시스템 프로그래밍 실습

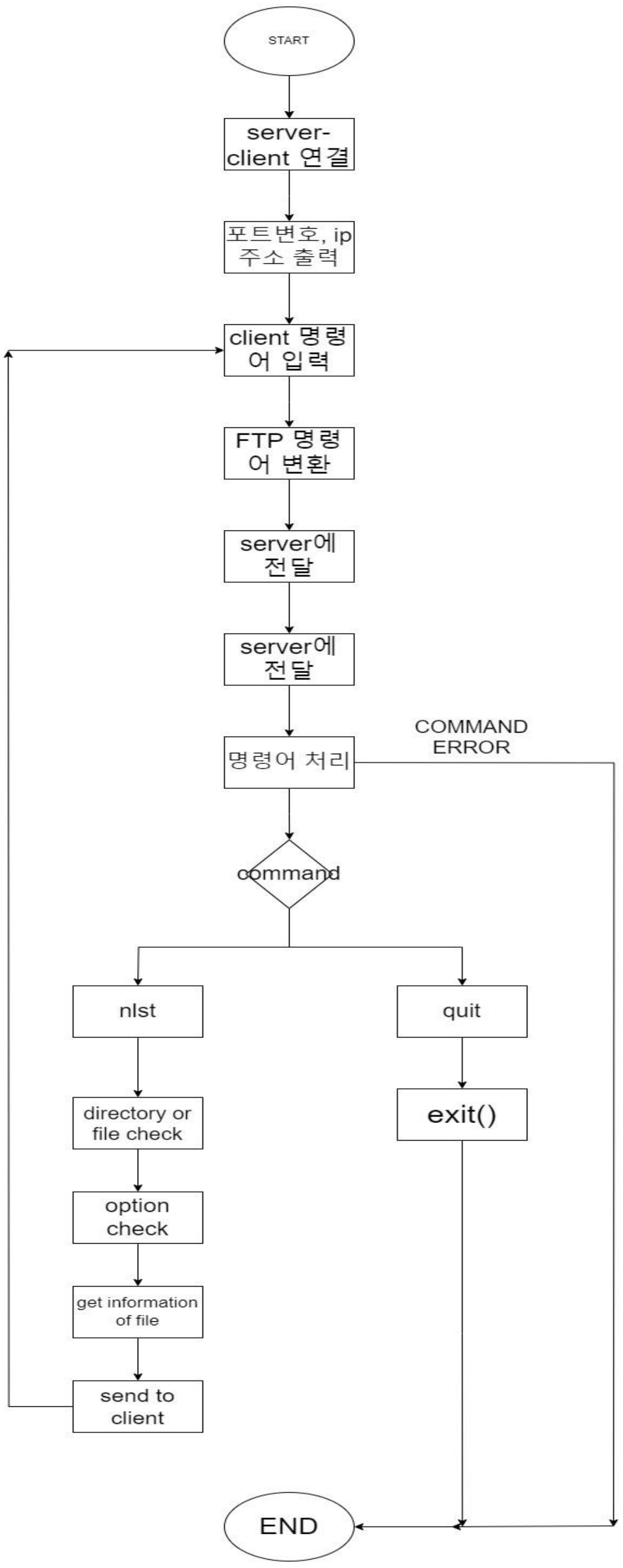
# [Assignment2-1]

| | | |
|---|---|---|
| Class | : | [A] |
| Professor | : | [김태석 교수님] |
| Student ID | : | [2019202032] |
| Name | : | [이상현] |

## Introduction

Client 와 server 소켓을 생성하고, 두 소켓을 연결한다. 이후 client 에서 명령어와 ip address, port number 를 입력 받고 명령어를 ftp 명령어로 변환한다. 이후 변환된 명령어를 server 소켓에 전송하고, server 에서는 해당 명령어에 대한 결과를 서버에 담아 client 에 전달한다. 명령어는 quit 와 옵션을 필요한 ls 두개에 대해서 처리하도록 하며 올바르지 않은 명령어에 대해 에러를 처리한다.

## Flow chart


START
server-client 연결
포트번호, ip 주소 출력
client 명령어 입력
FTP 명령어 변환
server에 전달
server에 전달
명령어 처리
COMMAND ERROR
command
nlst
directory or file check
option check
get information of file
send to client
quit
exit()
END

Client 와 server 소켓을 연결하고 server 에서 client 소켓의 ip address 와 port number 를 출력한다. 이후 client 에서 명령어를 입력받고 ftp 명령어로 변환한 뒤, server 로 전송한다. server 에서는 해당 명령어에 대한 동작을 처리하고 result_buffer 에 담아 client 로 전송한다.

# Pseudo code

<SERVER>

<cmd_process 함수>

```
int cmd_process(char*buff, char*result_buff){

/*              seperate command, options, directory from buffer                    */

    char *t_ptr = strtok(buff, " ");

    int td_num = 0;

    while(t_ptr != NULL){

        /*                          get command                              */

        if(td_num == 0){

            strcpy(command, t_ptr);

            td_num++;

        }

        //////////////   finish getting command   ////////////////



        else if(td_num == 1){

            /*                          get options                              */

            if(t_ptr[0] == '-'){

                for(int s = 1; s<strlen(t_ptr); s++){

                    if(t_ptr[s] == 'a')

                        aflag++;

                    else if(t_ptr[s] == 'l')

                        lflag++;
                    else{
                         error_handling(0);
                         return -1;
                    }
               }
               td_num++;
          }
          //////////////            finish getting options        ///////
          /*                        get directory                            */
          else{
               strcpy(directory, t_ptr);
               dir_cnt++;
               td_num++;
          }
          //////////////            finish getting directory       ///////
     }
}
/////////////////////  finish seperating from command       //////////////////////////

if(!strcmp(command, "QUIT")){          //case of QUIT
    strcpy(result_buff, "QUIT");
}
if(!strcmp(command, "NLST")){          //case of NLST
     /*                           start nlst -al                           */
```

```
if(aflag && lflag || !aflag && !lflag){
    write(1, "NLST -al₩n", 10);
    for(int i = 0; i<filecnt; i++){
        option_l(filenames[i], result_buff);
    }
}
//////////                  finish nlst -al                  /////////

/*                          start nlst -a                             */
if(aflag){
    write(1, "NLST -a₩n", 9);
      for(int i= 0; i<filecnt; i++){
        char filetype = GetFiletype(file, filenames[i]);
        if(filetype == 'd'){         //file type d
            strcat(result_buff, filenames[i]);
            strcat(result_buff, "/₩n");
        }
        else{
            strcat(result_buff, filenames[i]);
            strcat(result_buff, "₩n");
        }
    }
}
//////////                  finish nlst -a                  /////////
```

```
    /*                          start nlst -l                           */
    if(lflag){
        write(1, "NLST -l₩n", 9);
        for(int i= 0; i<filecnt; i++){
            option_l(filenames[i], result_buff);    //get file information
        }
        chdir(current_directory);
        return 1;
    }
    //////////                  finish nlst -l                  /////////
```

<CLIENT_INFO 함수>

```
int client_info(struct sockaddr_in client_addr){
        char*client_IP = inet_ntoa(client_addr.sin_addr);
        write(STDOUT_FILENO, client_IP, strlen(client_IP));
        sprintf(client_port, "%d", client_addr.sin_port);
}
```

<MAIN 함수>

```
int main(int argc, char **argv) {
/*                          open socket and listen                          */
    struct sockaddr_in server_addr, client_addr;
    int socket_fd, client_fd;
int len, len_out;
if((socket_fd = socket(PF_INET, SOCK_STREAM, 0)) < 0){
	printf("Server: Can't open stream socket.");
	return 0;
}
int opt = 1;
setsockopt(socket_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));

/*                              set information                         */
memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
int portno = atoi(argv[1]);
server_addr.sin_port = htons(portno);

/*                              binding socket                          */
if(bind(socket_fd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0){
	printf("Server: Can't bind local address₩n");
	return 0;
}
listen(socket_fd, 5);
/////////////////// finish opening socket and listening     ////////////////////////

/*          start communicating socket                    */
```

```
for(;;){
    int flag = 0;
    len = sizeof(client_addr);
    client_fd = accept(socket_fd, (struct sockaddr*)&client_addr, &len);

    /*              display client ip and port              */
    if(client_info(client_addr) < 0)
        write(STDERR_FILENO, "client_info() err!!₩n", 21);

    /*              communicate between sockets              */
    while(1){
        memset(buff, 0, sizeof(buff));
        n = read(client_fd, buff, MAX_BUFF);
        buff[n] = '₩0';

        /*              command execute and result              */
        if(cmd_process(buff, result_buff) < 0) {
            write(STDERR_FILENO, "cmd_process() err!!₩n", 21);
            close(client_fd);
            close(socket_fd);
            exit(0);
            break;
        }
        write(client_fd, result_buff, strlen(result_buff));
```

```
            /*                  case of QUIT                    */
            if(!strcmp(result_buff, "QUIT"))
            {
                flag = 1;
                write(STDOUT_FILENO, "QUIT₩n", 6);
                close(client_fd);
                close(socket_fd);
                exit(0);
                break;
            }
        }
    }
    /*   finish communicating socket          */

    close(client_fd);
    close(socket_fd);
    return 0;
}
```

<CLIENT>

<CONV_CMD 함수>

```
int conv_cmd(char*buff, char*cmd_buff){
    memset(cmd_buff, 0, sizeof(cmd_buff));
    char command[30];
```

```
int i = 0;
/*                    seperate command from buffer            */
for(i = 0; i<strlen(buff); i++){
    if(buff[i] != ' ')
    {
        command[i] = buff[i];
        command[i+1] = '₩0';
    }
    else{
        break;
    }
}
/////                 finish seperating command           ///////

if(!strcmp("quit", command)){    //case of quit
    strcpy(cmd_buff, "QUIT");
}
if(!strcmp("ls", command)){        // case of ls

    /*          change ls to NLST           */
    cmd_buff[0] = 'N';
    cmd_buff[1] = 'L';
    cmd_buff[2] = 'S';
    cmd_buff[3] = 'T';
```

```
        cmd_buff[4] = '₩0';

        /*              add option directory            */
        int j = 4;
        for(j = 2; j<strlen(buff); j++){
            cmd_buff[j+2] = buff[j];
            cmd_buff[j+2+1] = '₩0';
        }
        return 1;
}
```

<PROCESS RESULT> 함수

```
void process_result(char *rcv_buff){
    write(STDOUT_FILENO, rcv_buff, strlen(rcv_buff));
}
```

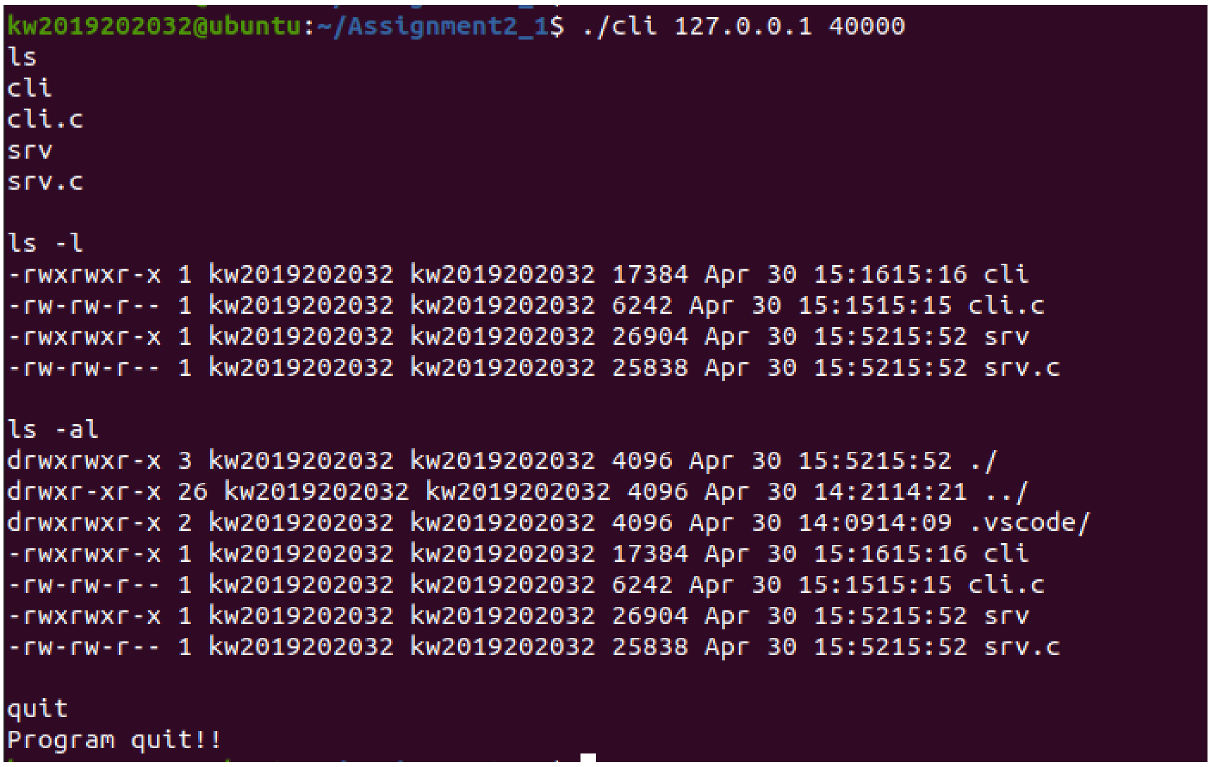
<MAIN 함수>

```
int main(int argc, char**argv){
    char buff[MAX_BUFF], cmd_buff[MAX_BUFF], rcv_buff[RCV_BUFF];
    int n;

    /*          open socket and connect to server                   */
    int sockfd, len;
    struct sockaddr_in server_addr;
```

```
        char *haddr = (char*)malloc(sizeof(char)*100);

        strcpy(haddr, argv[1]);

        if((sockfd = socket(PF_INET, SOCK_STREAM, 0)) < 0){

                printf("can't create socket₩n");

                return -1;

        }

        memset(buff, 0, sizeof(buff));

        memset(&server_addr, 0, sizeof(server_addr));

        server_addr.sin_family = AF_INET;

        server_addr.sin_addr.s_addr = inet_addr(haddr);        //set ip number

        int portno = atoi(argv[2]);                            //set port number

        server_addr.sin_port = htons(portno);

        if(connect(sockfd,      (struct      sockaddr*)&server_addr,      sizeof(server_addr))      <
    0){    //connect with server socket

                printf("Can't connect₩n");

                return -1;

        }

        //////////   Finish opening socket and connecting to server              ///////////


        /*            read and write buffer with server                                    */

        for(;;){

                if(conv_cmd(buff, cmd_buff) < 0){

                /* convert ls (including) options to NLST (including options) */

                        write(STDERR_FILENO, "conv_cmd() error!!₩n", 20);
                        exit(1);
                }


                n = strlen(cmd_buff);
                /*                  write to server socket                      */
                if(write(sockfd, cmd_buff, n) != n){
                        write(STDERR_FILENO, "write() error₩n", 15);
                        exit(1);
                }
                /*                  read from server socket                      */
                if((n = read(sockfd, rcv_buff, RCV_BUFF)) < 0){
                        write(STDERR_FILENO, "read() error₩n", 14);
                        exit(1);
                }
                rcv_buff[n] = '₩0';
                /*                        quit program                           */
                if(!strcmp(rcv_buff, "QUIT")){
                        write(STDOUT_FILENO, "Program quit!!₩n", 16);
                        exit(1);
                }
                process_result(rcv_buff);
                memset(rcv_buff, 0, sizeof(rcv_buff));


                while(n >= 4096){
                        n = read(sockfd, rcv_buff, RCV_BUFF);

                        process_result(rcv_buff);

                        memset(rcv_buff, 0, sizeof(rcv_buff));

                }

            }

        }
```

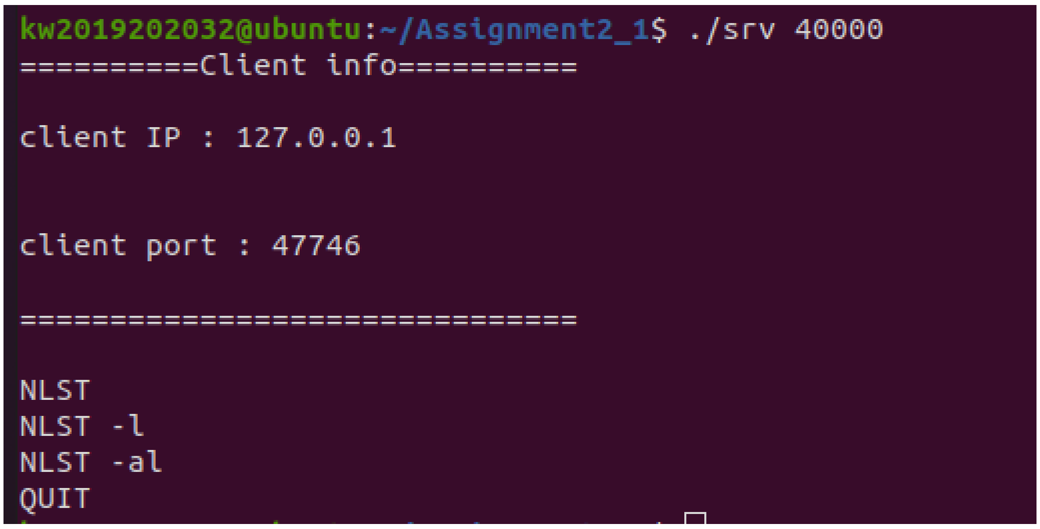
## 결과화면

<Client socket>



```
kw2019202032@ubuntu:~/Assignment2_1$ ./cli 127.0.0.1 40000
ls
cli
cli.c
srv
srv.c

ls -l
-rwxrwxr-x 1 kw2019202032 kw2019202032 17384 Apr 30 15:1615:16 cli
-rw-rw-r-- 1 kw2019202032 kw2019202032 6242 Apr 30 15:1515:15 cli.c
-rwxrwxr-x 1 kw2019202032 kw2019202032 26904 Apr 30 15:5215:52 srv
-rw-rw-r-- 1 kw2019202032 kw2019202032 25838 Apr 30 15:5215:52 srv.c

ls -al
drwxrwxr-x 3 kw2019202032 kw2019202032 4096 Apr 30 15:5215:52 ./
drwxr-xr-x 26 kw2019202032 kw2019202032 4096 Apr 30 14:2114:21 ../
drwxrwxr-x 2 kw2019202032 kw2019202032 4096 Apr 30 14:0914:09 .vscode/
-rwxrwxr-x 1 kw2019202032 kw2019202032 17384 Apr 30 15:1615:16 cli
-rw-rw-r-- 1 kw2019202032 kw2019202032 6242 Apr 30 15:1515:15 cli.c
-rwxrwxr-x 1 kw2019202032 kw2019202032 26904 Apr 30 15:5215:52 srv
-rw-rw-r-- 1 kw2019202032 kw2019202032 25838 Apr 30 15:5215:52 srv.c

quit
Program quit!!
```

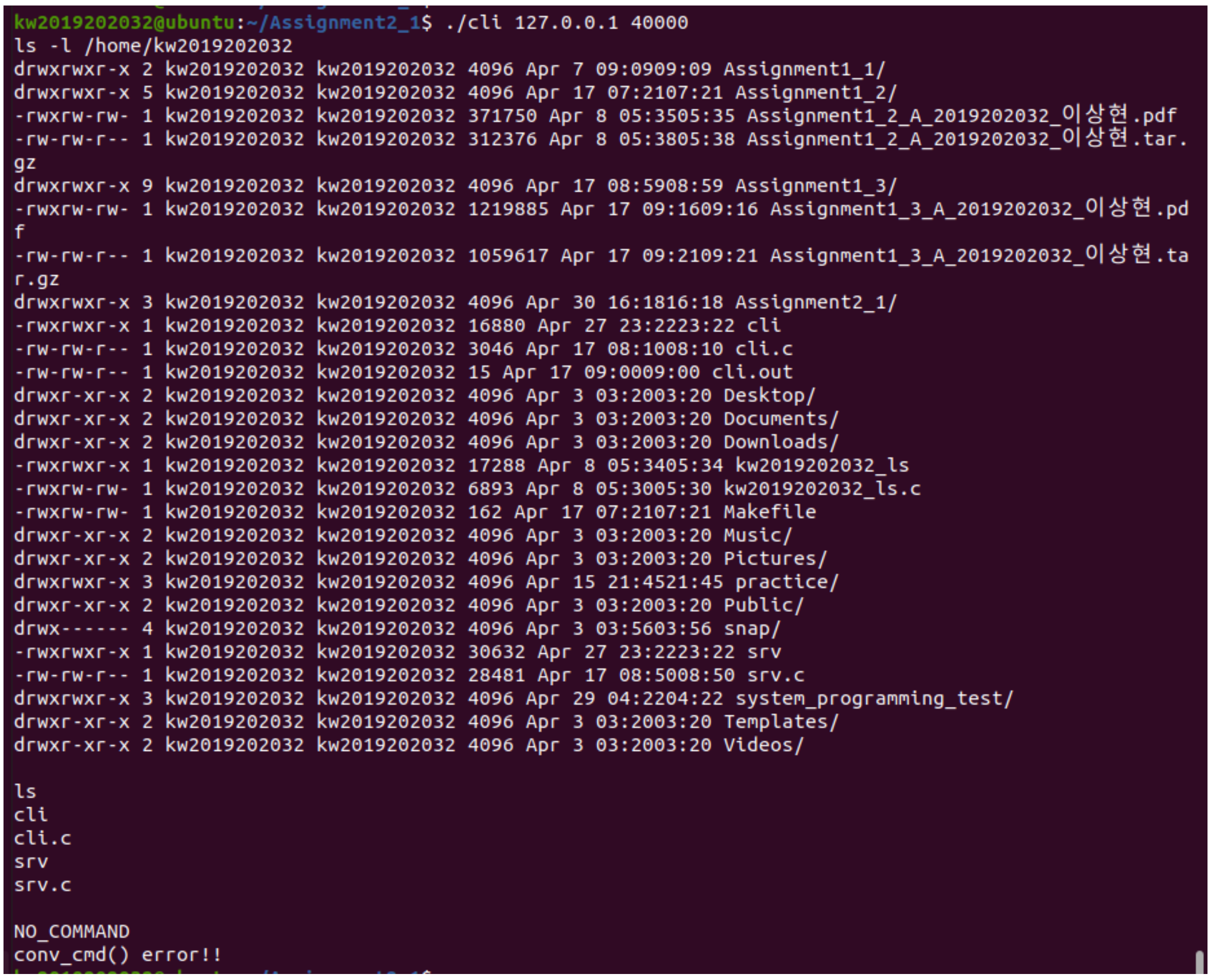
<Server socket>



```
kw2019202032@ubuntu:~/Assignment2_1$ ./srv 40000
==========Client info==========

client IP : 127.0.0.1


client port : 47746

===============================

NLST
NLST -l
NLST -al
QUIT
```

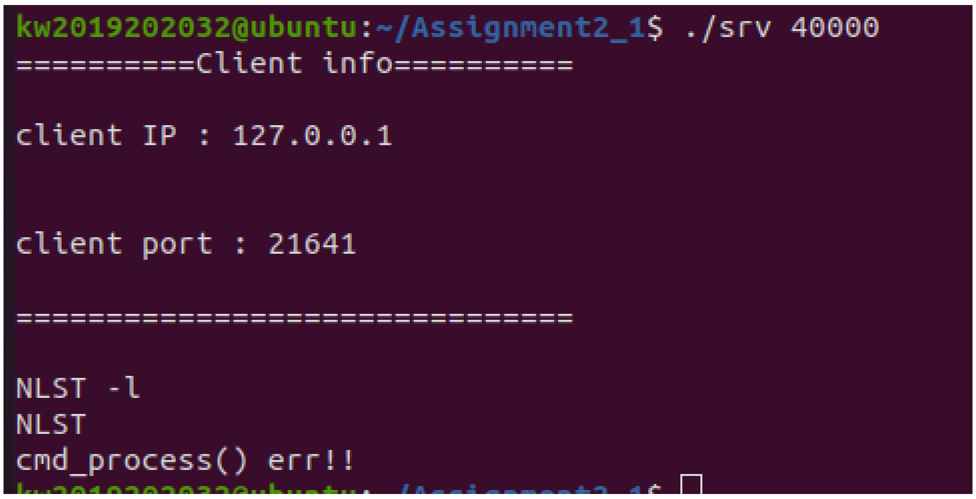
다음과 같이 ls 와 quit 명령어에 대해 정상적으로 동작하며, client 의 port 와 ip 값을 정상적으로 출력하는 것을 확인할 수 있다.

<Client socket>



```
kw2019202032@ubuntu:~/Assignment2_1$ ./cli 127.0.0.1 40000
ls -l /home/kw2019202032
drwxrwxr-x 2 kw2019202032 kw2019202032 4096 Apr 7 09:0909:09 Assignment1_1/
drwxrwxr-x 5 kw2019202032 kw2019202032 4096 Apr 17 07:2107:21 Assignment1_2/
-rwxrw-rw- 1 kw2019202032 kw2019202032 371750 Apr 8 05:3505:35 Assignment1_2_A_2019202032_이상현.pdf
-rw-rw-r-- 1 kw2019202032 kw2019202032 312376 Apr 8 05:3805:38 Assignment1_2_A_2019202032_이상현.tar.
gz
drwxrwxr-x 9 kw2019202032 kw2019202032 4096 Apr 17 08:5908:59 Assignment1_3/
-rwxrw-rw- 1 kw2019202032 kw2019202032 1219885 Apr 17 09:1609:16 Assignment1_3_A_2019202032_이상현.pd
f
-rw-rw-r-- 1 kw2019202032 kw2019202032 1059617 Apr 17 09:2109:21 Assignment1_3_A_2019202032_이상현.ta
r.gz
drwxrwxr-x 3 kw2019202032 kw2019202032 4096 Apr 30 16:1816:18 Assignment2_1/
-rwxrwxr-x 1 kw2019202032 kw2019202032 16880 Apr 27 23:2223:22 cli
-rw-rw-r-- 1 kw2019202032 kw2019202032 3046 Apr 17 08:1008:10 cli.c
-rw-rw-r-- 1 kw2019202032 kw2019202032 15 Apr 17 09:0009:00 cli.out
drwxr-xr-x 2 kw2019202032 kw2019202032 4096 Apr 3 03:2003:20 Desktop/
drwxr-xr-x 2 kw2019202032 kw2019202032 4096 Apr 3 03:2003:20 Documents/
drwxr-xr-x 2 kw2019202032 kw2019202032 4096 Apr 3 03:2003:20 Downloads/
-rwxrwxr-x 1 kw2019202032 kw2019202032 17288 Apr 8 05:3405:34 kw2019202032_ls
-rwxrw-rw- 1 kw2019202032 kw2019202032 6893 Apr 8 05:3005:30 kw2019202032_ls.c
-rwxrw-rw- 1 kw2019202032 kw2019202032 162 Apr 17 07:2107:21 Makefile
drwxr-xr-x 2 kw2019202032 kw2019202032 4096 Apr 3 03:2003:20 Music/
drwxr-xr-x 2 kw2019202032 kw2019202032 4096 Apr 3 03:2003:20 Pictures/
drwxrwxr-x 3 kw2019202032 kw2019202032 4096 Apr 15 21:4521:45 practice/
drwxr-xr-x 2 kw2019202032 kw2019202032 4096 Apr 3 03:2003:20 Public/
drwx------ 4 kw2019202032 kw2019202032 4096 Apr 3 03:5603:56 snap/
-rwxrwxr-x 1 kw2019202032 kw2019202032 30632 Apr 27 23:2223:22 srv
-rw-rw-r-- 1 kw2019202032 kw2019202032 28481 Apr 17 08:5008:50 srv.c
drwxrwxr-x 3 kw2019202032 kw2019202032 4096 Apr 29 04:2204:22 system_programming_test/
drwxr-xr-x 2 kw2019202032 kw2019202032 4096 Apr 3 03:2003:20 Templates/
drwxr-xr-x 2 kw2019202032 kw2019202032 4096 Apr 3 03:2003:20 Videos/

ls
cli
cli.c
srv
srv.c

NO_COMMAND
conv_cmd() error!!
```

<server socket>



```
kw2019202032@ubuntu:~/Assignment2_1$ ./srv 40000
==========Client info==========

client IP : 127.0.0.1


client port : 21641

===============================

NLST -l
NLST
cmd_process() err!!
```

다음과 같이 ls-l 에+ 경로가 주어졌을 때 정상적으로 동작하고 존재하지 않는 command 에 대해 에러를 처리하는 것을 확인할 수 있다.

## 고찰

해당과제를 수행하는 과정에서 겪었던 가장 어려웠던 점은 socket 과 client 를 연결하고 소켓들 사이의 정보를 주고받는 것이었다. 특히 처음에는 한쪽 서버에서 전달한 정보가 다른 서버에 언제 전달이 되고, 정보를 전달받은 서버에서는 언제 정보를 전달할 수 있는지 등을 알기가 어려워서 과제를 수행하는데 어려움을 겪었다. 하지만 read 와 write 함수에 대해 이해하고, buffer 를 이용한 정보의 전달에 대해 이해함으로써 해당 문제를 해결할 수 있었다. 이를 통해 두 소켓을 연결하고 정보를 주고받는 동작과정에 대해 잘 이해할 수 있었고, 나아가 fork 문 등을 활용한다면 여러 client 와 정보를 주고받는 server 를 생성하여 웹서버를 구현할 수 있을 것이라는 생각을 가지게 되었다.

## Reference

강의자료만  참고하였습니다.