시스템 프로그래밍 실습

9차 과제

실습 일시 : 화 1,2
담당 교수님 : 김태석 교수님
학번 : 2013722095
이름 : 최재은
실습 번호 : practice #3-1

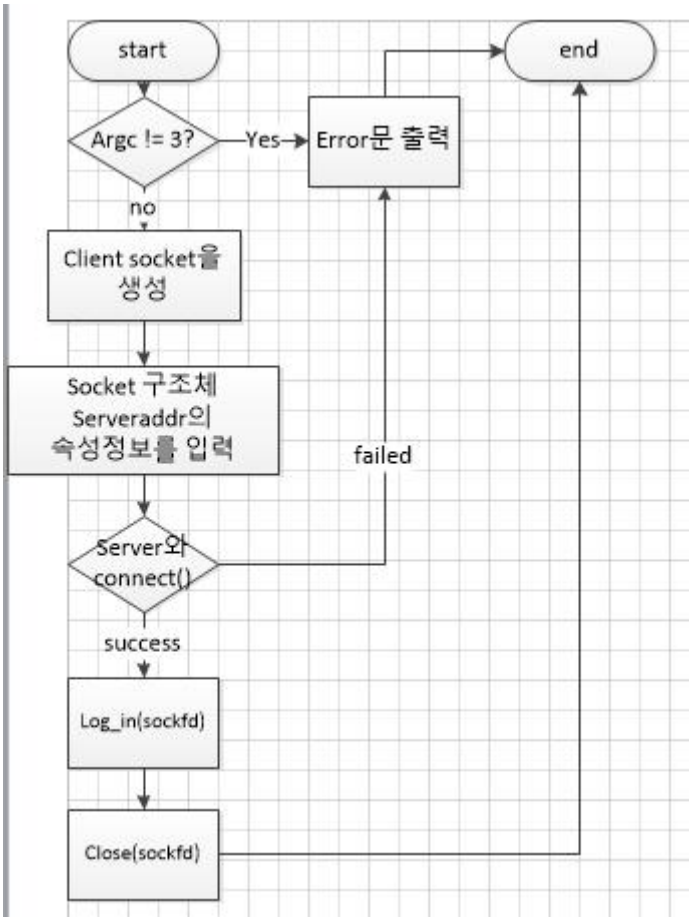
■ Introduction

socket을 이용한 client/server 통신에 허용된 IP만 접근이 가능하도록 프로그램을 구현해본다. 또한 사용자로부터 ID와 PASSWD를 입력받아 미리 등록된 사용자만 FTP 통신을 이용할 수 있도록 제한사항을 구현한다. 이에 필요한 getpass(), wild card등에 대하여 이해하고 활용해 본다.

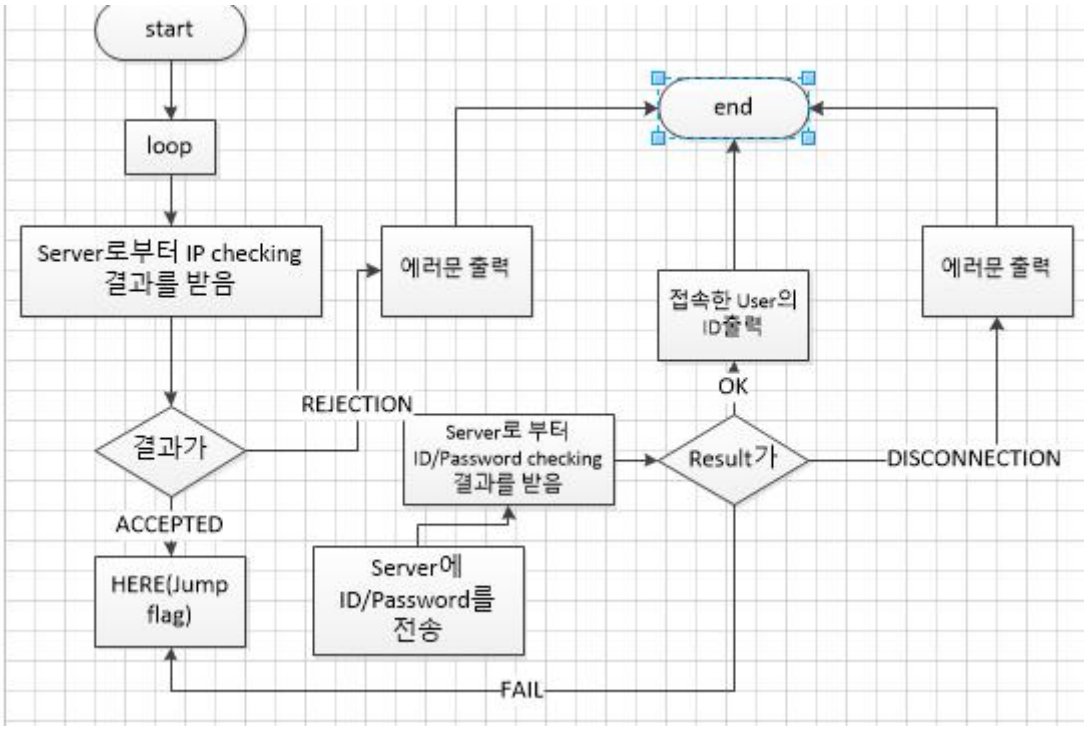
■ Flow Chart

1. cli.c

1) main

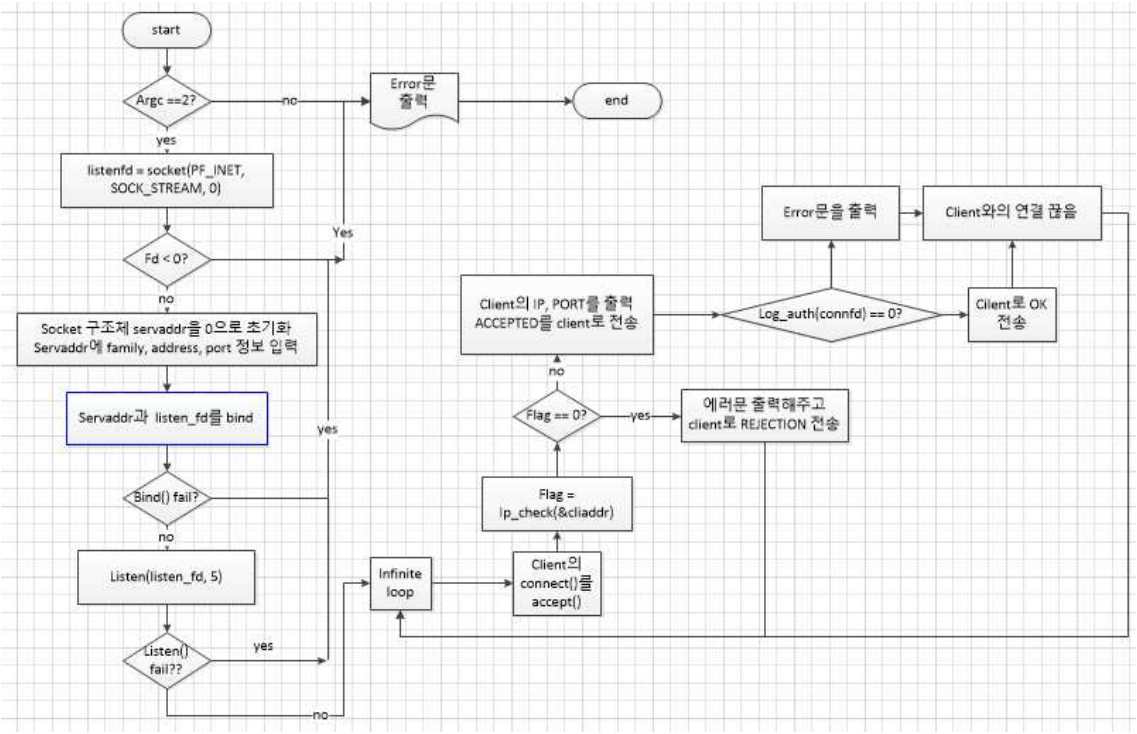


2) log_in

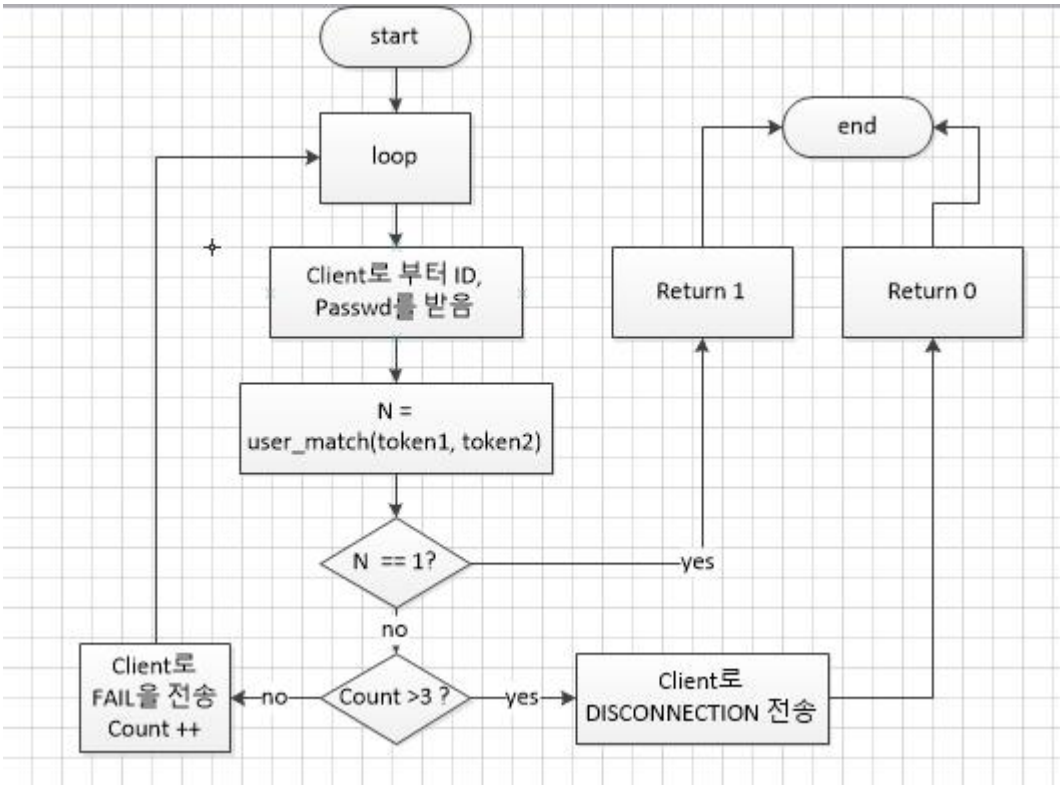

start
loop
Server로부터 IP checking 결과를 받음
에러문 출력
end
에러문 출력
접속한 User의 ID출력
OK
REJECTION
결과가
Server로 부터 ID/Password checking 결과를 받음
Result가
DISCONNECTION
ACCEPTED
HERE(Jump flag)
Server에 ID/Password를 전송
FAIL

2. srv.c

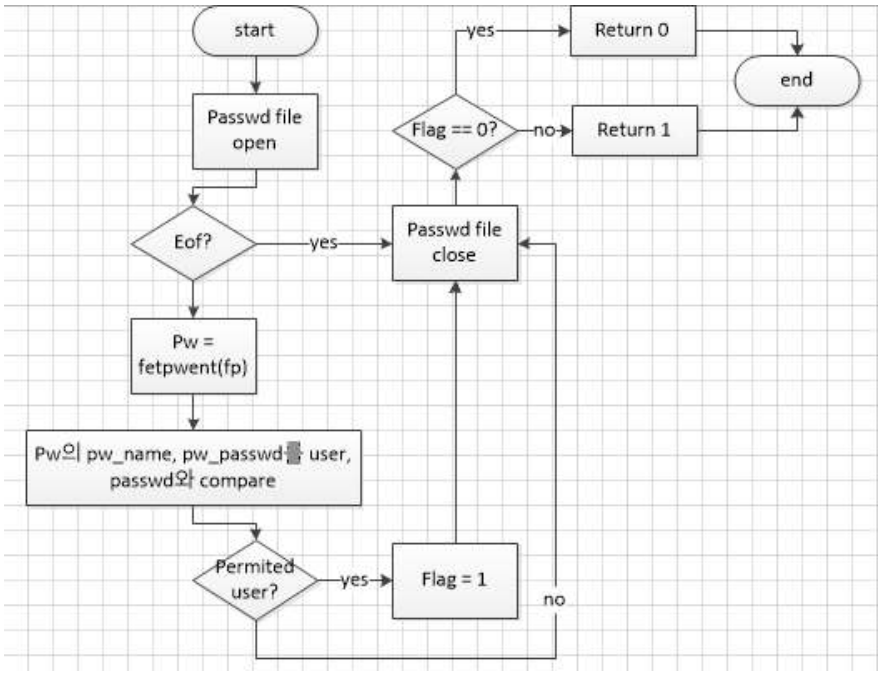
1) main



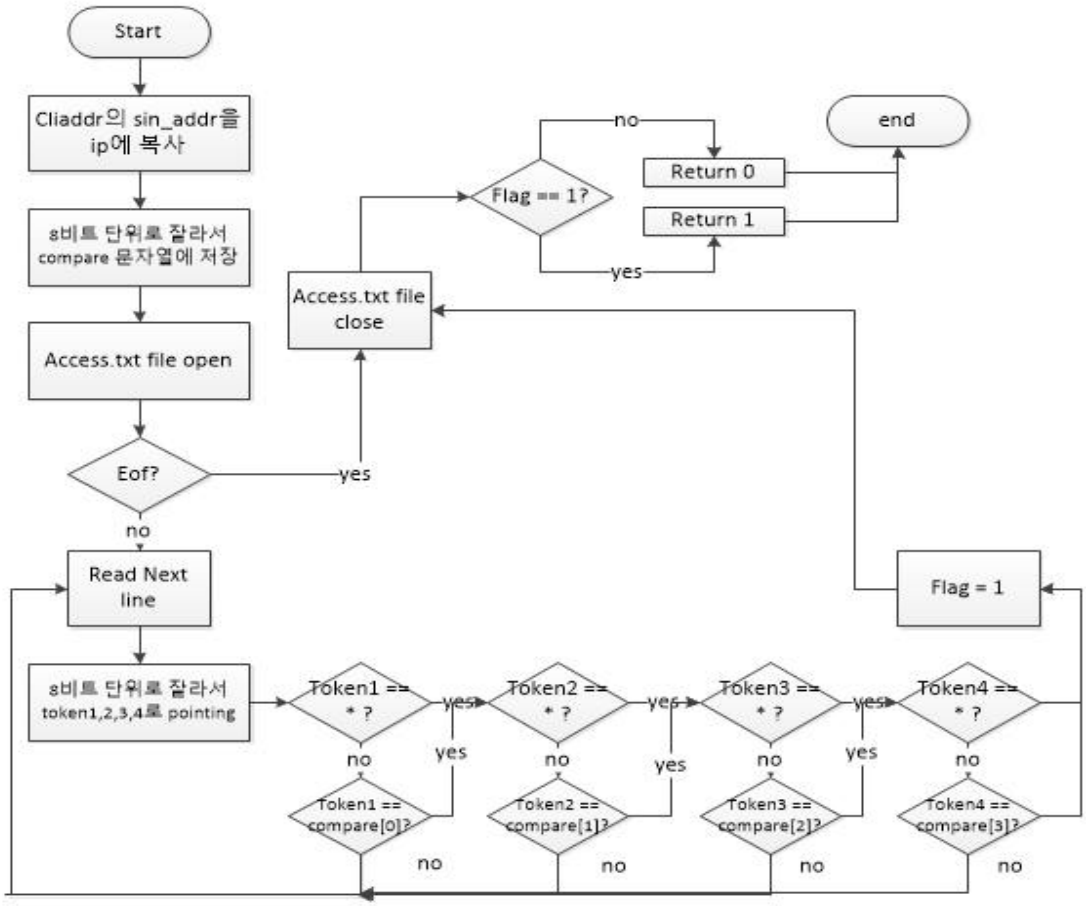
2)log_auth



3) user_match



4) ip_check



■ Source Code

1. cli.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <sys/wait.h>

#include <arpa/inet.h>
#include <netinet/in.h>
#define MAX_BUF 20
#define CONT_PORT 20001

void log_in(int);

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

        if(argc != 3)
        {
                printf("error : confirm arguments₩n");
                return -1;
        }

        /* make client socket */
        sockfd = socket(AF_INET, SOCK_STREAM, 0);

        /* initialize server socket structure */
        memset(&servaddr, 0 , sizeof(servaddr));
        servaddr.sin_family = AF_INET;
        servaddr.sin_addr.s_addr = inet_addr(argv[1]);
        servaddr.sin_port = htons(atoi(argv[2]));


        if(connect(sockfd,(struct sockaddr*)&servaddr, sizeof(servaddr)) < 0)
        {
                printf("error : failed to connect with server₩n");
                return -1;
```

```
	}

	log_in(sockfd);	// log in the conneted server with ID/Password

	close(sockfd);

	return 0;
}

void log_in(int sockfd)
{
	int n;
	char user[MAX_BUF], *passwd, buf[MAX_BUF];

	for(;;)
	{
		n = read(sockfd, buf, MAX_BUF);		// recieve IP checking result
		buf[n] = '₩0';

		if(!strcmp(buf, "ACCEPTED"))		// permited IP
		{
			printf("** It is connected to Server **₩n");

HERE:	// for re-type ID/PASSWD about 3time
			memset(user, 0, sizeof(user));	// buffer initialize

			/* insert ID/PASSWD from user */
			write(STDOUT_FILENO, "Input ID :", strlen("Input ID :"));
			n = read(STDIN_FILENO, user, MAX_BUF);
			user[n-1] = ' '; // remove enter

			passwd = getpass("Input passwd : ");

			sprintf(buf, "%s%s",user, passwd);
			write(sockfd, buf, sizeof(buf));		// send it to server

			n = read(sockfd, buf, MAX_BUF);		// recieve ID/PASSWD
checking result
			buf[n] = '₩0';

			if(!strcmp(buf, "OK"))		//permited
ID/PASSWD
			{
				printf("** User '%s' logged in **₩n", user);
				break;
			}
```

```
                        else if(!strcmp(buf, "FAIL"))               // unpermited ID/PASSWD
                        {
                                printf("** Log-in failed **₩n");
                                goto HERE;
        // go to re-type ID/PASSWD
                        }
                        else        // DISCONNECTION
                        {
                                printf("** Connection closed **₩n");
                                break;
                        }

                }
                else // REJECTION
                {
                        printf("** Connection refused **₩n");
                        exit(0);
                }
        }
}
```

2. srv.c

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pwd.h>

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <sys/wait.h>

#include <arpa/inet.h>
#include <netinet/in.h>


#define MAX_BUF 20

int log_auth(int);
int user_match(char*, char*);
int ip_check(struct sockaddr_in*);
int main(int argc, char *argv[])
{
```

```
	int listenfd, connfd;
	int ip_check_flag, len;
	struct sockaddr_in servaddr, cliaddr;
	FILE *fp_checkIP;
	if(argc!=2)
	{printf("error : check arguments₩n"); return -1;}

	/* create socket */
	if((listenfd = socket(PF_INET, SOCK_STREAM, 0)) < 0)
	{
		write(STDOUT_FILENO, "error : socket() is failed!₩n", strlen("error : socket() is failed!₩n"));
		return -1;
	}

	/* set server socket's information */
	memset(&servaddr, 0 , sizeof(servaddr));
	servaddr.sin_family = AF_INET;
	servaddr.sin_addr.s_addr = htons(INADDR_ANY);
	servaddr.sin_port = htons(atoi(argv[1]));

	/* bind server socket with address */
	if(bind(listenfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0)
	{
		write(STDOUT_FILENO, "error : bind() is failed!₩n", strlen("error : bind() is failed!₩n"));
		return -1;
	}

	/* prepare connection */
	if(listen(listenfd, 5) < 0)
	{
		write(STDOUT_FILENO, "error : listen() is failed!₩n", strlen("error : listen() is failed!₩n"));
		return -1;
	}


	for(;;)
	{
		len = sizeof(cliaddr);
		/*connect with client*/
		connfd = accept(listenfd, (struct sockaddr*)&cliaddr, &len);	//	connect server with client

		printf("** Client is trying to connect **₩n");
```

```
                /* check 'client's IP can access '*/
                ip_check_flag = ip_check(&cliaddr);

                if(ip_check_flag == 1)          // permited IP
                {
                        printf(" - IP   : %s₩n", inet_ntoa(cliaddr.sin_addr));
                        printf(" - PORT : %d₩n", ntohs(cliaddr.sin_port));
                        write(connfd, "ACCEPTED", strlen("ACCEPTED")); // send ACCEPTED to
client
                }
                else
                {
                        write(connfd, "REJECTION", strlen("REJECTION"));          //          send
REJECTION to client
                        printf("** It is NOT authenticated client **₩n");
                        close(connfd);
                        continue;  // or return 0;
                }

                if(log_auth(connfd) == 0)
                {
                        printf("** Fail to log-in **₩n");
                        close(connfd);
                        continue;
                }

                /* success to log-in */
                write(connfd, "OK", strlen("OK"));
                close(connfd);
        }

}

int log_auth(int connfd)
{
        char user[MAX_BUF];
        char *token1, *token2;
        int n, count = 1;

        while(1)
        {
                /* recieve username and password from client*/
                read(connfd, user, MAX_BUF);
                token1 = strtok(user, " ");
                token2 = strtok(NULL, " ");
```

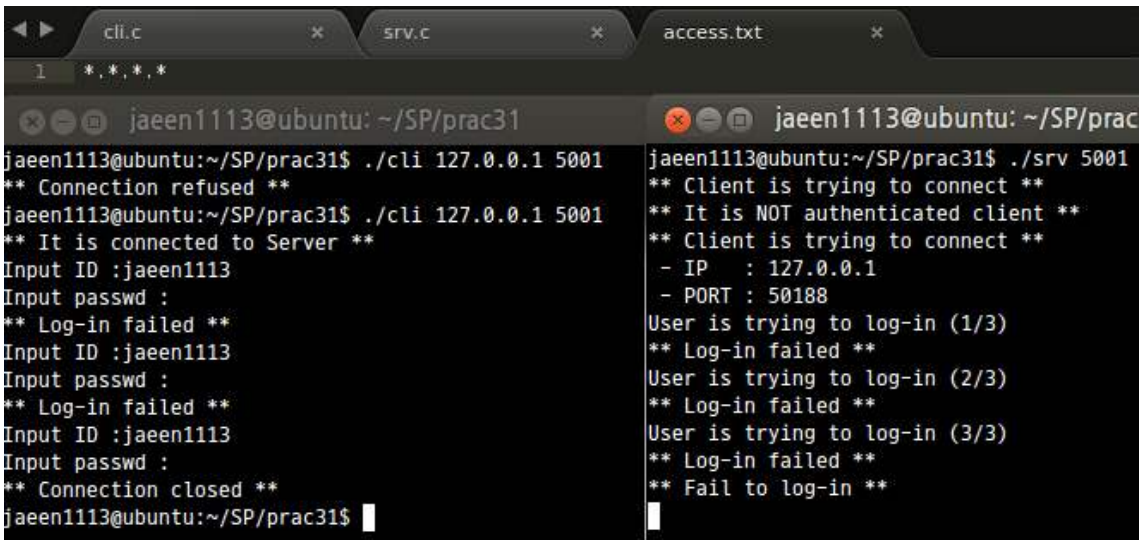
```
                printf("User is trying to log-in (%d/3)₩n", count);

                /* ID/PASSWORD checking*/
                if((n = user_match(token1, token2))==1)
                {
                        printf("** Success to log-in **₩n");
                        break;
                }

                else if(n == 0)     // failed to log-in
                {
                        if(count >= 3)// failed more than 3 times, disconnect client
                        {
                                printf("** Log-in failed **₩n");
                                write(connfd, "DISCONNECTION", strlen("DISCONNECTION"));
                                return 0;

                        }
                        //forgive it about 3 times
                        write(connfd, "FAIL", strlen("FAIL"));
                        printf("** Log-in failed **₩n");
                        count++;
                        continue;
                }

        }
        return 1;

}


/*checking Inserted ID&PASSWD is accessable or not*/
int user_match(char *user, char *passwd)
{
        FILE* fp;
        int flag = 0;
        struct passwd *pw;
        fp = fopen("passwd", "r");

        while(!feof(fp))
        {
                pw = fgetpwent(fp);          // get passwd structure info

                if(!strcmp(user, pw->pw_name) && !strcmp(passwd, pw->pw_passwd))
                {
```

```
                        flag=1;
                        break;
                }
                else
                        continue;
        }

        fclose(fp);
        if(flag == 0)
                return 0;
        else
                return 1;
}



int ip_check(struct sockaddr_in* cliaddr)
{
        FILE* fp;
        int flag = 0;
        char ip[50];
        char access[50];
        char temp[50];
        char *token1, *token2, *token3, *token4;
        char compare[4][10];
        strcpy(ip, inet_ntoa(cliaddr->sin_addr));

        /*get client's Dotted IP address*/
        sprintf(temp, "%s", strtok(ip, "."));
        strcpy(compare[0], temp);
        sprintf(temp, "%s", strtok(NULL, "."));
        strcpy(compare[1], temp);
        sprintf(temp, "%s", strtok(NULL, "."));
        strcpy(compare[2], temp);
        sprintf(temp, "%s", strtok(NULL, " "));
        strcpy(compare[3], temp);


        fp = fopen("access.txt", "r");              // open access file
        while(!feof(fp))
        {

                fgets(access, sizeof(access), fp); // get line and seperate it

                token1 = strtok(access, ".");
```

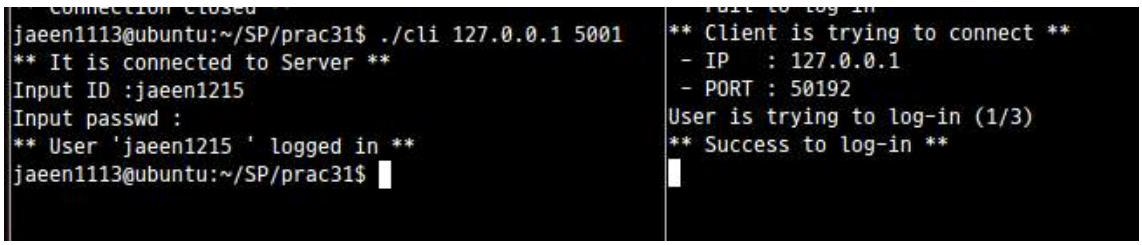
```
                token2 = strtok(NULL, ".");
                token3 = strtok(NULL, ".");
                token4 = strtok(NULL, " ");

                if(strcmp(token1, "*") != 0)  // if first 8bit isn't a wild card
                {
                        if(strcmp(token1, compare[0]) != 0) // not matched first 8bit
                                continue;
                }

                if(strcmp(token2, "*") != 0)  // if second 8bit isn't a wild card
                {
                        if(strcmp(token2, compare[1]) != 0) // not matched second 8bit
                                continue;
                }

                if(strcmp(token3, "*") != 0)  // if third 8bit isn't a wild card
                {
                        if(strcmp(token3, compare[2]) != 0) // not matched first 8bit
                                continue;

                }

                if(strcmp(token4, "*") != 0)  // if forth 8bit isn't a wild card
                {
                        if(strcmp(token4, compare[3]) != 0) // not matched first 8bit
                                continue;
                }

                flag = 1;
                break;
        }
        fclose(fp);

        if(flag == 1)
                return 1;

        else
                return 0;

}
```

■ Result **(왼 : client / 오 : server)**

- **허가되지 않은 IP에 대해서는 connection이 거부 된다.**



- **connection에 성공하면 ID와 PASSWD를 입력하는데 이때 2번까지 봐준다.**
- **3번째 입력에서도 실패하면 client를 꺼버린다.**



- **올바른 ID/PASSWD를 입력하면 위처럼 연결이 되었음을 보여 주는 화면이 나온다.**