시스템 프로그래밍 실습

[Assignment3-1]

Class : **[A]**

Professor : **[김태석 교수님]**

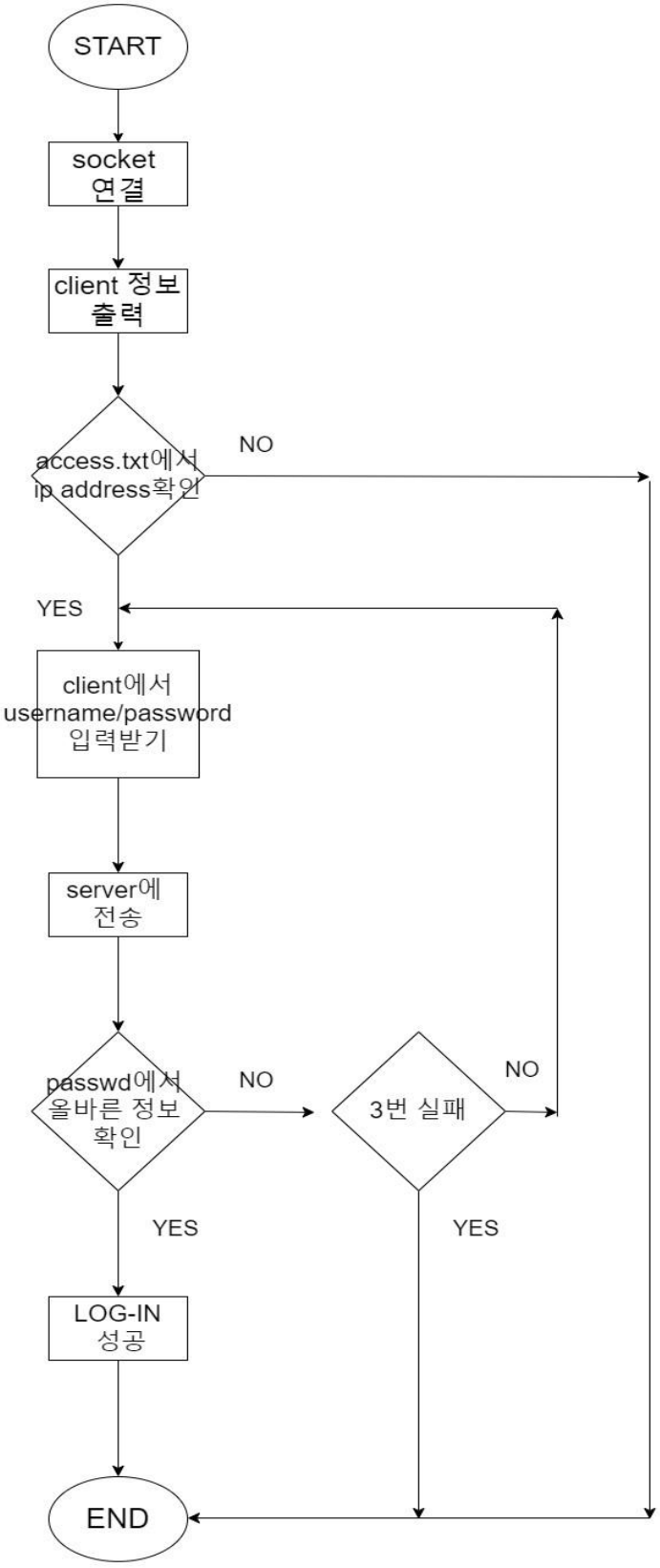
Student ID : **[2019202032]**

Name : **[이상현]**

Introduction

해당과제는 client 소켓에서 username 과 password 를 통해 로그인을 시도하고, server 소켓에서 해당 username 과 password 가 존재하는지 확인한 후 로그인 성공/실패를 알려주는 과제이다. Username 과 password 는 passwd 구조체 파일에서 확인하며, access.txt.파일을 통해 client 의 port 번호로 접속이 가능한지 확인한다. 만약 client 의 port 로 접속에 성공한다면 client 로부터 3 번의 username 과 password 를 입력받고 3 번이상 틀릴경우 connection 을 종료한다.

Flow chart


START
socket
연결
client 정보
출력
access.txt에서
ip address확인
NO
YES
client에서
username/password
입력받기
server에
전송
passwd에서
올바른 정보
확인
NO
3번 실패
NO
YES
YES
LOG-IN
성공
END

Pseudo code

<srv.c>

```
int client_info(struct sockaddr_in client_addr, char*ip){
	write(1, ip, strlen(ip));
	write(STDOUT_FILENO, client_port, strlen(client_port));
}
int user_match(char*user, char*passwd)
{
	FILE *fp;
	struct passwd *pw;
	fp = fopen("passwd", "r");   //open passwd
	if(fp == NULL) {
		return 0;
	}
	/*********check if password and user name is efficient*********/
	while((pw = fgetpwent(fp))!= NULL) {
		if(!strcmp(user, pw->pw_name)) {	//if username if correct
			if(!strcmp(passwd, pw->pw_passwd)) {	//if password is correct
				return 1;
			}
		}
	}
	/************************************************************/
}
```

```
int log_auth(int connfd)
{
	while(1) {;
		read from client and get username, password;
		if(n = (user_match(user, passwd)) == 1) {	//log-in succeeded
			return 1;
		}
		else if(n == 0) {
			if(count >= 3) {	/* 3 time failed*/
				return 0;
			}
			/****************re-try*****************/
			write(connfd, "FAIL", 5);
			count++;
			continue;
			/*************************************/
		}
	}
	return 1;
}
void Ip_Slicing(char *str, char**sliced) {
	int num = 0;
	char *ptr = strtok(str, ".");
	/*************** start slicing *************/
```

```
        while(ptr != NULL) {
                strcpy(sliced[num++], ptr);//strcat sliced string
                ptr = strtok(NULL, ".");
        }
}
int main(int argc, char*argv[]) {
        int listenfd, connfd;
        struct sockaddr_in servaddr, cliaddr;
        FILE *fp_checkIP;     //FILE stream to check client's IP


       listenfd = socket(PF_INET, SOCK_STREAM, 0);
        int opt = 1;
        setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));

        memset(&servaddr, 0, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = htonl(INADDR_ANY);//set address
        servaddr.sin_port = htons(atoi(argv[1]));//set port

        if(bind(listenfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0){   //bind socket
                printf("Server: Can't bind local address₩n");
                return 0;
        }
```

```
    listen(listenfd, 5);     //listen from client

/*****************************************************************************************
**/
    for(;;) {
        int clilen = sizeof(cliaddr);
        connfd = accept(listenfd, (struct sockaddr *) &cliaddr, &clilen);
        get ip address from client;
        open access.txt;

        /**************** check if client ip is acceptable ****************/
        while((fgets(IPs, MAX_BUF, fp_checkIP) != NULL))
        {
            /********** Slicing IP address by dot(.) ***********/
             Ip_Slicing(IPs, access_IP);
             Ip_Slicing(client_ip, input_IP);
             /************************************************/


             /***************** Check string *****************/
             for(int i = 0; i<4; i++){
                  if(!strcmp(input_IP[i], "*"))     //wildcard
                       continue;
                  if(!strcmp(access_IP[i], "*"))    //wildcard
```

```
                continue;
            if(strcmp(access_IP[i], input_IP[i]))     //not equal string
                find = 0;
        }
        /*********************************************/

        /************** client connected ***************/
        if(find) {
            write(connfd, "ACCEPTED", 9);
        }
        /*********************************************/
}

/***************       failed to connect     ***************/
if(!find) {
          write(connfd, "REJECTION", 10);}
/*****************************************************/

/*********************************************************************/

if(log_auth(connfd) == 0) {        //log-in failed
        write(connfd, "DISCONNECTION", 14); //send disconnection
}
/******log-in success*******/
```

```
            write(connfd, "OK", 3);
            /*************************/
        }
}
```

<cli.c>

```
#define MAX_BUF 20
#define CONT_PORT 20001

char ip_str[30];
void log_in(int sockfd) {
        write(sockfd, ip_str, strlen(ip_str));    //send ip address to server

        /************** read result from server **************/
        n = read(sockfd, buf, MAX_BUF);
        buf[strlen(buf)] = '₩0';
        /************************************************/

        /****************** check result ********************/
        if(!strcmp(buf, "ACCEPTED"))
            write(1, "** It is connected to Server **₩n", 33);

        if(!strcmp(buf, "REJECTION")){
            write(1, "** Connection refused **₩n", 26);
```

```
        close(sockfd);
        exit(0);
}
/*************************************************/

for(;;){
/******************** send ID and username from user *****************/
     if(read(STDIN_FILENO, username, MAX_BUF) < 0){ /* receive string from user*/
            close(sockfd);
            exit(0);
        }
        username[strlen(username)-1] = '₩0';

        //get password from user
        passwd = getpass("Input Password : ");


        //pass username to server
        write(sockfd, username, strlen(username));
        write(sockfd, passwd, strlen(passwd));
/*************************************************************/

        memset(buf, 0, MAX_BUF);
```

```
n = read(sockfd, buf, MAX_BUF);

buf[strlen(buf)] = '₩0';
if(!strcmp(buf, "OK")){
        memset(buf, 0, MAX_BUF);

        read(sockfd, buf, MAX_BUF); //get result from server
        buf[strlen(buf)] = '₩0';
        /***************case of success***************/
        if(!strcmp(buf, "OK")){
                write(1, "' logged in **₩n", 16);
                break;
        }
        /*****************************************/

        /*************Failed to log-in**************/
        if(!strcmp(buf, "FAIL")){     //fail->re-try
                write(1, "** Log-in failed **₩n", 21);
                continue;
        }
        if(!strcmp(buf, "DISCONNECTION")) { // buf is “DISCONNECTION”
                write(1, "** Connection closed **₩n", 25);
                break;
        }
```

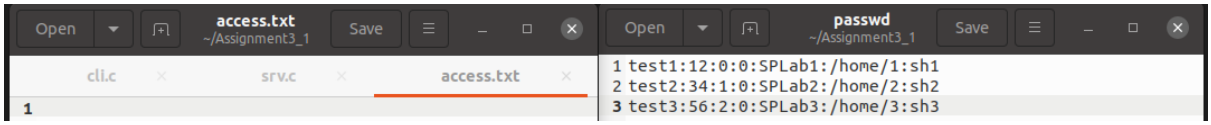
```
                /****************************************/
            }
        }
}
int main(int argc, char *argv[])
{
        int sockfd, n, p_pd;
        struct sockaddr_in servaddr;



/********************* prepare client socket **********************/
        sockfd = socket(AF_INET, SOCK_STREAM, 0);
        memset(&servaddr, 0, sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = inet_addr(argv[1]);

        //pointer to dotted-decimal string//
        strcpy(ip_str,inet_ntoa(servaddr.sin_addr));

        servaddr.sin_port = htons(atoi(argv[2]));
        connect(sockfd, (struct sockaddr*) &servaddr, sizeof(servaddr));        //connect with
server
/****************************************************************/
        log_in(sockfd); //log_in process
```

```
    close(sockfd);

    return 0;

}
```

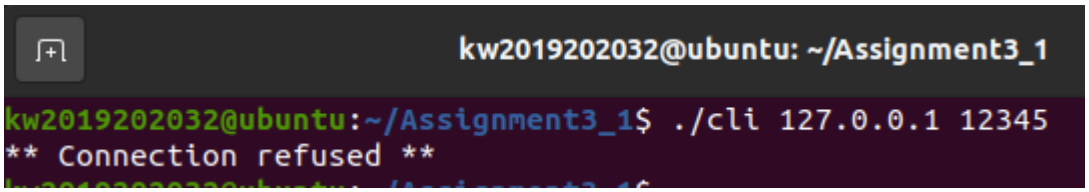
결과화면

<접속이 불가한 IP 를 가진 Client 가 접속할 경우>

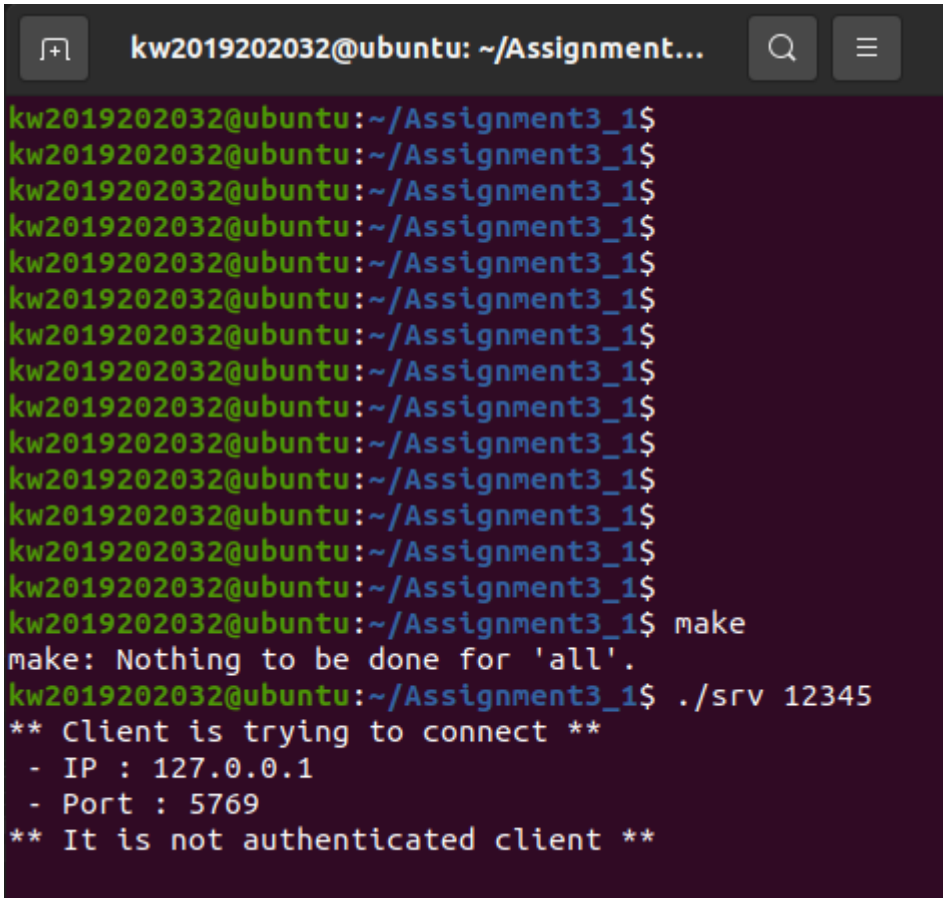
-access.txt & passwd



-client

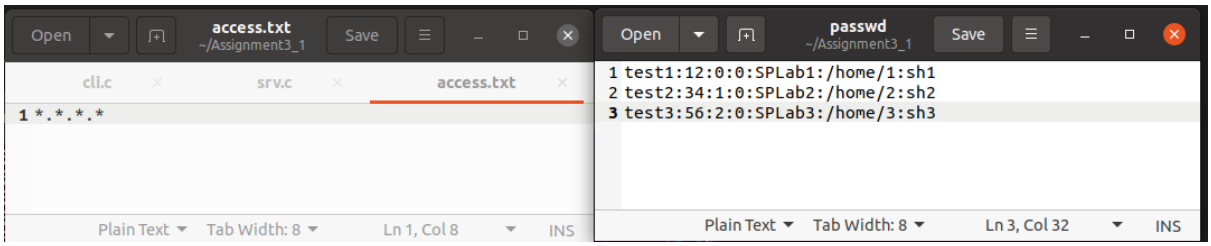


-server

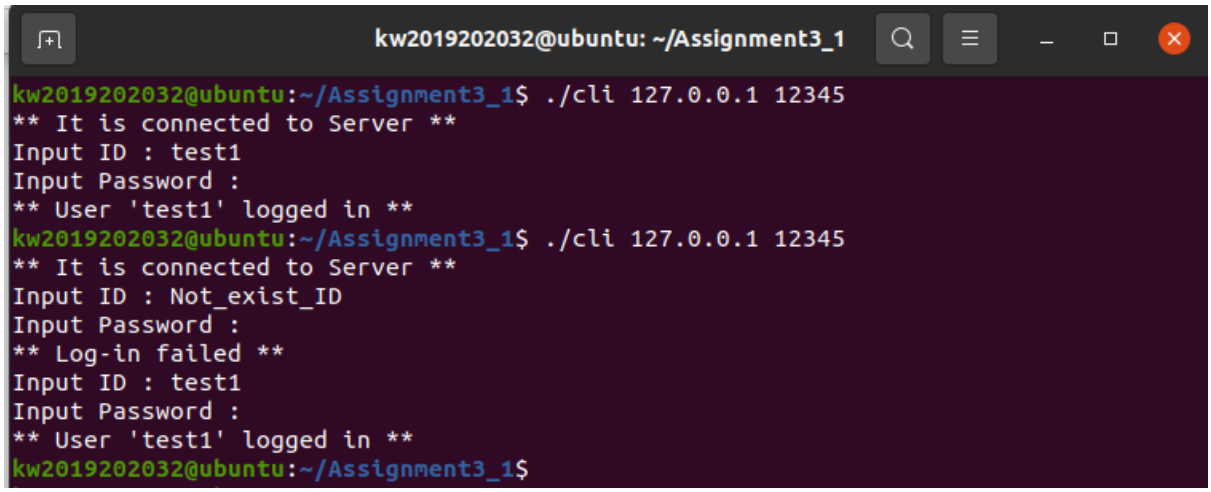


다음과 같이 access.txt 가 비어있는 상태에서는 127.0.0.1 로 접근이 불가능한 것을 확인할 수 있다.

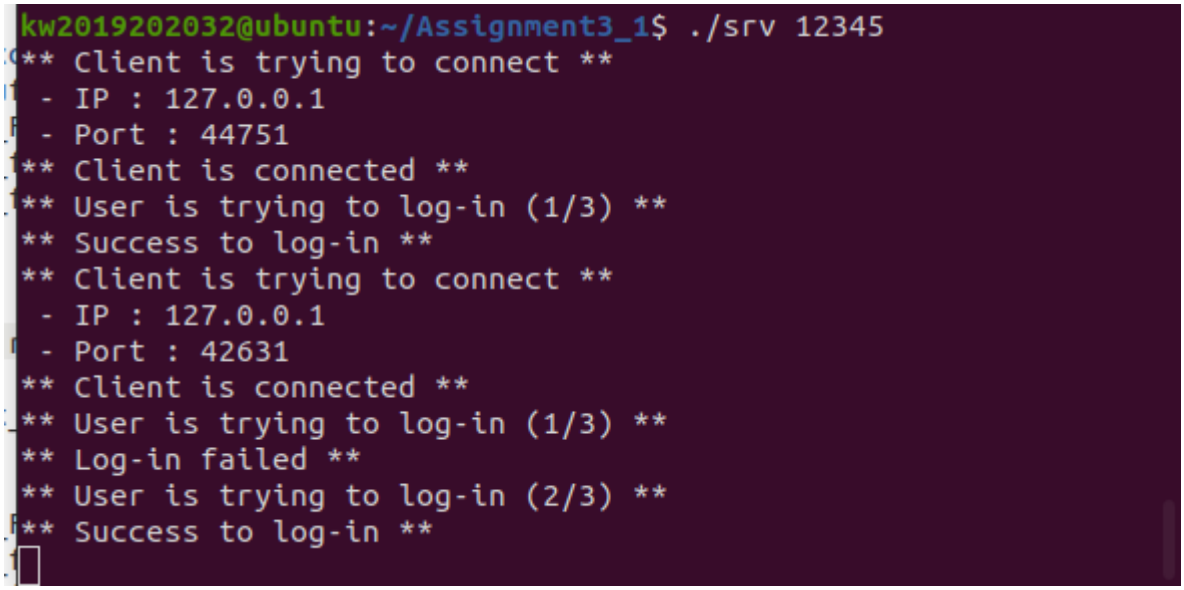
<성공적으로 로그인을 마친 경우>



-client



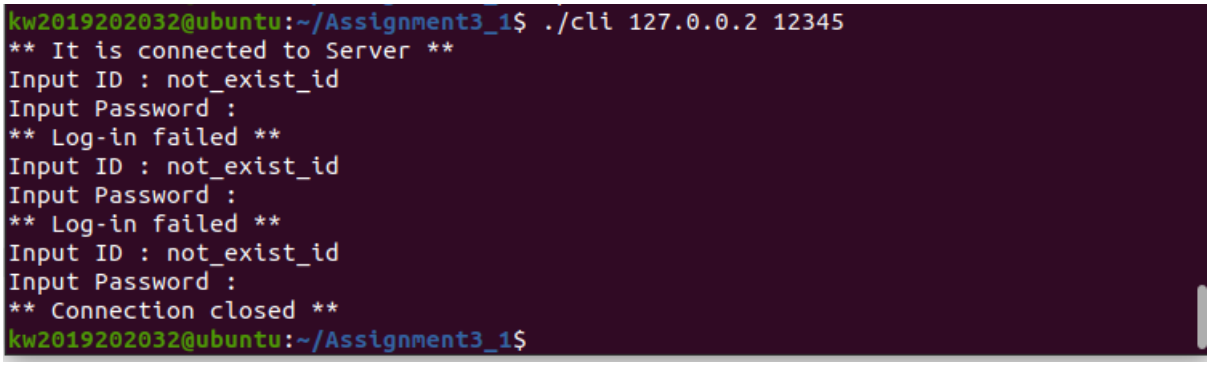
-server



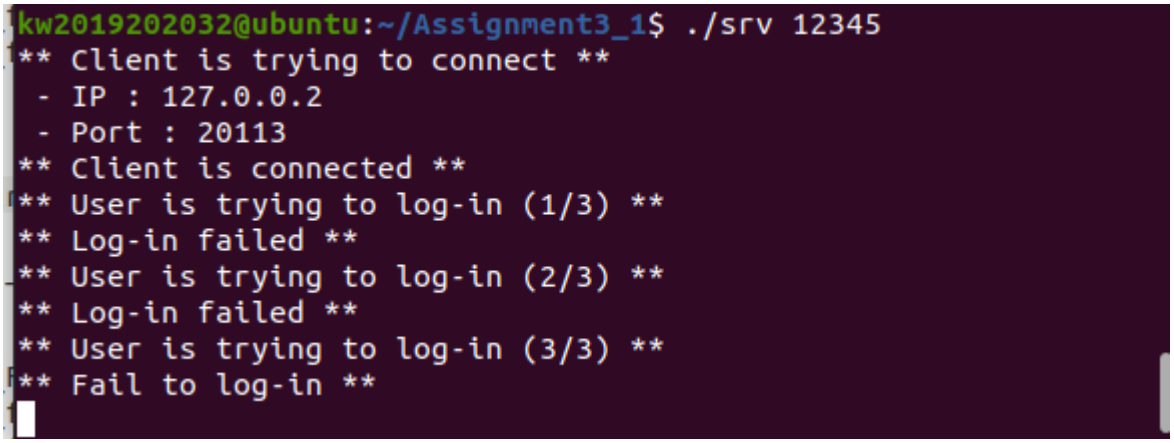
다음과 같이 client 가 연결될 때마다 ip 주소를 출력하고 있고, 로그인이 성공적으로 이루어질때까지(3 번이하로) 입력을 받다가 정상적으로 로그인이 되었을 경우를 처리하고 있는 것을 확인할 수 있다.

<로그인을 세번 시도했지만 모두 실패한 경우>

-client



-server



다음과 같이 로그인을 3 번 실패했을 경우 로그인이 실패하면서 client 를 종료시키는 것을 확인할 수 있다.

고찰

해당과제를 하는 과정에서 client 에서 username 과 password 를 올바르게 입력으로 주었을 때, server 에서는 정상적으로 log-in 이 성공했다고 되지만, client 에서는 그렇지 않고 input ID 를 다시 요구하는 경우가 가끔씩 발생하는 것을 확인하였다. 이를 해결하기 위해서 여러방면으로 알아보았지만, 명확한 해결책을 찾지 못하였다. 개인적으로 이러한 문제에 대해 server 가 log-in 에 성공했다고 출력하기 이전에 client 로 "OK"를 write 해줘야 하고, 반대로 client 에서는 "OK"를 read 해야 하는데, 여기서 문제가 발생했을 것이라고 추측하였다. 따라서 server 의 write 문에서

if(write() < 0){print()}

else {printf()}

를 통해 확인해본 결과 write 는 동작하는 것을 확인하였다. 반면 client 에서는 read 함수를 통해 server 에서 write 한 "OK"문장을 읽어오지 못하는 것을 확인하였다. 이 이유에 대해서 개인적으로 생각해본 것은 server 에서 write 를 2 번 연속하고, 반대로 client 에서는 read 2 번 연속하는데, 이 과정에서 write 와 read 의 실행 순서가 꼬여서 발생한 것 같다는 것이었다. 따라서 이러한 문제를 해결하기 위해 server 에서 write 와 write 사이에 read 를, client 에서는 read 와 read 사이에 write 를 하여 의미없는 문장을 주고받음으로써 write 와 read 의 순서를 이전에 비해 비교적 명시적으로 표현해주었고, 이에 따라 앞서 발생한 그러한 문제가 더 이상 발생하지 않는다는 것을 확인하였다. 이를 통해 socket programming 에서는 write 와 read 의 순서가 내가 원하는대로, 혹은 내가 생각한대로 구성되지 않을 수도 있다는 것을 유념할 필요가 있다고 생각되었다.

Reference

강의자료만 참고하였습니다.