시스템 프로그래밍 실습

[Assignment3-1]

Class : **[D]**

Professor : **[최상호교수님]**

Student ID : **[2020202060]**

Name : **[홍왕기]**

Introduction

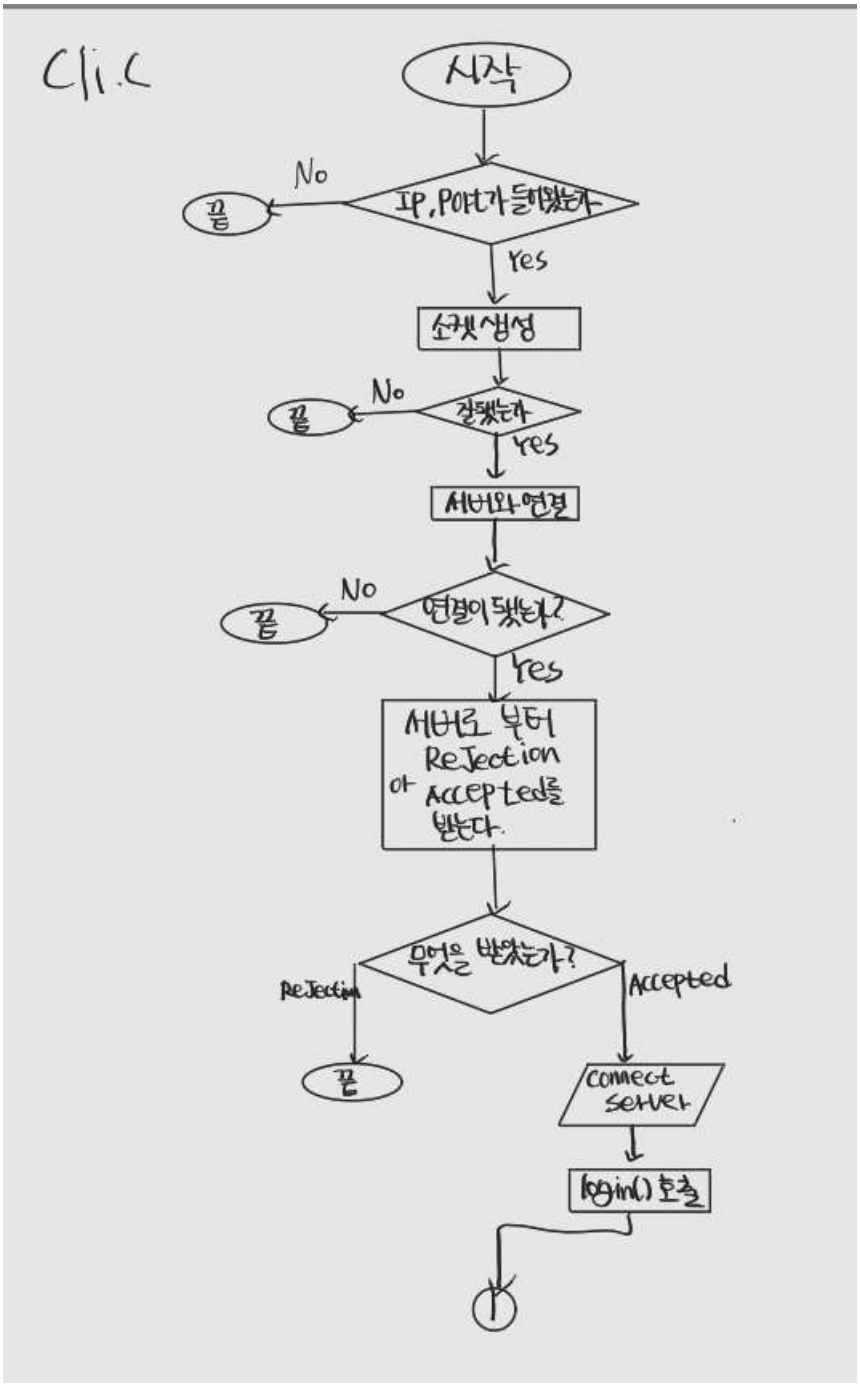
이 프로젝트는 클라이언트와 서버 간의 효율적인 통신을 기반으로 한 로그인 시스템을 구현하는 프로젝트이다. 클라이언트는 사용자가 입력한 아이디와 비밀번호를 서버에 전송하고, 서버는 이 정보를 받아서 저장된 passwd 파일에서 일치하는 아이디와 비밀번호를 찾아내어 인증한다. 클라이언트가 보낸 정보가 서버에서 확인되지 않으면, 서버는 클라이언트에게 인증 실패 메시지를 전달한다. 특히, 보안을 강화하기 위해 클라이언트가 3 번 인증에 실패할 경우, 클라이언트의 접속을 차단하여 보안을 유지하는 형태로 구현해야 한다.

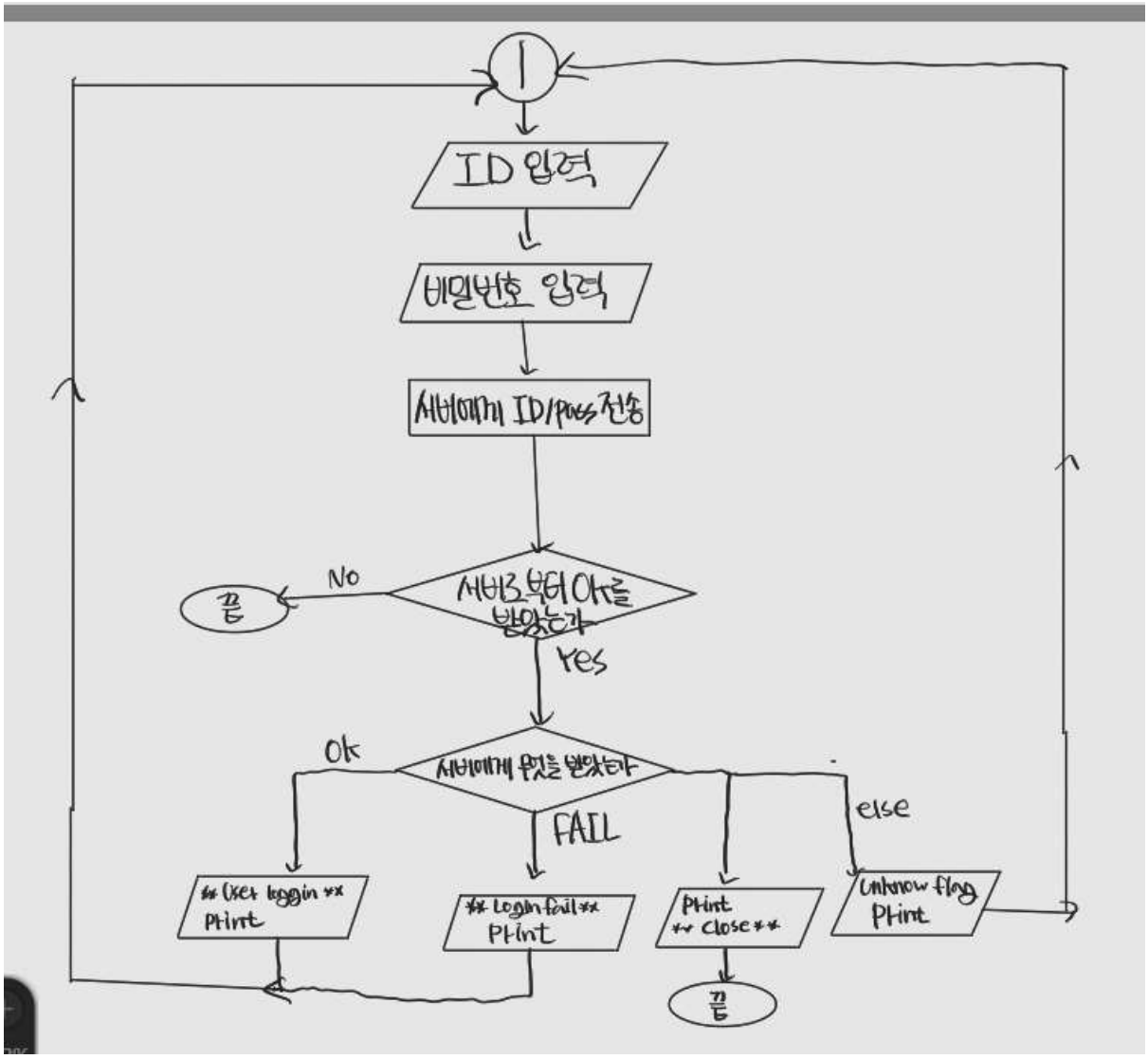
반면에, 클라이언트가 제대로 된 아이디와 비밀번호를 제공할 경우에, 서버는 로그인에 성공했다는 메시지를 클라이언트에게 전달하여 인증을 완료한다.

이러한 시스템을 통해 클라이언트와 서버 간의 안전하고 신뢰할 수 있는 통신이 가능해지며, 이를 통해서 사용자 인증과 접근제어에 대한 이해를 높인다.

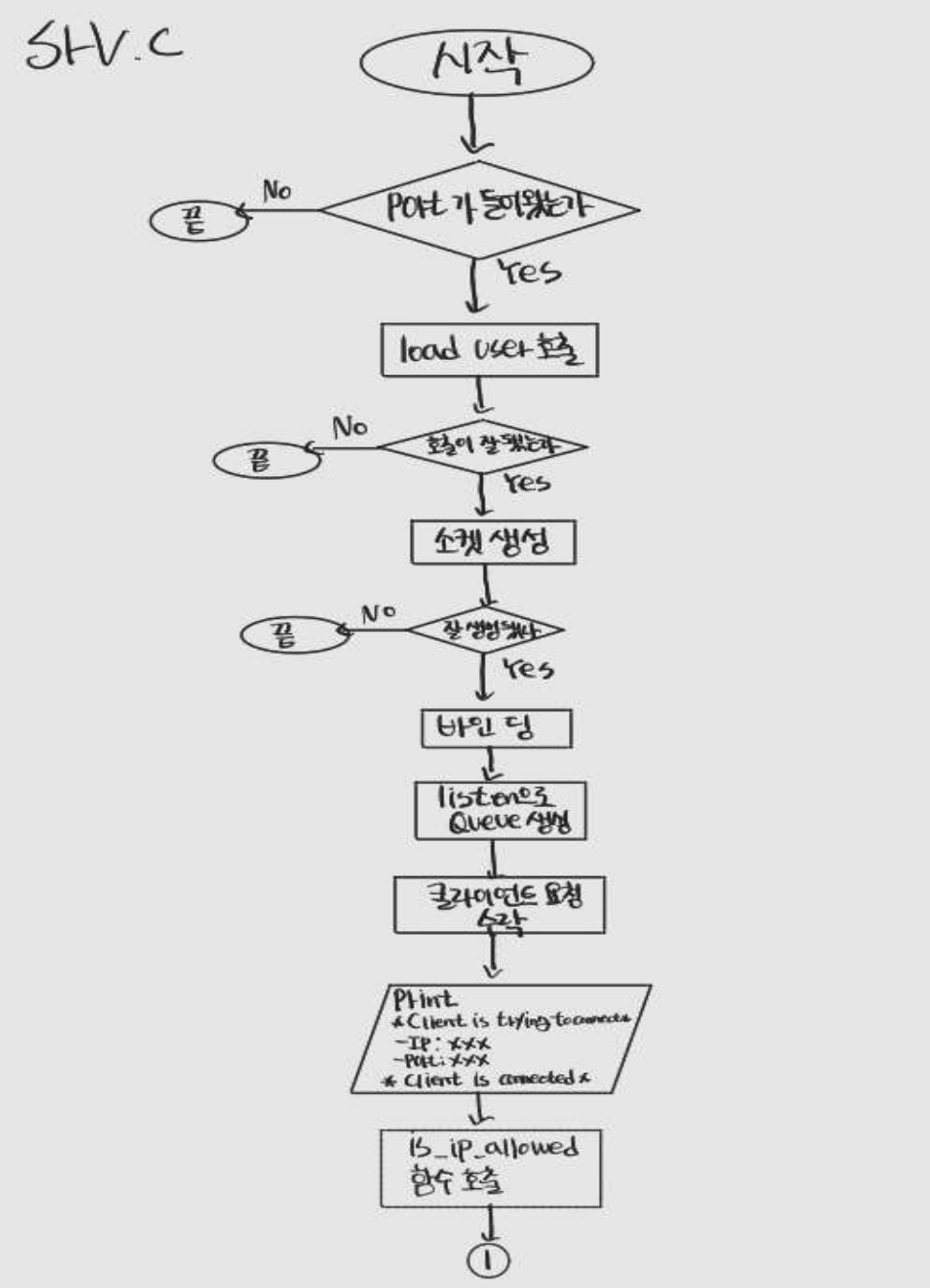
Flow chart

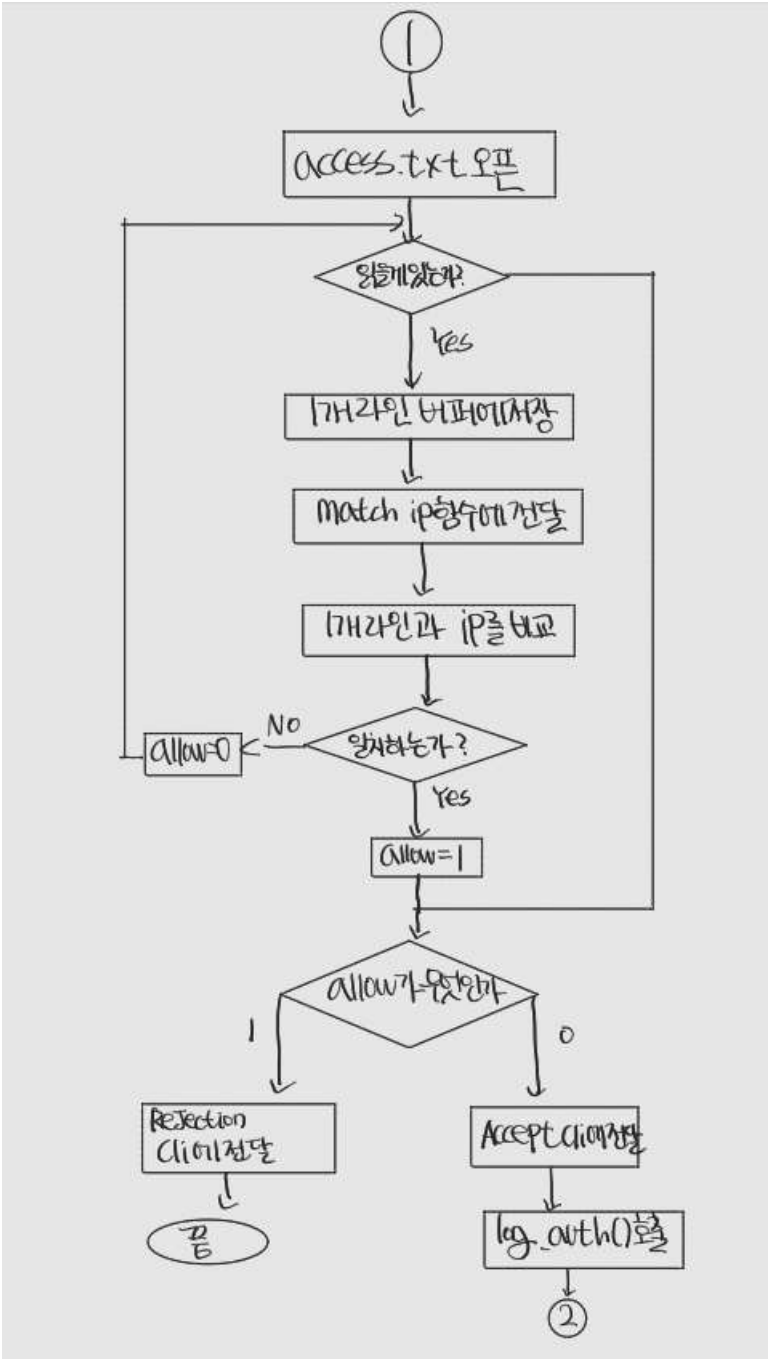
cli.c

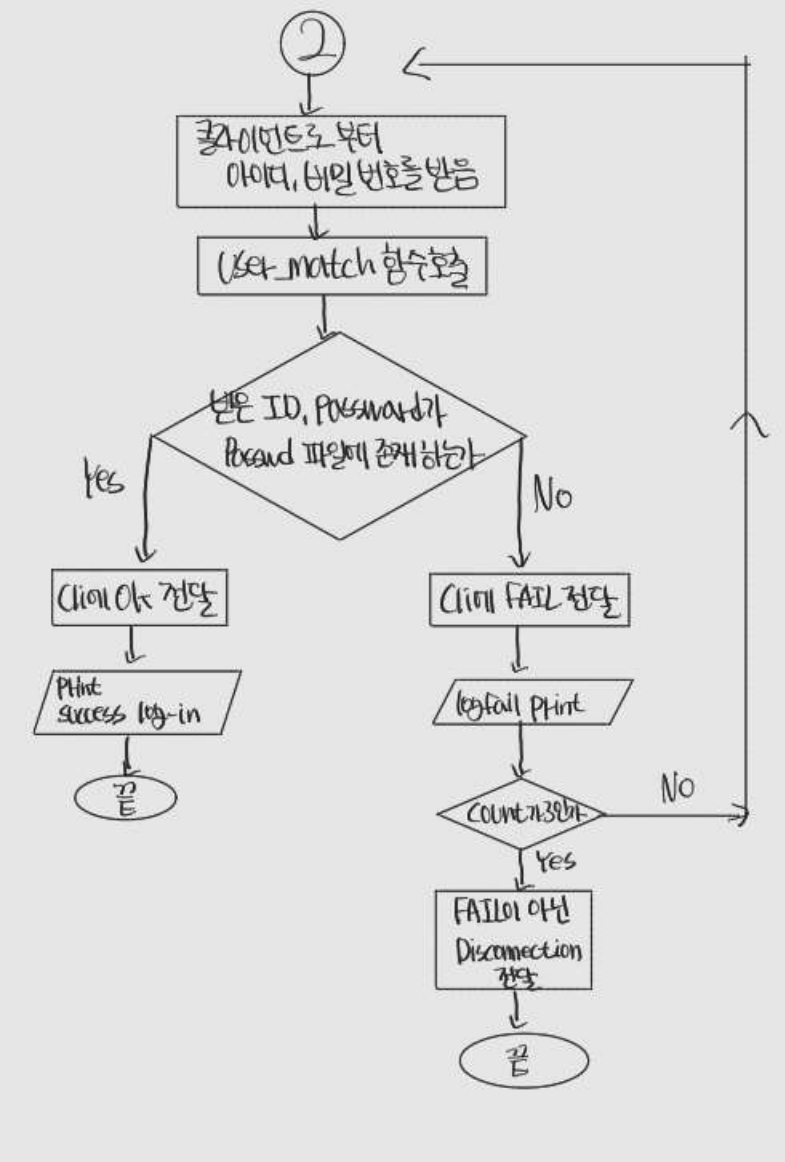



ID 입력
비밀번호 입력
서버에게 ID/Pass 전송
No
끝
서버로부터 ok를 받았는가
Yes
ok
서버에게 뭔가를 받았다
FAIL
else
** User loggin **
Print
** Login fail **
Print
Print
** close **
끝
Unknow flag
Print

Srv.c


Srv.c
시작
No
끝
PORT가 들어왔는가
Yes
load user 호출
No
끝
호출이 잘됐는가
Yes
소켓 생성
No
끝
잘생성됐나
Yes
바인딩
listen으로 Queue생성
클라이언트 요청 수락
Print
*Client is trying to connect
-IP : xxx
-Port : xxx
Client is connected
IS_IP_allowed 함수 호출
①


1
access.txt 오픈
읽을게있는가
Yes
1개라인 버퍼에저장
Match ip형식에전달
1개라인과 ip를 비교
일치하는가?
No
allow=0
Yes
allow=1
allow가 있는가
1
0
Rejection
cli에전달
끝
Accept cli에전달
log_outh()호출
2


②
클라이언트로 부터
아이디, 비밀번호를 받음
User_match 함수호출
받은 ID, Password가
Passwd 파일에 존재하는가
Yes
No
Client에 ok 전달
Client에 FAIL 전달
Print
success log-in
logfail Print
끝
count가 3인가
No
Yes
FAIL이 아닌
Disconnection
전달
끝

Pseudo code

cli.c

```
void log_in(int sockfd){
    int n
    char user[], *passwd, buf[]
    int attempts = 0

    while(1){
        print "Input ID :" [아이디 입력]
        print "Input Password" [비밀번호 입력]
        아이디와 비밀번호를 결합  -> ID : Passward
        결합한 것을 서버에 전달

        서버로부터 flag 전달 받음
        if(버퍼가 OK 면){
            서버로부터 flag를 전달 받음

            if(flag= OK){
                print "** User ['user'] logged in **"
                return
            }
            else if(flag= FAIL){
                print "** Log-in failed **"
            }
            else if(flag=DISCONNECTION){
                print "** Connection closed **"
            }
            else{
                print "** Unknown response from server **"
            }
        }

    }
    close filediscriptor
    exit
}
```

```
int main(int argc, char *argv[]){
    int sockfd, n
    struct sockaddr_in servaddr
    char buf[]

    if(argc != 3){
        error massage "Usage: <IP address> <Port number>"
        exit
    }
    sockfd = socket(AF_INET, SOCK_STREAM, 0)
    if socket make error-> print errno

    servaddr :family ,addr,port setting

    서버로부터 연결
    만약 연결을 실패했다면 print errno

    if(서버로부터 REJECTION을 받았다면){
     print "** Connection refused **"
     exit
    }
    else if(서버로부터 ACCEPTED를 받았다면){
        print "** It is connected to Server **"
        log_in(sockfd)함수 호출
    }

    close sockfd
    return 0
}
```

Srv.c

```
typedef struct {
    char username[MAX_BUF] //이름
    char password[MAX_BUF]  //비밀번호
    char user_id[MAX_BUF]   //유저 아이디
    char group_id[MAX_BUF]  //그룹 아이디
    char real_name[MAX_BUF] // 실제 이름
    char home_directory[MAX_BUF]  //홈 디렉토리
    char shell_program[MAX_BUF] //셸 프로그램
} User; //정보를 저장하기 위한 구조체

User users[]
int user_count = 0

int load_users(const char *filename) {
    FILE *fp
    struct passwd *pwd

    인자로 받은 파일을 읽기 형태로 open
    만약 실패했다면 ->print errno

    while(fgetpwent 함수를 통해 file 탐색-> null이면 종료){
        탐색하며 구조체 변수에 대한 내용들을 useres배열에 저장
        user_count++
    }
    파일닫기
    password database 닫기
    return 1
}
```

```
int user_match(char *user, char *passwd) {

    for(user_count까지 반복){
    access.txt로 부터 받은 name과 passwd를
    user 배열 name과 passwd를 비교
    }
    if(user 배열 name과 passwd가 같은 access.txt 내용이 있다면){
        return 1 ->인증성공
    }
    return 0->인증 실패
}

int match_ip(char *ip, char *pattern) {
    char ip_copy[]=ip[]
    char ip_parts[4][4], pattern_parts[4][4]
    char *ip_token, *pattern_token

    ip를 .기준으로 파싱하여 ip_token 배열에 저장
    pattern를 .기준으로 파싱하여 ip_pattern배열에 저장

    id passwd가 아무것도 입력되지 않았다면 ""를 각각 추가

    패턴 형태가 정확하지 않다면 에러

}
```

```
int is_ip_allowed( char *ip) {
    FILE *fp
    char line[]
    access.txt를 읽기 권한으로 오픈
    오픈 실패시 에러 출력 후 return

    int allowed = 0; // 허용 상태를 저장할 변수 초기화

    while(access.txt를 끝까지 읽을때까지 반복){
        한줄을 읽어 mathip함수에 전달
        만약 allwoed가 1이라면 탈출
    }
    if (allowed)
        return 1;
    else
        return 0;
}
```

```
int log_auth(int connfd) {
    char user[],passwd[],user_passwd[]
    int n, count=0
    while(1){
        클라이언트로 부터 id와 passwd를 read
        read에러시 print errno 후 return;

        id:passwd형태를 각각 user와 passw변수에 저장

        클라이언트에게 OK표시를 write
        print"** User is trying to log-in ((0부터 증가)/3) **"

        user_match함수에 name,passwd를 전달하여 호출
        if(user_match함수에 반환값이 1이라면){
            클라이언트에게 OK flag를 보냄
            print "** User '%s' logged in **"
            return 1
        }
        else if(3회 이상 실패했다면){
            클라이언트에게 DISCONNECTION flag보냄
            return 0;
        }
        else{
            클라이언트에게 FAIL flag를 보냄
            print "**Log-in faild**"
            count++
        }
    }
    return 1
}
```

```
int main(int argc, char *argv[]) {
    if (argc != 2) {
        print "Usage: <Port number>"
        exit
    }
    if (!load_users("passwd")) {
        print "Error: Failed to load users from passwd"
        exit(1);
    }

    int listenfd, connfd
    struct sockaddr_in servaddr, cliaddr
    socklen_t clilen
    char client_ip[INET_ADDRSTRLEN]
    int client_port

    listenfd = socket(AF_INET, SOCK_STREAM, 0)
    소켓 생성 에러 발생시 print errno ->exit

    servaddr family,addr,port 설정

    바인딩 후 바인딩 에러시 print errno ->exit

    listen으로 queue생성 (size=5)

    for (;;) {
        클라이언트로 부터 accept
        accept 에러 발생시 print errno->exit
       print this
        "   ** Client is trying to connect **
            - IP: 127.0.0.1
            - Port: 49780
            ** Client is connected **"

        if(!is_ip_allowed(client_ip)){
            클라이언트에게 REJECTION을 전송
            print "** It is NOT an authenticated client **"
            close fd
            continue
        }
        else{
            클라이언트에게 ACCEPTED를 전송
        }

        if (log_auth(connfd) == 0) {
            print "** Fail to log-in **"
            continue
        }
        print "** Success to log-in **"
        close fd
    }
    close listenfd
    return 0
}
```

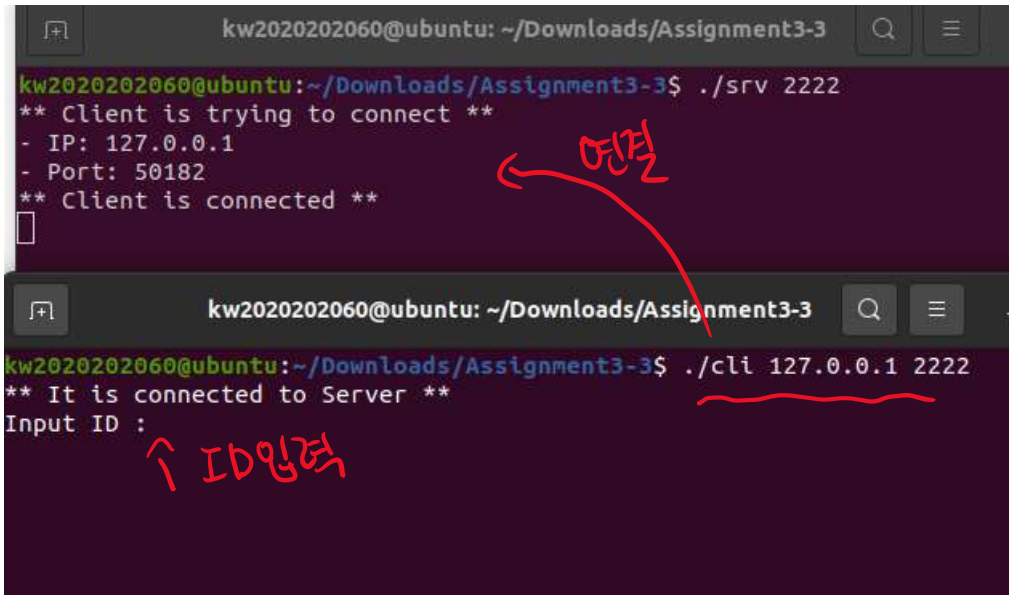
결과화면

먼저 테스트를 진행하기위한 passwd 파일은 다음과 같다.

```
1 test1:12:0:0:kw2020202060:/home/1:sh1
2 test2:34:1:0:kw2020202060:/home/2:sh2
3 test3:56:2:0:kw2020202060:/home/3:sh3
```

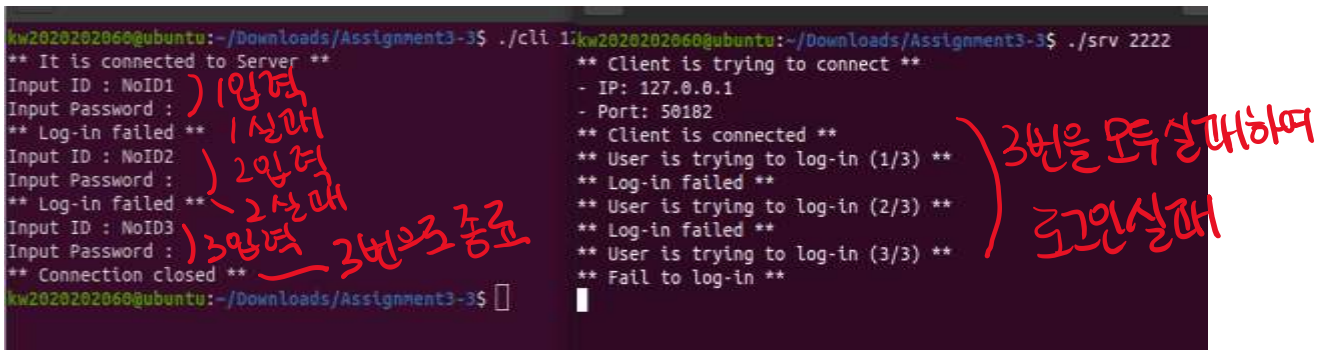
총 test 1 2 3 id 와 passwd 는 차례대로 12,34,56 으로 설정되어 있는 것을 확인할 수 있다.

Access.txt 는 127.0.0.1 로 정의되어 있는 상태이다.



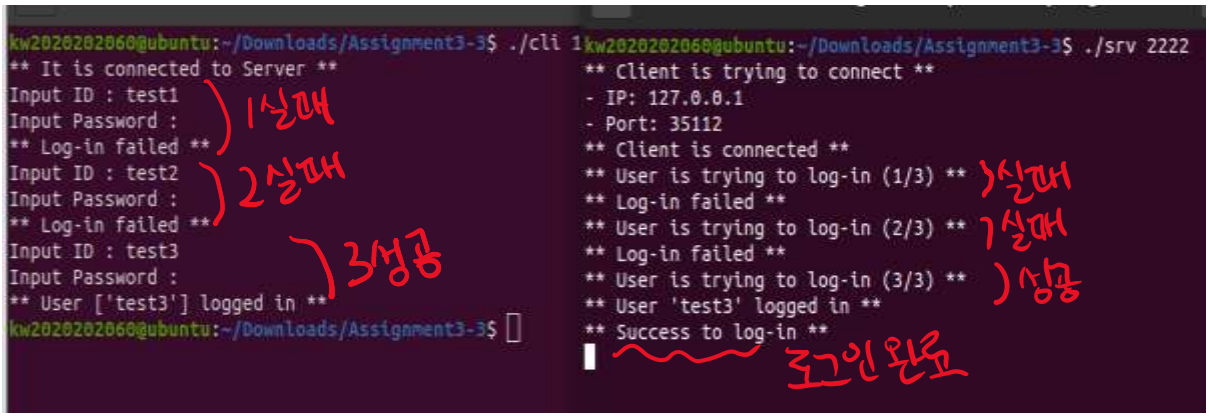
위를 확인하면 서버와 클라이언트 연결을 정상적으로 하였을 때 연결 출력 형태가 모두 올바른 것을 확인할 수 있다. 이때 클라이언트는 ID 를 입력한다.

아래는 아이디와 비밀번호를 3 번 연속 틀렸을 때 횟수 초과로 종료되는 case 를 나타낸다.



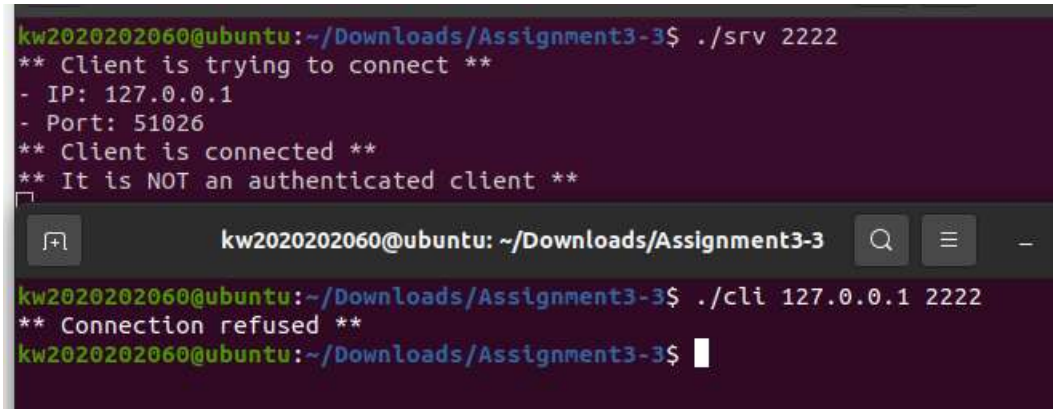
설명과 함께 위를 확인한다면 3 번모두 실패이기에 클라이언트는 종료되는 형태를 확인할 수 있다.

다음은 로그인을 성공하였을 때 대한 case 검증 단계이다.



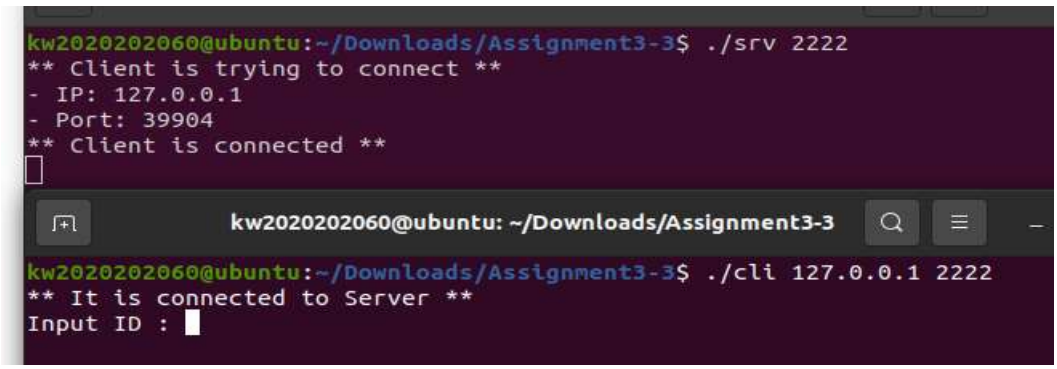
위를 확인하면 두 번은 실패했지만, 마지막에 test3 에 대한 로그인이 성공하여 로그인이 완료된 형태를 확인할 수 있다. 이때 로그인이 성공하면 클라이언트는 종료되는 것을 확인할 수 있다.

다음은 접속이 불가한 IP 를가진 client 에 경우에 대한 검증이다. Access.txt 는 비어 있는 상태이다.



위를 확인하면 IP 가 허용되지 않아 오류와 함께 종료되는 형태를 볼 수 있다.

다음은 access.txt 가 모두 *로 와일드 카드에 대한 접속 혀용에 대한 검증이다. (*.*.*.*)



위를 확인하면 모두 '*' 로 표현되어 있지만 서버와 연결된 형태를 확인할 수 있다.

고찰

이번 프로젝트를 통해 일상에서 흔히 볼 수 있는 아이디와 비밀번호 관리 시스템이 실제로 어떤 방식으로 구현되는지를 깊이 이해하는 기회가 되었다. 클라이언트가 입력한 아이디와 비밀번호를 서버가 검증하고 인증하는 과정에서 다양한 기술적인 문제들을 접하게 되었고, 이를 해결하면서 많은 것을 배웠다.

특히, read 함수와 관련된 오류는 많은 시간을 소요했다. 문제는 read 함수를 두 번 호출할 때 발생했는데, 첫 번째 read 함수 호출에서 모든 데이터가 읽혀 버리면 두 번째 read 호출 시에는 읽을 데이터가 남아있지 않아 빈 값이 반환되는 현상이 발생했다. 이로 인해 흐름 순서가 이상해진 것이다.

이 문제를 해결하기 위해 read 함수에 대해 처음부터 다시 공부하였다. read 함수의 동작 원리와 데이터를 어떻게 읽어드리는지에 대해 깊이 이해한 후, 문제의 원인을 파악할 수 있었다. 결국, 연속적으로 read 함수를 호출하는 대신, write 과정을 한 번 거치고 나서 read 를 호출하는 방식으로 순차적으로 데이터를 처리하는 방법을 사용하였다. 이를 통해 데이터 소실 문제를 해결하고, 안정적이게 되었다.

이번 경험을 통해 소켓 프로그래밍의 기본 원리와 read, write 함수의 사용법을 더 깊이 이해하게 되었으며, 실제 네트워크 프로그래밍에서 발생할 수 있는 다양한 문제들을 해결하는 능력을 키웠다. 또한, 이러한 문제 해결 과정을 통해 디버깅과 문제 분석 능력이 향상되었다. 이 프로젝트에서 네트워크 프로그래밍에 대한 자신감을 얻어, 향후 유사한 문제에 직면했을 때 효율적이게 해결할 수 있을 것이라고 생각했다.