIT")

```
                                    break
                            set arrays with 0
                    }
                    close client
                    set arrays with 0
            }
            close server
            return 0
}

cmd_process
{
            parse cmd_buff with ' '(space)
            rearrange buff using getopt()

            if(cmd is "NON")
                    write error to result_buff;          return 0
            if(cmd is "WRONG")
                    write error to result_buff;          return 0
            if(wrong options exist)
                    write error to result_buff;          return 0
            if(cmd is "QUIT")
                    write "QUIT" to result_buff;         return 0
            if(cmd is "NLST")
```

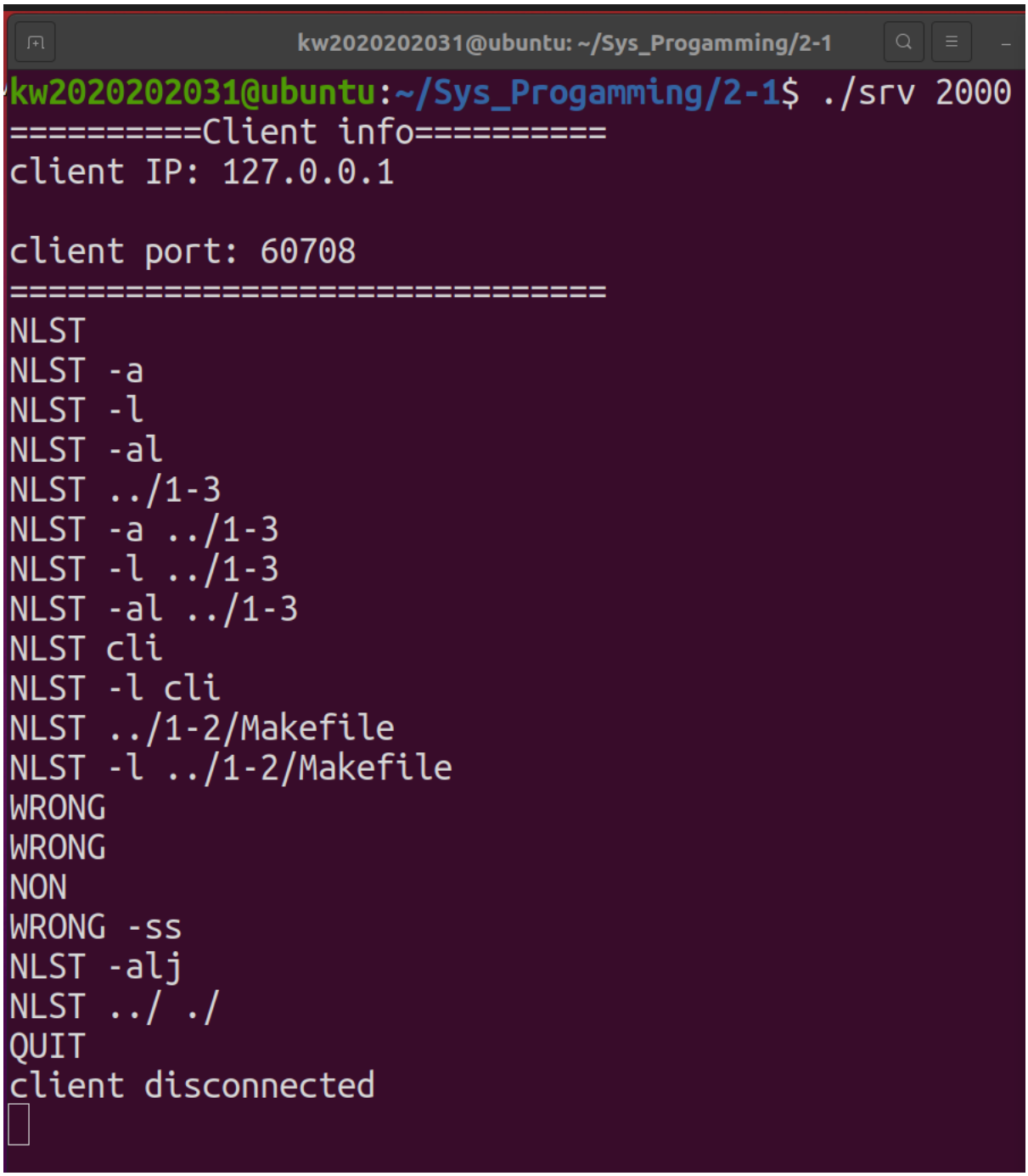
```
        {
                if(arguments >= 2)
                        write error to result_buff;          return 0
                make opflag with aflag, lflag
                if(NLST fails)
                        write error to result_buff;          return -1
        }
        return 0
}

NLST
{
        if(path is file)
        {
                if(opflag doesn't include lflag)
                        just write filename to result_buff
                if(opflag includes lflag)
                        write (File type, access permissions, number of links, owner and
                        group, file size, modification time, name, etc) to result_buff
        }
        if(path is directory)
        {
                make pointer array
                point all of subfile, subdirectory of path
                arrange in ASCII code order
                if(opflag doesn't include lflag)
```

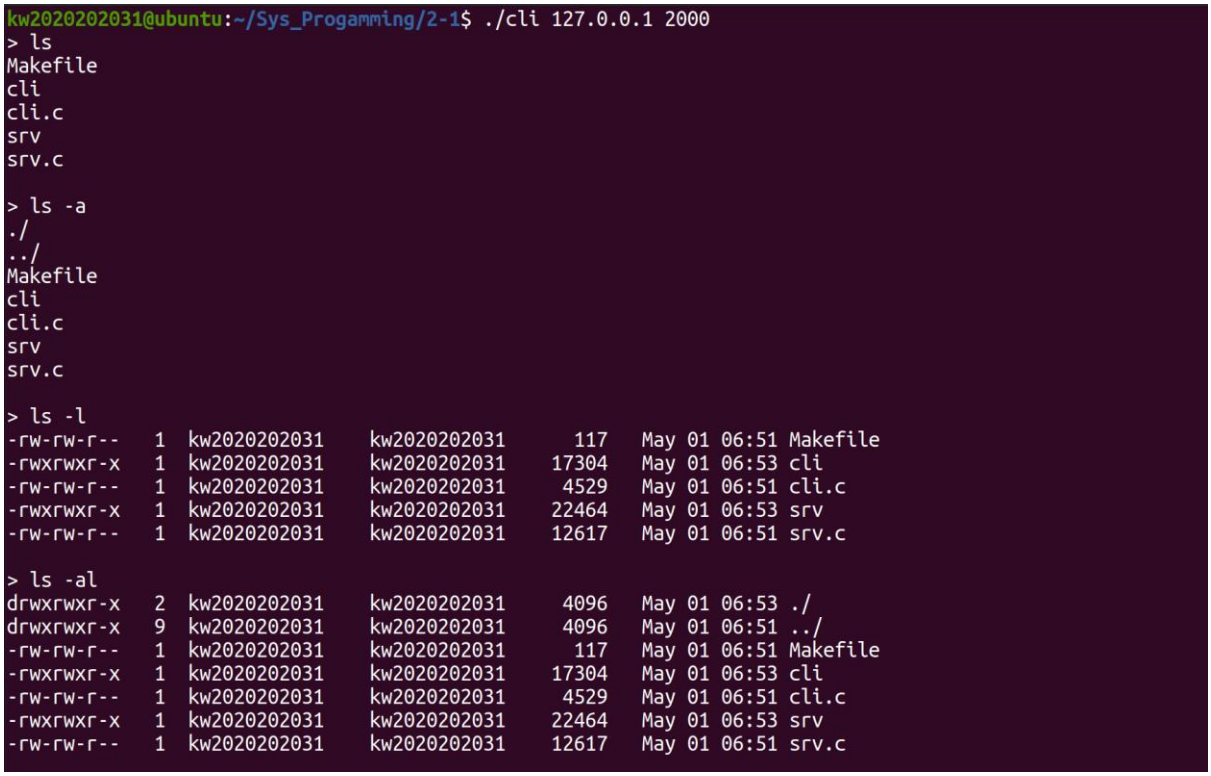
```
        {
            while(all)
            {
                if(aflag OFF && hidden)
                    continue
                just write the name of subfiles, subdirectory to result_buff
            }
        }

        if(opflag includes lflag)
        {
            while(all)
            {
                if(aflag OFF && hidden)
                    continue
                write (File type, access permissions, number of links, owner
                and group, file size, modification time, name, etc) of
                subfiles, subdirectory to result_buff
            }
        }
    }
    return 0
}
```

결과화면

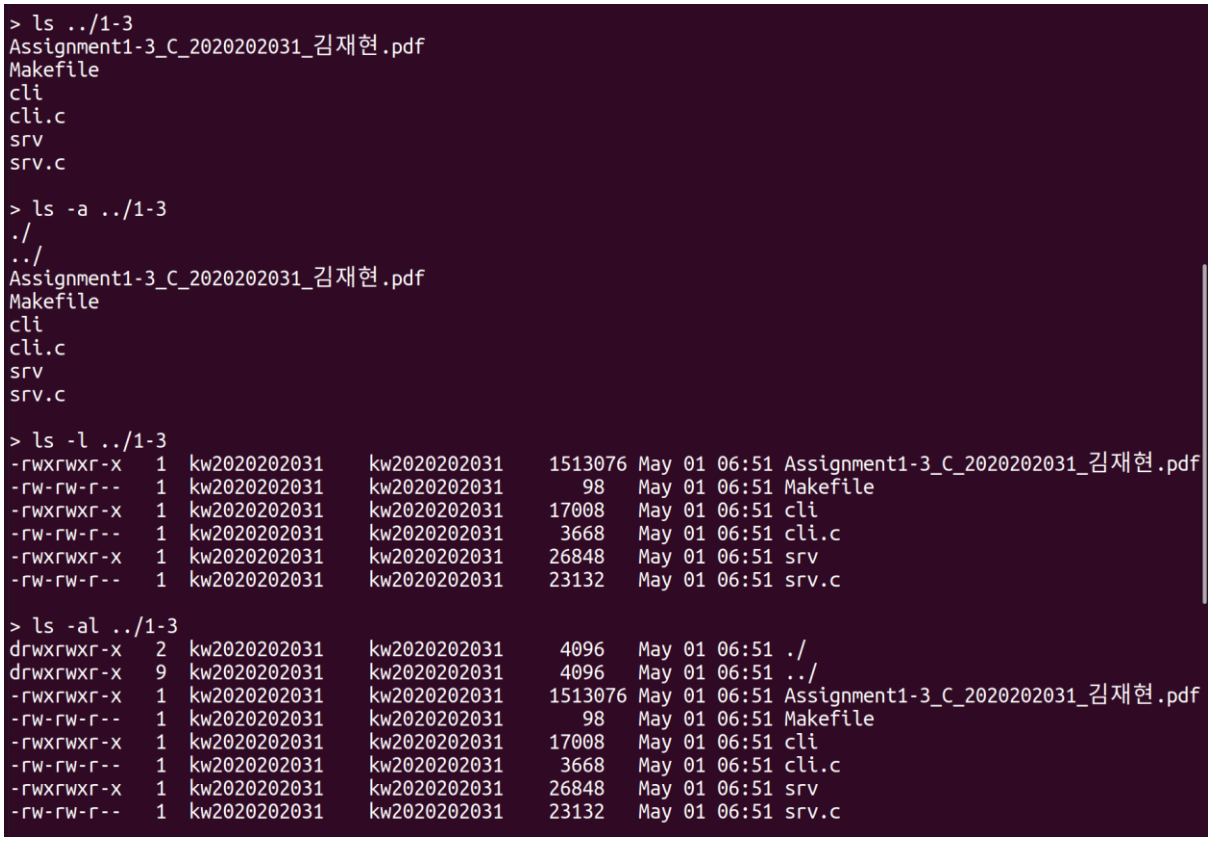


server 측 출력입니다. client 의 connect 를 accept 하는 순간 Client 의 IP 와 port 를 출력합니다. 또한 Client 에서 보내온 FTP command 를 그대로 출력합니다.

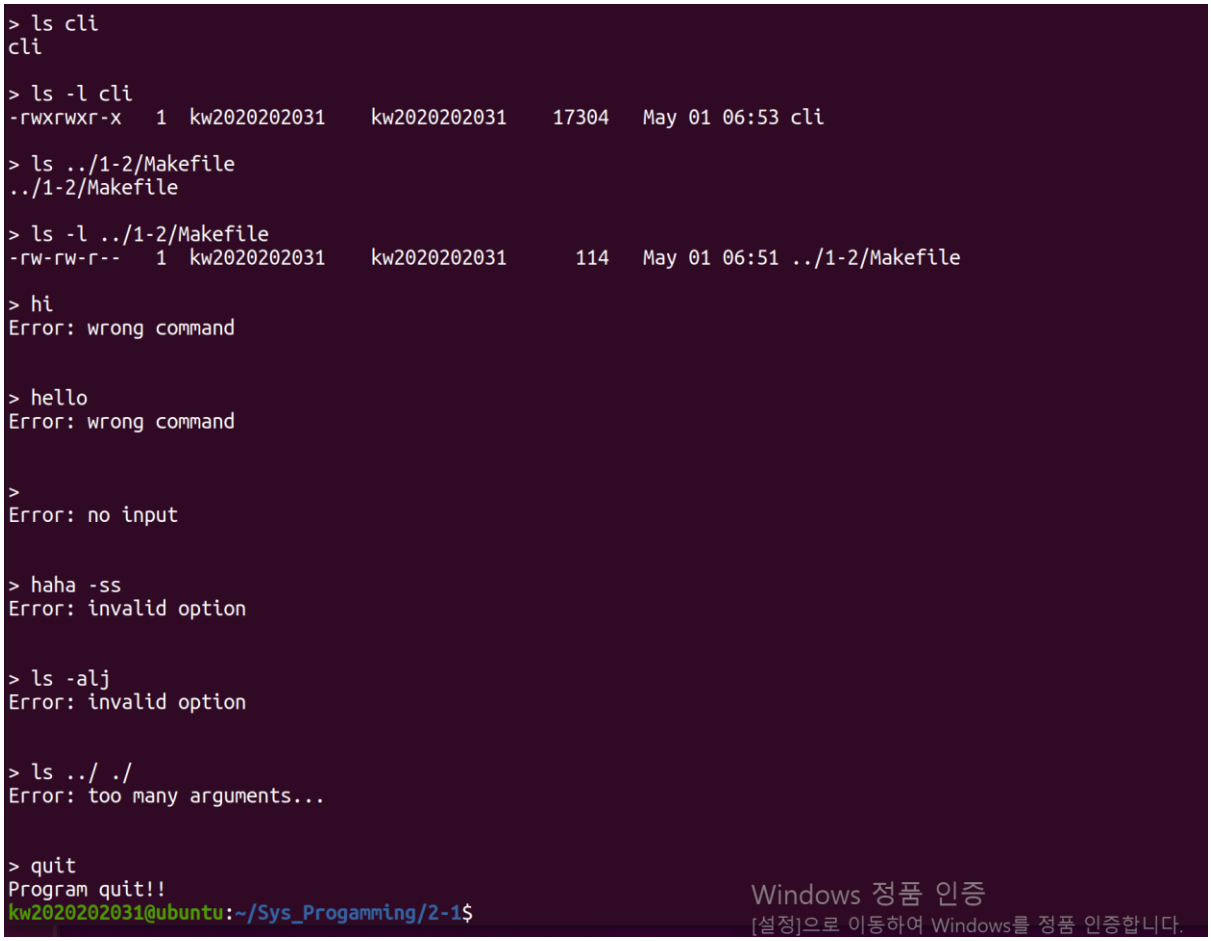


client 측 출력입니다. command 를 입력함에 따라 해당 command 를 FTP command 로 변환하여 server 로 전송하고 server 에서 결과물을 수신하여 출력합니다.

위 결과화면은 ls 에 -a, -l, -al 옵션을 줬을 때 결과가 잘 출력되는지 확인하기 위해 첨부했습니다. -a 이 추가되면 hidden file, hidden directory 도 함께 출력되는 것을 확인할 수 있습니다. -l 이 추가되면 파일유형, 접근권한, 링크 수, 소유자 및 그룹, 파일 크기, 수정 시간, 이름 등의 정보가 함께 출력되는 것을 확인할 수 있습니다.



위 결과화면 또한 ls 에 -a, -l, -al 옵션을 줬을 때 결과가 잘 출력되는지 확인하기 위해 첨부했습니다. 그러나 이번엔 현재 디렉토리가 아닌 상위 디렉토리의 하위 디렉토리를 인자로 주었습니다. 이번에도 마찬가지로 -a 이 추가되면 hidden file, hidden directory 도 함께 출력되는 것을 확인할 수 있습니다. -l 이 추가되면 파일유형, 접근권한, 링크 수, 소유자 및 그룹, 파일 크기, 수정 시간, 이름 등의 정보가 함께 출력되는 것을 확인할 수 있습니다.



위 결과화면 또한 ls 에 -a, -l, -al 옵션을 줬을 때 결과가 잘 출력되는지 확인하기 위해 첨부했습니다. 그러나 이번엔 현재 디렉토리 내의 파일, 상위 디렉토리의 하위 디렉토리 내의 파일을 인자로 주었습니다. 이번에도 마찬가지로 -l 이 추가되면 파일유형, 접근권한, 링크 수, 소유자 및 그룹, 파일 크기, 수정 시간, 이름 등의 정보가 함께 출력되는 것을 확인할 수 있습니다.

잘못된 명령어가 입력되면 "WRONG"이라는 문자열을 server 로 전송하고, server 에서 수신 후 "Error: wrong command"를 전송하면 client 는 그대로 출력합니다.

명령어가 입력되지 않았다면 "NON"이라는 문자열을 server 로 전송하고, server 에서 수신 후 "Error: no input:을 전송하면 client 는 그대로 출력합니다.

a, l 을 제외한 옵션이 입력된다면 server 에서 "Error: invalid option"이라는 문자열을 전송하고 client 는 그대로 출력합니다.

만일 ls 뒤에 2 개 이상의 인자가 입력된다면 server 에서 "Error: too many arguments..."이라는 문자열을 전송하고 client 는 그대로 출력합니다.

위와 같이 잘못된 경로를 입력하면 cmd_process 에서 오류를 발생하여 server 는 곧바로 client 와의 연결을 끊습니다. client 는 ctrl + C 로 대기상태를 빠져나갈 수 있습니다.

고찰

getopt() 함수는 argv 중 option 은 명령어 뒤에 위치시키고 non-option 은 option 뒤에 위치시키는 명령어입니다. getopt()에는 내부변수 optind, optarg, opterr, optopt 를 가집니다. getopt()가 역할을 마치면, optind 는 argv 를 재배열시킨 후 option 뒤에 따라오는 non-option 의 index 를 가리킵니다. 그리고 프로그램이 종료될 때까지 그 값을 유지합니다. 평상시에는 각 프로세스 당 getopt 를 한 번 호출하기에 큰 문제가 없습니다. 하지만 이번 과제에서는 cmd_process 함수를 구현할 때, getopt()함수를 사용하여, 함수가 호출되고 종료됨에 따라 getopt()가 초기화될 것으로 생각했습니다. 그러나 함수가 종료돼도, 프로그램은 종료되지 않았기에 optind 가 계속해서 이전 값을 유지하고 있었고, 새로운 명령어를 입력 받아 getopt()를 실행하니 segmentation fault 가 뜨는 것을 확인할 수 있었습니다. optind 가 이전 값을 유지하고 있으니, 새로운 command 에서는 유효하지 않은 주소에 접근하려고 하니 당연한 결과였습니다.

이에 cmd_process 함수의 시작 부분에 optind = 0 으로 초기화해주는 코드를 작성하여 segmentation fault 를 방지할 수 있었습니다.

Reference

시스템프로그래밍 / 광운대학교 / 최상호 교수님 / 6._SP_-_Sockets

시스템프로그래밍 / 광운대학교 / 최상호 교수님 / 2024-1_SPLab_07_FTP2_1_v1

시스템프로그래밍 / 광운대학교 / 최상호 교수님 / 2024-1_SPLab_FTP_Assginment2_1_v1