시스템 프로그래밍 실습

[Assignment #3-2: control & data connection]

Class : [금요일 1,2 교시]

Professor : [최상호 교수님]

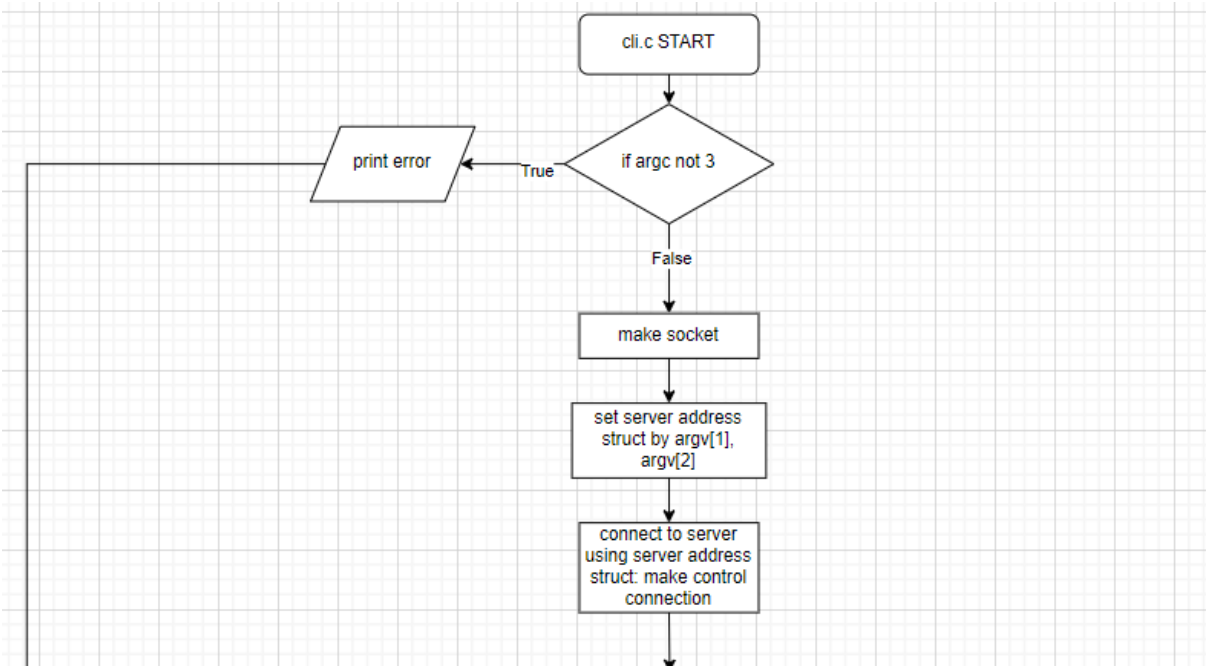
Student ID : [2020202034]

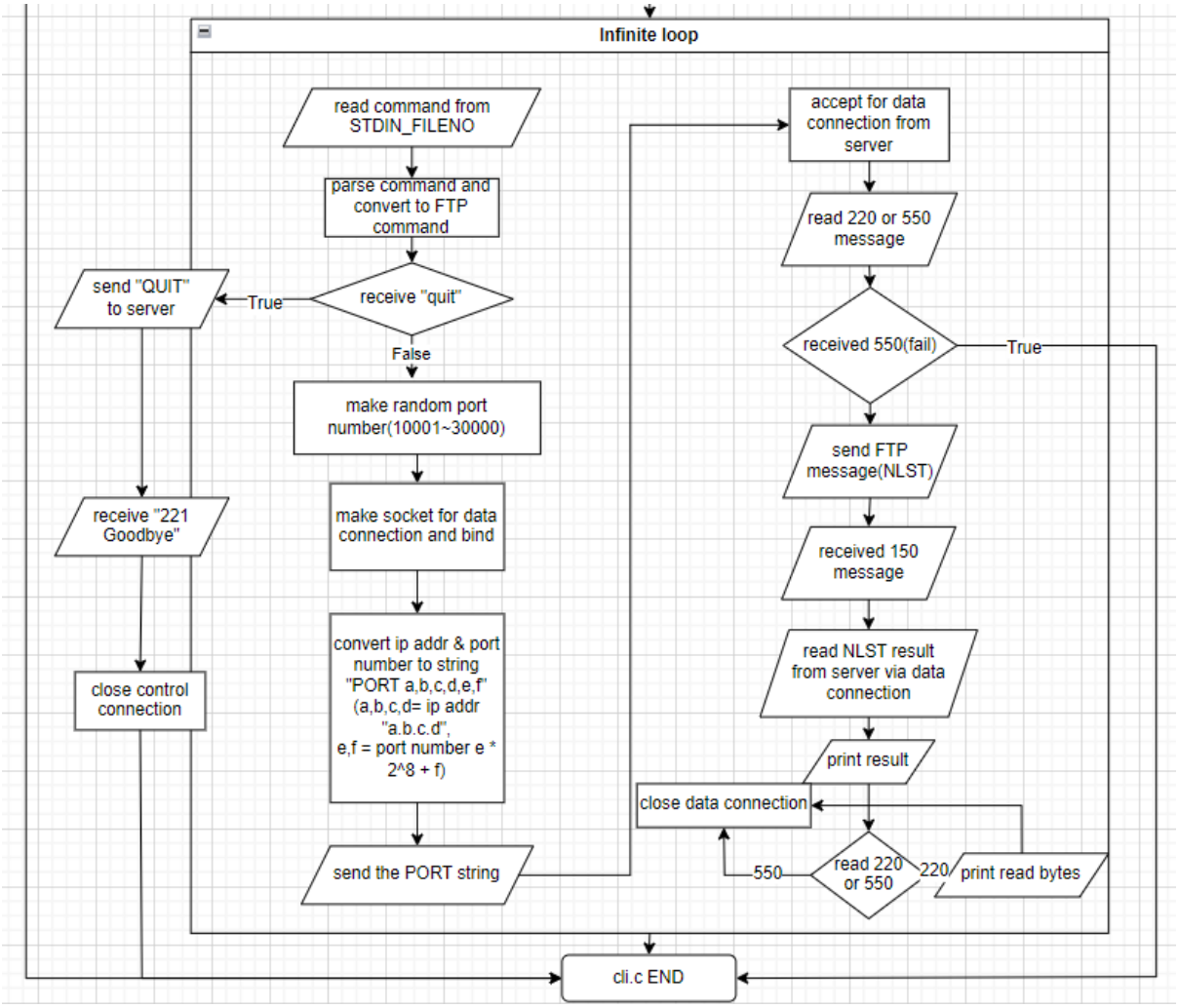
Name : [김태완]

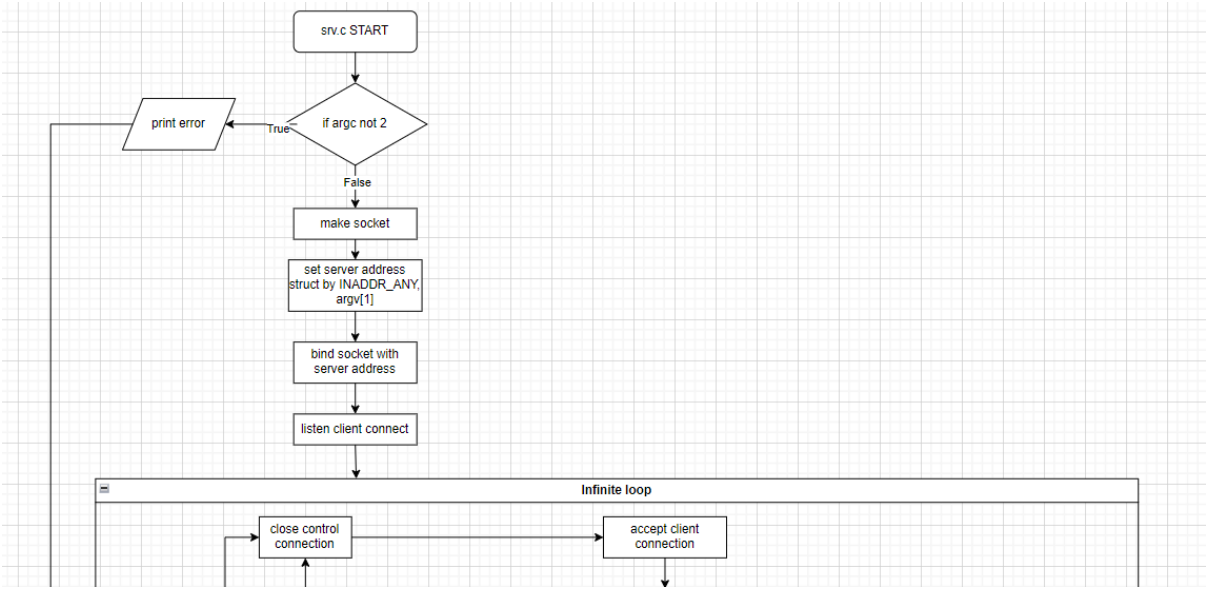
Introduction

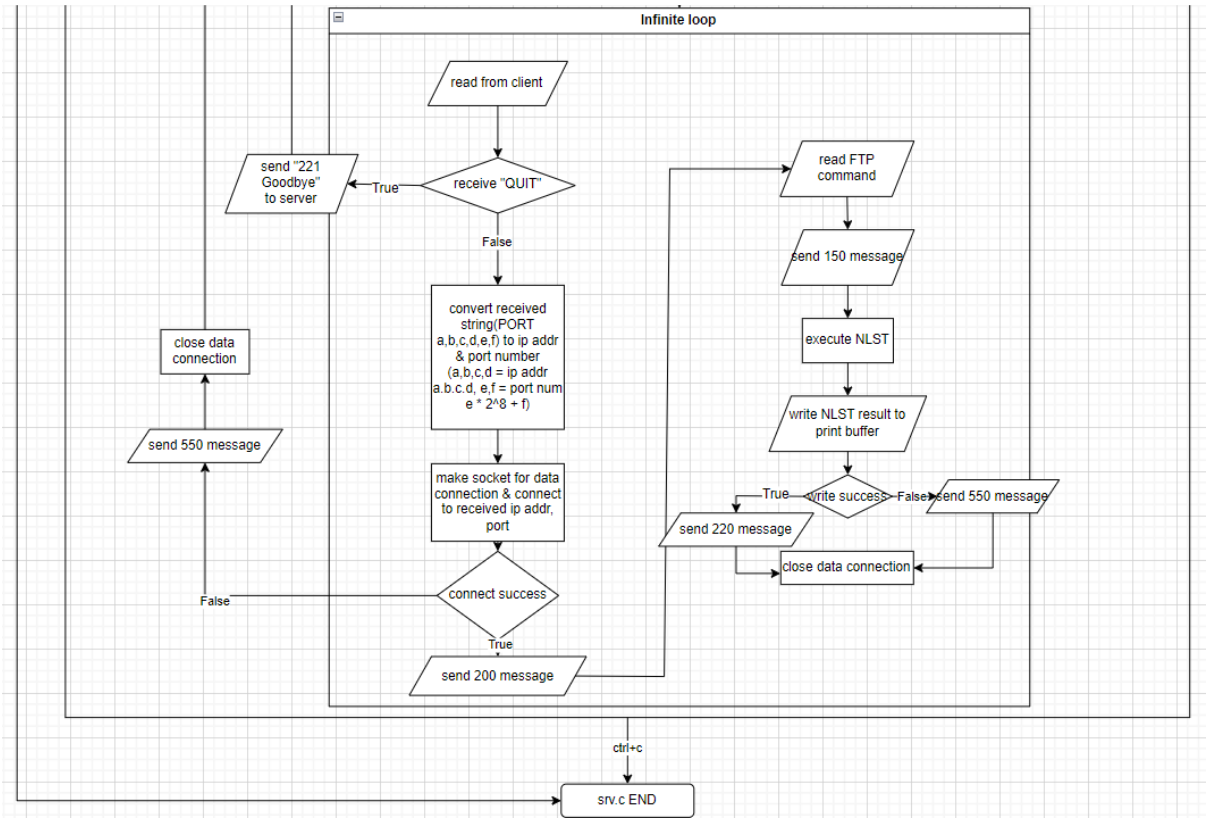
이번 과제는 data transfer 를 위한 data connection 을 추가로 만들어 명령어의 결과, 즉 데이터가 전송되는 길과 명령어와 signal 이 전송되는 길인 control connection 을 분리한 서버, 클라이언트 프로그램을 만드는 내용입니다. 기존에는 server, client 모두 상호 연결을 위한 소켓을 하나씩만 만들었는데, 이제는 server 에서 data connection 을 열 때, server 가 client 의 입장이 되어 접속하고, client 가 server 의 입장이 되어 연결을 수용해야 하므로 기존에 만들었던 서로의 구조를 참조하여 프로그램을 구성해야 합니다.

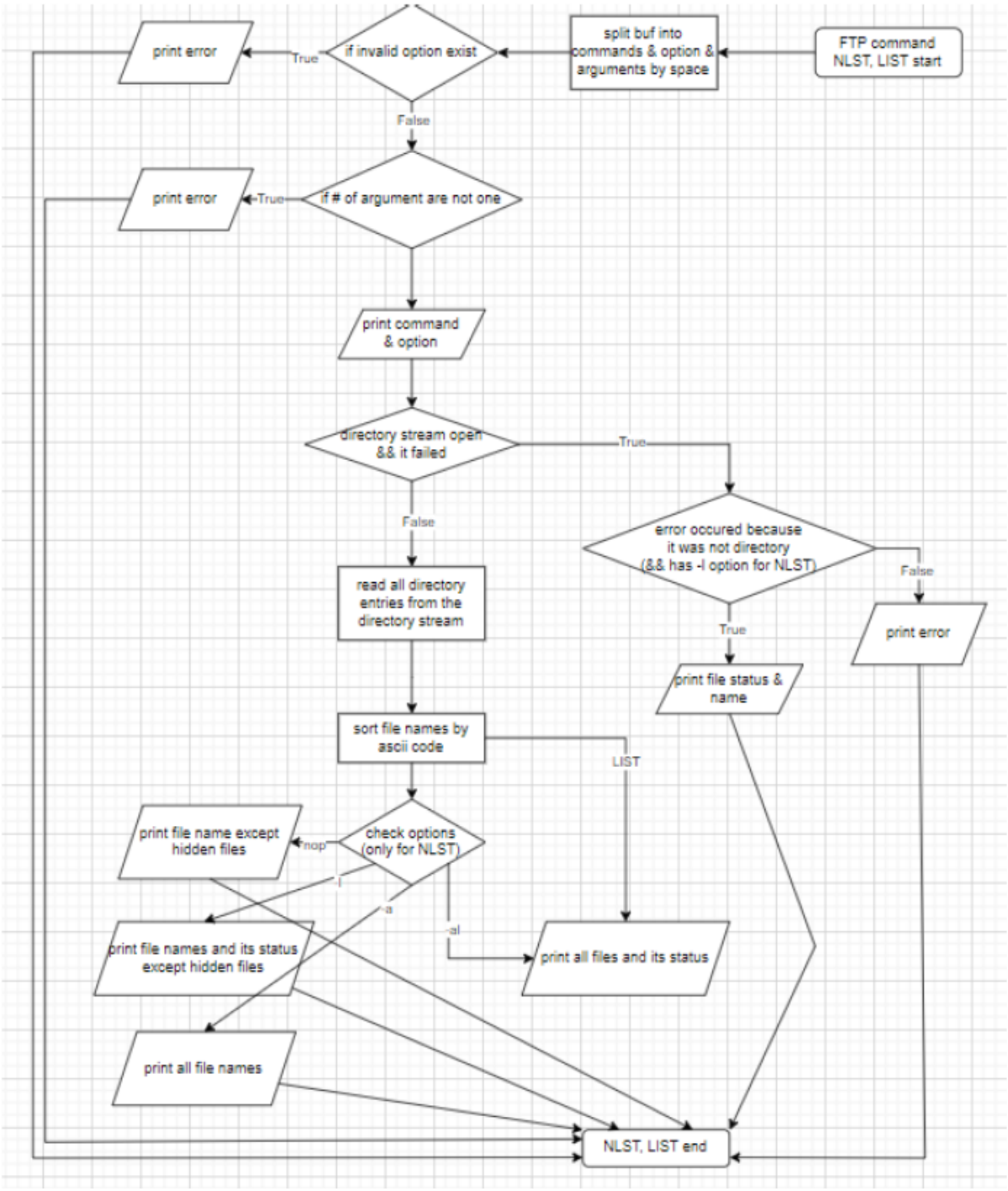
Flow chart




Infinite loop
read command from STDIN_FILENO
parse command and convert to FTP command
receive "quit"
True
send "QUIT" to server
False
make random port number(10001~30000)
make socket for data connection and bind
convert ip addr & port number to string "PORT a,b,c,d,e,f" (a,b,c,d= ip addr "a.b.c.d", e,f = port number e * 2^8 + f)
send the PORT string
receive "221 Goodbye"
close control connection
accept for data connection from server
read 220 or 550 message
received 550(fail)
True
send FTP message(NLST)
received 150 message
read NLST result from server via data connection
print result
read 220 or 550
220
print read bytes
550
close data connection
cli.c END


srv.c START
if argc not 2
True
print error
False
make socket
set server address struct by INADDR_ANY, argv[1]
bind socket with server address
listen client connect
Infinite loop
close control connection
accept client connection


Infinite loop
read from client
receive "QUIT"
True
send "221 Goodbye" to server
False
convert received string(PORT a,b,c,d,e,f) to ip addr & port number (a,b,c,d = ip addr a.b.c.d, e,f = port num e * 2^8 + f)
make socket for data connection & connect to received ip addr, port
connect success
False
True
send 200 message
read FTP command
send 150 message
execute NLST
write NLST result to print buffer
write success
True
False
send 550 message
send 220 message
close data connection
close data connection
send 550 message
ctrl+c
srv.c END



(LIST 부분은 제외)

Pseudo code

cli.c

DEFINE BUF_SIZE = 4096

```
FUNCTION convert_addr_to_str(buffer, address):
    buffer = "PORT " + inet_ntoa(address.sin_addr)
    FOR each '.' in buffer:
        REPLACE '.' WITH ','
    buffer += "," + (ntohs(address.sin_port) >> 8) + "," + (ntohs(address.sin_port) & 0xFF)

FUNCTION main(argc, argv):
    IF argc != 3:
        PRINT "Two arguments are needed: IP, port"
        EXIT(1)

    ctrlfd = socket(AF_INET, SOCK_STREAM, 0)
    temp.sin_family = AF_INET
    temp.sin_addr.s_addr = inet_addr(argv[1])
    temp.sin_port = htons(atoi(argv[2]))

    IF connect(ctrlfd, (struct sockaddr *)&temp, sizeof(temp)) < 0:
        PRINT "connect error"
        EXIT(1)

    WHILE TRUE:
        PRINT "> "
        n = read(STDIN_FILENO, buff, BUF_SIZE)
```

```
IF n < 0:
    PRINT "Read error"
    EXIT(1)
buff[n - 1] = '₩0'

split = tokenize(buff, " ₩b₩v₩f₩r₩t₩n")

IF split is empty OR (split[0] != "ls" AND split[0] != "quit"):
    CONTINUE

IF split[0] == "ls":
    cmd = "NLST"
ELSE IF split[0] == "quit":
    cmd = "QUIT"

FOR i = 1 TO length(split) - 1:
    cmd += " " + split[i]

IF cmd == "QUIT":
    write(ctrlfd, cmd, length(cmd))
    n = read(ctrlfd, buff, BUF_SIZE)
    IF n < 0:
        PRINT "read error"
        EXIT(1)
```

```
        buff[n] = 0
        PRINT buff
        close(ctrlfd)
        RETURN

port = 10001 + rand() % 20000
temp.sin_family = AF_INET
temp.sin_addr.s_addr = htonl(INADDR_LOOPBACK)
temp.sin_port = htons(port)

datafd = socket(AF_INET, SOCK_STREAM, 0)
bind(datafd, (struct sockaddr *)&temp, sizeof(temp))
listen(datafd, 5)

convert_addr_to_str(portcmd, &temp)
write(ctrlfd, portcmd, length(portcmd))

dataconfd = accept(datafd, (struct sockaddr *)&temp, &n)
IF dataconfd < 0:
        PRINT "data connection error"
        EXIT(1)
close(datafd)

n = read(ctrlfd, buff, BUF_SIZE)
```

```
IF n < 0:
    PRINT "read error"
    EXIT(1)
buff[n] = '₩0'
PRINT buff

IF buff starts with "550":
    close(dataconfd)
    close(ctrlfd)
    EXIT(0)

write(ctrlfd, cmd, length(cmd))

n = read(ctrlfd, buff, BUF_SIZE)
IF n < 0:
    PRINT "read error"
    EXIT(1)
buff[n] = '₩0'
PRINT buff

n = read(dataconfd, buff, BUF_SIZE)
IF n < 0:
    PRINT "read error"
    EXIT(1)
```

```
        buff[n] = '₩0'
        PRINT buff
        bytes = n

        n = read(ctrlfd, buff, BUF_SIZE)
        IF n < 0:
            PRINT "read error"
            EXIT(1)
        buff[n] = '₩0'
        PRINT buff

        IF buff starts with "226":
            PRINT "OK. " + bytes + " bytes is received."

        close(dataconfd)
```

srv.c

```
DEFINE BUF_SIZE = 4096
DEFINE MAX_BUF = 4096

FUNCTION convert_str_to_addr(str, addr):
    ip = 0
    port = 0
    tmp = strtok(str, " ")
    ptr = strtok(tmp, ",")
```

```
    FOR i = 0 TO 3:
        ip += atoi(ptr) << (24 - 8 * i)
        ptr = strtok(NULL, ",")
    FOR i = 0 TO 1:
        port += atoi(ptr) << (8 - 8 * i)
        ptr = strtok(NULL, ",")
    addr.sin_addr.s_addr = htonl(ip)
    addr.sin_port = htons(port)
    addr.sin_family = AF_INET

FUNCTION MtoS(infor, pathname, print_buf):
    time_buf = format_time(infor.st_mtime)
    str = determine_file_type(infor.st_mode)
    str += determine_permissions(infor.st_mode)
    IF infor.st_mode is directory:
        print_buf = format("%s %2ld %s %s %6ld %s %s/₩n", str, infor.st_nlink, getpwuid(infor.st_uid).pw_name, getgrgid(infor.st_gid).gr_name, infor.st_size, time_buf, pathname)
    ELSE:
        print_buf = format("%s %2ld %s %s %6ld %s %s₩n", str, infor.st_nlink, getpwuid(infor.st_uid).pw_name, getgrgid(infor.st_gid).gr_name, infor.st_size, time_buf, pathname)

FUNCTION NLST(buf, print_buf):
    split = tokenize(buf, " ")
```

```
options = parse_options(split)
IF options contains unknown option:
    print_buf += "Error: invalid option₩n"
    RETURN

IF number of arguments != 0 OR 1:
    print_buf += "Error: too many arguments₩n"
    RETURN

pathname = get_pathname(split)
dp = opendir(pathname)
IF dp is NULL:
    print_buf += format("Error: %s₩n", strerror(errno))
    RETURN

filenames = read_directory_entries(dp)
sort_filenames(filenames)
start_idx = find_start_index(filenames, options)

IF options contains 'l':
    FOR i = start_idx TO length(filenames):
        path = join(pathname, filenames[i])
        infor = stat(path)
        IF infor is error:
```

```
                print_buf += format("Error: %s₩n", strerror(errno))
                RETURN
            line = MtoS(infor, filenames[i], line_buf)
            print_buf += line
    ELSE:
        FOR i = start_idx TO length(filenames):
            path = join(pathname, filenames[i])
            infor = stat(path)
            IF infor is error:
                print_buf += format("Error: %s₩n", strerror(errno))
                RETURN
            print_buf += filenames[i]
            IF infor.st_mode is directory:
                print_buf += "/"
            print_buf += "₩n"

    closedir(dp)

FUNCTION main(argc, argv):
    IF argc != 2:
        PRINT "One argument is needed: port"
        EXIT(1)

    server_fd = socket(PF_INET, SOCK_STREAM, 0)
```

```
server_addr.sin_family = AF_INET
server_addr.sin_addr.s_addr = htonl(INADDR_ANY)
server_addr.sin_port = htons(atoi(argv[1]))

IF bind(server_fd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0:
    PRINT "bind error"
    EXIT(1)

listen(server_fd, 5)

WHILE TRUE:
    client_fd = accept(server_fd, (struct sockaddr *)&client_addr, &len)
    IF client_fd < 0:
        PRINT "control accept error"
        EXIT(1)

    WHILE TRUE:
        n = read(client_fd, buff, BUF_SIZE)
        IF n <= 0:
            PRINT "read error"
            EXIT(1)
        buff[n] = '₩0'
        PRINT buff
```

```
IF buff == "QUIT":
        send_buff = "221 Goodbye."
        write(client_fd, send_buff, length(send_buff))
        PRINT send_buff
        close(client_fd)
        BREAK

convert_str_to_addr(buff, &client_addr)

data_fd = socket(AF_INET, SOCK_STREAM, 0)
IF connect(data_fd, (struct sockaddr *)&client_addr, sizeof(client_addr)) < 0:
        send_buff = "550 Failed to access."
        write(client_fd, send_buff, length(send_buff))
        close(data_fd)
        close(client_fd)
        BREAK

send_buff = "200 PORT command successful"
write(client_fd, send_buff, length(send_buff))
PRINT send_buff

n = read(client_fd, buff, BUF_SIZE)
IF n < 0:
        PRINT "read error"
```

```
        EXIT(1)
buff[n] = '₩0'
PRINT buff

send_buff = "150 Opening data connection for directory list"
write(client_fd, send_buff, length(send_buff))
PRINT send_buff

NLST(buff, send_buff)
IF write(data_fd, send_buff, length(send_buff)) < 0:
        send_buff = "550 Failed transmission."
ELSE:
        send_buff = "226 Result is sent successfully"
write(client_fd, send_buff, length(send_buff))
PRINT send_buff
close(data_fd)
```

결과화면

1. server 를 2000 port 로 열고, client 에서 server 에 port 2000 으로 접속하니 '>'가 출력되어 연결이 되었음을 볼 수 있고(control connection), ls 를 입력하니 PORT command 가 전송되어 data connection 이 성공적으로 되었음을 200 message 로 알 수 있습니다. 이후 server 로 NLST(ls 의 FTP command)가 전송되어 출력되었고, NLST 가 수행되기 전 150 message 를 전송하고(server), 전송 받았음(client)을 볼 수 있습니다. 이후 client 에 ls 의 결과가 전송되었는데, 이후 ls 의 결과가 성공적으로 전송되었음을 보여주는 220 message 가 양쪽에 출력된 것으로 잘 전송되었음이 확인되면서, 마지막으로 client 에 전송된 data 의 bytes 수가 출력되었습니다.

2. 같은 명령어를 다시 입력해도, client 에는 똑같이 성공한 결과가 그대로 출력되지만, server 로 보내진 PORT 명령어가 약간 다릅니다. 2.를 실행할 때 control connection 은 유지되지만 data connection 은 새로 생성되어 port number 가 다시 랜덤으로 정해질 텐데, 1. 에서의 port number 59 * 2^8 + 54 = 15158 과 비교하면 43*2^8 + 128 = 11136 으로 서로 다르며, 10001~30000 사이의 값임을 볼 수 있습니다.

3. quit 을 입력하니 서버에 QUIT 이 전송되었고 221 Goodbye signal 이 server 에서 client 로 전송되어 client 는 종료되었음을 볼 수 있습니다.

4. 3.에서 quit 으로 control connection 을 끊어도 server 는 살아있으므로, 다시 client 가 server 에 접속하면 다시 control connection 이 연결되어 ls 를 입력하였을 때 1,2 번과 같이 결과가 잘 출력됨을 볼 수 있습니다.

5. 다시 연결한 control connection 을 다시 끊어보는 test 입니다. 잘 실행됨을 볼 수 있습니다.

고찰

기존에 프로그램의 시작부분에서만 진행했던 socket connection(socket(), bind(), listen(), connect(), accept())을 프로그램의 중간 부분에서 진행하니 accept() 함수가 무리한 요청을 받거나 새로운 연결이 들어오지 않는 한 block 된다는 점을 간과해 프로그램이 무한로딩에 걸리는 문제가 있었습니다. 이를 고려해 read(), write()의 순서를 바꾸니 정상적으로 프로그램이 동작해 그 후로 문제없이 코드를 작성하였습니다.