[CYDF321-P1] 2023150225 - 오정민

1. Project 목적
2. 요구 사항
3. 실행 환경, 프로그램 빌드 및 실행 방법
4. BSD socket API 및 TCP와의 상호작용에 대한 설명
5. Code detail
6. 실행 예시

1. Project 목적

1. BSD socket에서 TCP가 구현되는 방식을 이해한다.
2. socket API를 이용해 Network Application을 구현하는 방법을 이해한다.

2. 요구 사항

1. 환경
    - POSIX 표준을 지키는 운영체제(예: Linux, macOS, UNIX)에서 C언어를 이용해 작성해야한다.
    - BSD sokcket API를 client와 server 프로그램이 통신하도록 사용한다.
    - 프로그램은 BSD socket의 함수들 (예 : socket, bind, close ...) 을 포함해야한다.
2. Chat Server
    - 서버 프로그램은 반드시 포트 번호 하나만을 command-line 인자로 가져야한다.

      즉, 서버를 실행할 때 사용하는 인자는 하나이며, 그 인자는 포트 번호이다.
    - 서버는 하나의 TCP 클라이언트만 연결을 허용한다.
    - TCP 연결이 된 이후, 클라이언트의 IP 주소와 포트 번호가 화면에 표준 출력으로 출력된다.

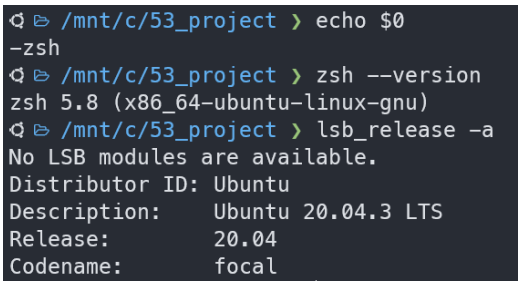
      (예 : “Connection from 163.152.162.144:12345”)
    - 클라이언트에서 받은 메시지는 화면에 출력되어야 한다.

      그 후 서버는 사용자가 입력한 문자열을 클라이언트로 전송한다.
    - 소켓을 통한 수신과 전송은 동시에 발생하지 않는다. 즉, multi-threading을 적용하지 않아도 된다.
    - 채팅 프로그램은 사용자가 “QUIT”을 입력할 때 까지 계속된다. 서버가 “QUIT” 메시지를 받거나 전송하게 되면, “Disconnected”가 화면에 출력되고, 소켓이 닫히며 프로그램이 종료된다.
3. Chat Client
    - 클라이언트는 command-line 인자로 서버의 IP 주소와 서버가 지정하는 포트 번호를 갖는다.
    - 클라이언트는 위의 인자를 이용해 TCP 연결을 진행한다.

- 연결에 성공하면 “Connected” 메시지를 화면에 출력한다.
- 클라이언트가 먼저 사용자의 입력값을 서버에 메시지로 보낸다.
- 클라이언트는 이후 서버의 메시지를 수신하고 표준 출력으로 화면에 출력한다.
- 위의 수신, 송신을 프로그램 종료시 까지 반복한다.
- 채팅 프로그램은 사용자가 “QUIT”을 입력할 때 까지 계속된다. 클라이언트가 “QUIT” 메시지를 받거나 전송하게 되면, “Disconnected”가 화면에 출력되고, 소켓이 닫히며 프로그램이 종료된다.

3. 실행 환경, 프로그램 빌드 및 실행 방법

- 실행 환경

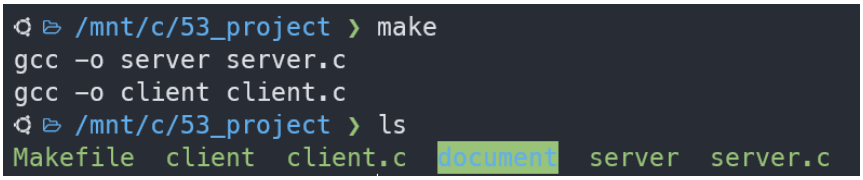


실행 환경은 우분투 20.04 에 유닉스 쉘 중 하나인 zsh을 설치해 진행하였다.

해당 정보는 위의 사진과 같다.

- 프로그램 빌드 및 실행 방법

```
compile :
  gcc -o server server.c
  gcc -o client client.c
clean :
  rm client server
```



위와 같이 Makefile을 만들고 make 명령어를 이용해 컴파일 하였다.

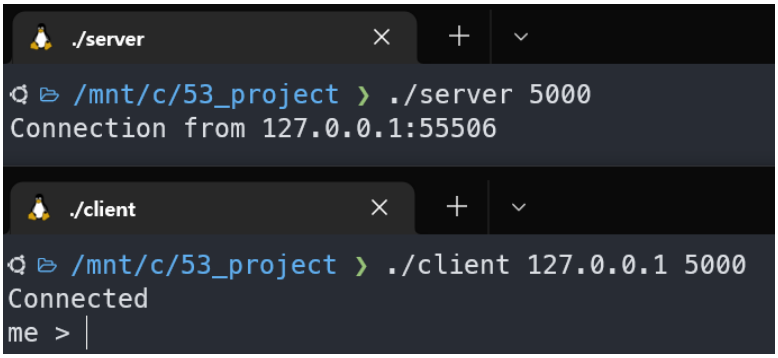
```
[실행 방법]
1. server
  ./server <port number>
2. client
  ./client <server ip> <port number>
```

위와 같이 적절한 인자를 이용해 실행한다.

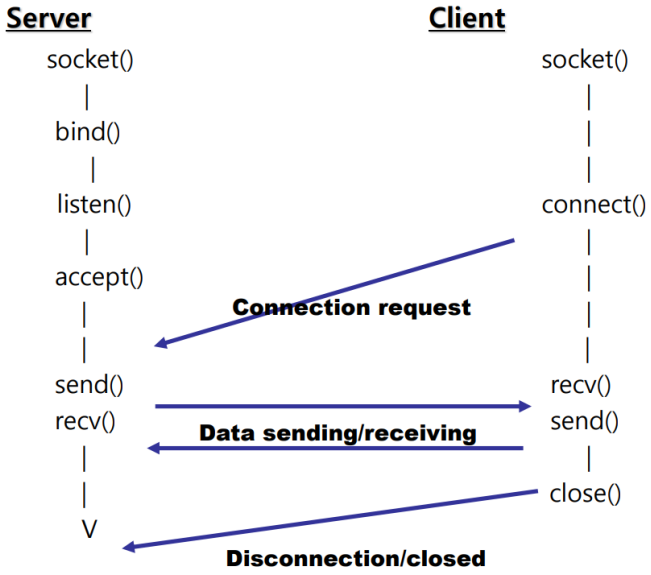
가상 머신을 이용해 두 환경을 이용하여 실행하는 경우 server를 실행하는 환경의 ip 주소를 client 실행시에 인자로 넘겨주면 된다.

하지만, 현재 zsh 하나 만을 이용할 것이므로, localhost인 127.0.0.1을 인자로 넘겨준다.

아래와 같이 두 창을 이용해 실행하면 된다.



4. BSD socket API 및 TCP와의 상호작용에 대한 설명



socket API를 이용한 TCP 통신 구현은 위의 그림과 같이 이루어진다.

순서대로 설명하면 아래와 같다.

1. server 와 client에서 socket 함수를 이용해 각각 소켓을 생성함.
2. server에서 bind 함수를 통해 서버 소켓에 주소를 할당함.
3. server에서 listen 함수를 통해 서버 소켓을 연결 대기 상태로 전환함.
4. client에서 listen 상태에 있는 서버 소켓에 connect 함수를 이용해 연결 요청을 함.
5. server에서 accept 함수를 통해 받은 요청을 수락함.

6. 연결이 된 이후 send / recv 함수를 이용해 각각 메시지를 보냄.
7. 연결이 종료된 후 close 함수를 통해 각각 소켓을 닫음.

위의 과정에서 TCP connection이 만들어지는 과정은 3, 4, 5번 과정이다.

server와 client에서 사용되는 함수를 각각 살펴보자.

<Server, Clinet 공통>

- socket()

```
header file : <sys/socket.h>, <sys/types.h>, <netinet/in.h>, <arpa/inet.h>
[함수 원형]
int socket(int family, int service, int protocol);
```

함수의 원형은 위와 같다. 반환값은 소켓을 가리키는 소켓 디스크립터(socket descriptor)를 반환한다.

에러 발생시 -1을 발생시킨다.

함수의 인자는 protocol family, service type, protocol이 들어간다.

각 인자는 아래와 같다.

1. `int family` (domain) : 어떤 영역에서 통신할 것인지에 대해 영역을 지정해준다. 우리는 IPv4를 이용할 것이므로, 해당 인자에 PF_INET을 넣어주면 된다.
2. `int service` : 어떤 서비스 타입의 프로토콜을 사용할 것인지 지정해준다. 우리는 TCP 통신을 이용하므로, stream socket인 SOCK_STREAM을 넣어주면 된다.
3. `int protocol` : 어떤 통신 프로토콜을 사용할 것인지 지정한다. 우리는 TCP 통신을 이용하므로, IPPROTO_TCP를 넣어주면 된다.

parameter	value	
domain	PF_INET	IPv4 Internet protocol family
	PF_INET6	IPv6 Internet protocol family
	PF_LOCAL	Local UNIX Socket program family
	PF_UNIX	Local UNIX Socket program family
type	SOCK_STREAM	Stream Socket
	SOCK_DGRAM	Datagram Socket
	SOCK_RAW	Raw Socket
	SOCK_SQPACKET	Sequence packet Socket
protocal	IPPROTO_TCP	Stream Socket
	IPPROTO_UDP	Datagram Socket

각 인자에 대해 들어갈 수 있는 값은 위의 표와 같다.

- close()

```
header file : <unistd.h>
[함수 원형]
int close(int socket); //socket : socket descriptor
```

해당 소켓 디스크립터가 가리키는 소켓을 종료시킨다.

- send() / recv()

```
header file : <sys/socket.h>, <sys/types.h>
[함수 원형]
int send(int socket, const void *msg, unsigned int msgLength, int flags);
int recv(int socket, void *rcvBuffer, unsigned int bufferLength, int flags);
```

1. send

   다른 소켓으로 메시지를 보내는데 사용된다. send는 각 소켓이 연결된 상태에서 사용된다.

   각 인자는 아래와 같다,

   a. `int socket` : 데이터를 보낼 대상의 소켓

   b. `const void *msg` : 보낼 메시지의 포인터

   c. `unsigned int msgLength` : 메시지의 길이

   d. `int flags` : 함수 호출 시에 사용할 옵션을 결정한다. 아무 옵션 없이 실행하려면 0을 넣어 준다.

2. 다른 소켓으로 메시지를 보내는데 사용된다. send는 각 소켓이 연결된 상태에서 사용된다.

   각 인자는 아래와 같다,

   a. `int socket` : 메시지를 받을 대상의 소켓

   b. `void *rcvBuffer` : 메시지를 받을 버퍼의 포인터

   c. `unsigned int bufferLength` : 버퍼의 길이

   d. `int flags` : 함수 호출 시에 사용할 옵션을 결정한다. 아무 옵션 없이 실행하려면 0을 넣어 준다.

<Server>

- bind()

```
header file : <sys/socket.h>
[함수 원형]
int bind(int socket, struct sockaddr *localAddress, unsigned int addressLength);
```

bind 함수는 주소 정보를 위에서 생성한 소켓에 할당한다.

각각의 인자는 아래와 같다.

a. `int socket` : 주소를 할당할 소켓의 디스크럽터

b. `struct sockaddr *localAddress` : sockaddr 구조체로 캐스팅 된 앞에서 정의한 sockaddr_in구조체의 주소

이 인자를 구성하는 구조체를 살펴보자.

- 구조체

```
struct sockaddr_in
{
    sa_family_t    sin_family;  //주소체계(Address Family)
    uint16_t       sin_port;    //16비트  PORT번호
    struct in_addr sin_addr;    //32비트의 IP주소
    char           sin_zero[8];
}

struct in_addr{
    in_addr_t    s_addr;  //32비트의 IPv4 인터넷 주소가 담긴다.
}

struct sockaddr
{
    sa_family_t sin_family    //주소체계(Address Family)
    char        sa_data[14];  //주소정보 = IP addr + port
}
```

위의 구조체의 값은 함수를 호출하기 이전에 먼저 초기화 시켜주어야 한다.

c. `unsigned int addressLength` : 2번째 인자에 넣어준 구조체 변수의 크기

반환값은 아래와 같다.

- error 발생 : -1
- 성공 : 0

- listen()

```
header file : <sys/socket.h>
[함수 원형]
int listen(int socket, int queueLimit);
```

주소가 할당된 소켓을 연결 요청 대기상태로 만든다.

각각의 인자는 아래와 같다,

a. `int socket` : 앞에서 주소 할당을 한 소켓의 디스크립터

b. `int queueLimit` : 클라이언트를 최대 몇 개까지 대기상태로 둘 것인지를 정한다.

해당 단계인 서버에 클라이언트는 connect함수를 통해 연결 요청을 보낸다.

반환값은 아래와 같다.

- error 발생 : -1
- 성공 : 0

- accept()

```
header file : <sys/socket.h>
[함수 원형]
int accept(int socket, struct sockaddr *clientAddress, unsigned int *addressLength);
```

listen 상태인 서버 소켓에 대기상태로 들어온 클라이언트 연결 요청을 수락한다.

각각의 인자는 아래와 같다.

a. `int socket` : listen 상태인 서버 소켓의 디스크립터

b. `struct sockaddr *clientAddress` : 연결 요청을 받은 클라이언트의 주소정보를 담는 구조체

c. `unsigned int *addressLength` : 두번째 인자에 넣어준 구조체의 크기를 담고있는 주소

반환값은 아래와 같다.

- error 발생 : -1
- 성공 : accept 된 클라이언트 소켓의 디스크립터

<Client>

- connect()

```
header file : <sys/socket.h>
[함수 원형]
int connect(int socket, struct sockaddr *foreignAddress, unsigned int addressLength);
```

listen 상태인 서버 소켓에 연결 요청을 보낸다.

각각의 인자는 아래와 같다.

a. `int socket` : listen 상태인 서버 소켓의 디스크립터

b. `struct sockaddr *foreignAddress` : sockaddr 구조체로 캐스팅 된 앞에서 정의한 sockaddr_in 구조체의 주소

c. `unsigned int addressLength` : 2번째 인자에 넣어준 구조체 변수의 크기

5. Code detail

두 소스코드에서 사용된 함수인 socket, bind, listen, connect, accept, read, write 는 모두 비정상적인 종료시에 -1을 반환한다. 해당 경우에는 에러에 대해 출력해주고, 프로그램을 종료시켜주었다.

1. server.c

```
#include <sys/socket.h>
#include <sys/stat.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
```

```
#define BUFMAX 1024

int main(int argc, char **argv){
    int server_socket, client_socket;

    struct sockaddr_in client_addr, server_addr;
    int client_addr_size = sizeof(client_addr);

    char buf[BUFMAX];
    memset(buf, 0x00, BUFMAX);

    server_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(server_socket == -1){
        printf("SERVER : bind error\n");
        return 1;
    }

    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family      = AF_INET;                  //IPv4 인터넷 프로토콜
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);        //32bit IPv4 주소 자동으로 배정함.
    server_addr.sin_port        = htons(atoi(argv[1]));     //인자로 port 번호 할당

    if(bind(server_socket, (struct sockaddr *) &server_addr, sizeof(server_addr)) == -1){
        printf("SERVER : bind error\n");
        return 1;
    }

    if(listen(server_socket, 5) == -1){
        printf("SERVER : listen error\n");
        return 1;
    }

    client_socket = accept(server_socket, (struct sockaddr *) &client_addr, &client_addr_size);
    if(client_socket == -1){
        printf("SERVER : accept error\n");
        return 1;
    }

    printf("Connection from %s:%d\n",inet_ntoa(client_addr.sin_addr),client_addr.sin_port);

    while(1){
        if(recv(client_socket, buf, BUFMAX, 0) == -1){
            printf("SERVER : recv error");
            close(client_socket);
            close(server_socket);
            return 0;
        }

        if(!strcmp(buf, "QUIT")){
            printf("client : %s\n", buf);
            printf("Disconnected\n");
            close(server_socket);
            return 0;
        }

        printf("client : %s\n", buf);

        memset(buf, 0x00, BUFMAX);
        printf("me > ");
        fgets(buf, BUFMAX, stdin);
        buf[strlen(buf) - 1] = '\0';
        if(!strcmp(buf, "QUIT")){
            if(send(client_socket, buf, BUFMAX, 0) == -1){
                printf("SERVER : write error\n");
```

```
                return 1;
            }
            printf("Disconnected\n");
            close(client_socket);
            close(server_socket);
            return 0;
        }

        if(send(client_socket, buf, BUFMAX, 0) == -1){
            printf("SERVER : write error\n");
            return 1;
        }
    }

    close(client_socket);
    close(server_socket);
    return 0;
}
```

서버의 전체 소스코드는 위와 같다.

부분별로 나눠서 코드를 살펴보자.

- 변수 선언

```
int server_socket, client_socket;

struct sockaddr_in client_addr, server_addr;

char buf[BUFMAX];
memset(buf, 0x00, BUFMAX); //BUFMAX = 1024
```

  - `int server_socket, client_socket` : socket 함수의 반환값을 저장할 서버 소켓과 클라이언트 소켓 디스크립터
  - `int str_len` : read함수의 반환값인 수신한 데이터의 크기를 저장할 변수
  - `struct sockaddr_in client_addr, server_addr` : 소켓에 주소와 포트 번호를 할당하기 위한 구조체
  - `char buf[BUFMAX]` : 전송과 수신이 되는 문자열을 저장할 버퍼

- socket()

```
server_socket = socket(AF_INET, SOCK_STREAM, 0);
if(server_socket == -1){
    printf("SERVER : bind error\n");
    return 1;
}
```

  socket 함수를 이용해 소켓을 생성해준다.

- bind()

```
memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;                   //IPv4 인터넷 프로토콜
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);    //32bit IPv4 주소 자동으로 배정함.
server_addr.sin_port = htons(atoi(argv[1]));        //인자로 port 번호 할당.

if(bind(server_socket, (struct sockaddr *) &server_addr, sizeof(server_addr)) == -1){
    printf("SERVER : bind error\n");
    return 1;
}
```

server_addr 구조체를 초기화 해주고, bind 함수를 통해 위에서 생성한 소켓에 주소와 port 번호를 할당한다.

- listen()

```
if(listen(server_socket, 5) == -1){
    printf("SERVER : listen error\n");
    return 1;
}
```

listen 함수를 통해 클라이언트의 접속 요청을 확인한다.

두번째 인자인 queueLimit은 넉넉하게 5로 설정하였다.

- accept()

```
int client_addr_size = sizeof(client_addr);
client_socket = accept(server_socket, (struct sockaddr *) &client_addr, &client_addr_size);
if(client_socket == -1){
    printf("SERVER : accept error\n");
    return 1;
}
```

listen 함수를 통해 확인한 접속 요청을 accept 함수로 승인한다.

accept 함수는 소켓을 생성해준다. 해당 소켓을 client_socket이라고 하자.

- send() / recv()

```
if(recv(client_socket, buf, BUFMAX, 0) == -1){
    close(client_socket);
    close(server_socket);
    return 0;
}
if(!strcmp(buf, "QUIT")){
    printf("client : %s\n", buf);
    printf("Disconnected\n");
    close(server_socket);
    return 0;
}
printf("client : %s\n", buf);
```

해당 부분을 통해 client에서 전송된 데이터를 read로 읽어봐 buf에 저장한다.

만약 read가 비정상적으로 종료되면, -1을 반환하므로 해당 경우에는 소켓을 닫고 종료한다.

읽어온 문자열이 “QUIT”이라면, 출력한 뒤 소켓을 닫고 종료한다.

다른 문자열이라면, 출력해준다.

```
memset(buf, 0x00, BUFMAX);
printf("me > ");
fgets(buf, BUFMAX, stdin);
buf[strlen(buf) - 1] = '\0';
if(!strcmp(buf, "QUIT")){
    if(send(client_socket, buf, BUFMAX, 0) == -1){
        printf("SERVER : write error\n");
        return 1;
    }
    printf("Disconnected\n");
    close(client_socket);
    close(server_socket);
    return 0;
}

if(sned(client_socket, buf, BUFMAX, 0) == -1){
    printf("SERVER : write error\n");
    return 1;
}
```

우선 buf를 초기화 해주고, buf에 문자열을 입력받는다.

fgets를 이용해 입력받으면, 맨 뒤에 개행문자도 저장되므로, 해당 부분만 ‘\0’ 으로 바꿔준다.

입력받은 문자열이 “QUIT”이라면, 클라이언트 소켓으로 write를 이용해 전송하고, 소켓을 닫아준 뒤, 종료한다.

다른 문자열이라면, write를 이용해 클라이언트 소켓으로 전송한다.

- close()

```
close(client_socket);
close(server_socket);
return 0;
```

두 소켓을 닫고, main 함수 종료.

2. client.c

```
#include <sys/socket.h>
#include <arpa/inet.h>
#include <sys/stat.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

#define BUFMAX     1024

int main(int argc, char **argv){
    int sock;
```

```
    struct sockaddr_in server_addr;

    char buf[BUFMAX];
  memset(buf, 0x00, BUFMAX);

    //socket()
    sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(sock == -1){
        printf("CLIENT : socket error\n");
        return 1;
    }

    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family      = AF_INET;                  //IPv4 인터넷 프로토콜
    server_addr.sin_addr.s_addr = inet_addr(argv[1]);       //인자로 서버의 IP 주소 할당
    server_addr.sin_port        = htons(atoi(argv[2]));     //인자로 port 번호 할당

  //connect()
    if (connect(sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1){
        printf("CLIENT : connect error\n");
        return 1;
    }

    printf("Connected\n");

    while(1){
        printf("me > ");
        fgets(buf, BUFMAX, stdin);
        buf[strlen(buf) - 1] = '\0';
        if(!strcmp(buf, "QUIT")){
            if(send(sock, buf, BUFMAX, 0) == -1){
                printf("CLIENT : write error\n");
                return 1;
            }
            printf("Disconnected\n");
            close(sock);
            return 0;
        }

        if(send(sock, buf, BUFMAX, 0) == -1){
            printf("CLIENT : write error\n");
            return 1;
        }

        memset(buf, 0x00, BUFMAX);
        if(recv(sock, buf, BUFMAX, 0) == -1){
            printf("CLIENT : read error\n");
            return 1;
        }

        if(!strcmp(buf, "QUIT")){
            printf("server : %s\n", buf);
            printf("Disconnected\n");
            close(sock);
            return 0;
        }
        printf("server : %s\n", buf);
    }
    close(sock);
    return 0;
}
```

client의 전체 소스코드는 위와 같다.

부분별로 나눠서 코드를 살펴보자.

- 변수선언

```
int sock;
struct sockaddr_in server_addr;

char buf[BUFMAX]; //BUFMAX = 1024
memset(buf, 0x00, BUFMAX);
```

  - `int sock` : socket 함수의 반환값을 저장할 소켓 디스크립터
  - `struct sockaddr_in server_addr` : 소켓에 주소와 포트 번호를 할당하기 위한 구조체
  - `char buf[BUFMAX]` : 전송과 수신이 되는 문자열을 저장할 버퍼

- socket()

```
sock = socket(AF_INET, SOCK_STREAM, 0);
if(sock == -1){
    printf("CLIENT : socket error\n");
    return 1;
}
```

  socket 함수를 이용해 소켓을 생성해준다.

- connect()

```
if (connect(sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1){
    printf("CLIENT : connect error\n");
    return 1;
}
```

- read() / write()

```
memset(buf, 0x00, BUFMAX);
printf("me > ");
fgets(buf, BUFMAX, stdin);
buf[strlen(buf) - 1] = '\0';
if(!strcmp(buf, "QUIT")){
    if(send(sock, buf, BUFMAX, 0) <= 0){
        printf("CLIENT : write error\n");
        return 1;
    }
    printf("Disconnected\n");
    close(sock);
    return 0;
}

if(send(sock, buf, BUFMAX, 0) <= 0){
    printf("CLIENT : write error\n");
```

```
    return 1;
}
```

우선 buf를 초기화 해주고, buf에 문자열을 입력받는다.

fgets를 이용해 입력받으면, 맨 뒤에 개행문자도 저장되므로, 해당 부분만 '\0' 으로 바꿔준다.

입력받은 문자열이 "QUIT"이라면, 소켓으로 write를 이용해 전송하고, 소켓을 닫아준 뒤, 종료한다.

다른 문자열이라면, write를 이용해 소켓으로 전송한다.

```
memset(buf, 0x00, BUFMAX);
if(recv(sock, buf, BUFMAX, 0) <= 0){
    printf("CLIENT : read error\n");
    return 1;
}

if(!strcmp(buf, "QUIT")){
    printf("server : %s\n", buf);
    printf("Disconnected\n");
    close(sock);
    return 0;
}
printf("server : %s\n", buf);
```

해당 부분을 통해 server에서 전송된 데이터를 read로 읽어봐 buf에 저장한다.

만약 read가 비정상적으로 종료되면, -1을 반환하므로 해당 경우에는 소켓을 닫고 종료한다.

읽어온 문자열이 "QUIT"이라면, 출력한 뒤 소켓을 닫고 종료한다.

다른 문자열이라면, 출력해준다.

- close()

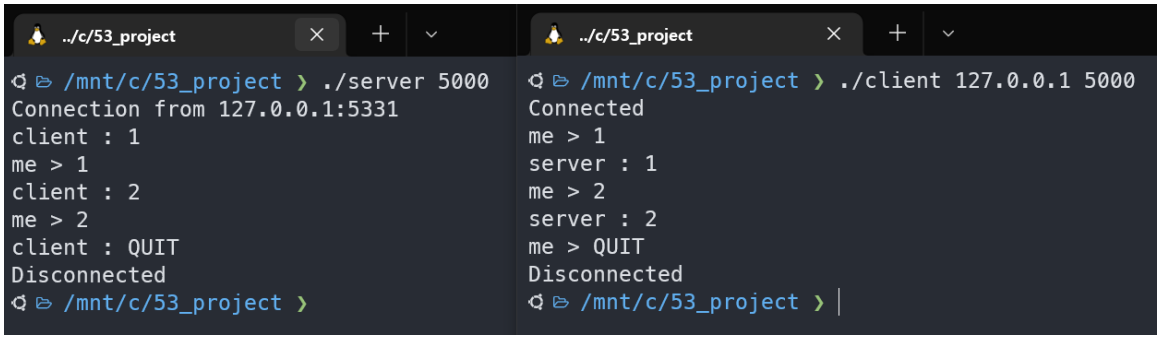
```
close(sock);
return 0;
```

소켓을 닫고, main 함수 종료.

6. 실행 예시

서로 2번씩 문자열을 주고 받은 후 각자가 종료하는 경우에 대한 실행 예시는 아래와 같다.

- client가 QUIT



- server가 QUIT