1. TCP 소켓 통신 : server - client 간 문자 전송 및 응답

server.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <pthread.h>

int main()
{
    int server_sockfd, client_sockfd;
    int server_len, client_len;
    struct sockaddr_in server_address;
    struct sockaddr_in client_address;

    unlink("server_socket");
    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);

    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
    server_address.sin_port = htons(9734);
    server_len = sizeof(server_address);
    bind(server_sockfd, (struct sockaddr *)&server_address, server_len);

    listen(server_sockfd, 5);
    while(1) {
        char ch;

        printf("server waiting\n");

        client_len = sizeof(client_address);
        client_sockfd = accept(server_sockfd, (struct sockaddr
*)&client_address, &client_len);

        read(client_sockfd, &ch, 1);
        ch++;
        write(client_sockfd, &ch, 1);
        close(client_sockfd);
    }
}
```

서버 소켓 생성, 설정

```
int server_sockfd, client_sockfd;
int server_len, client_len;
struct sockaddr_in server_address;
struct sockaddr_in client_address;

unlink("server_socket");
server_sockfd = socket(AF_INET, SOCK_STREAM, 0);

server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = htonl(INADDR_ANY);
server_address.sin_port = htons(9734);
server_len = sizeof(server_address);
bind(server_sockfd, (struct sockaddr *)&server_address, server_len);
```

클라이언트의 연결을 받기 위한 소켓인 **server_sockfd**과, 클라이언트와의 통신을 위한 소켓 **client_sockfd**을 선언한다. **server_address, client_address**에 서버 주소 정보를 담는 구조체를 선언한다. **server_address**의 주소 패밀리는 **AF_INET**(IPv4)로 설정하고, 서버의 포트 번호를 **9734**로 설정한다. 또한 **htonl**로 호스트 시스템이 빅 엔디안, 리틀 엔디안에 상관 없이 네트워크 표준의 빅 엔디안 순서로 변환시킨다. **INADDR_ANY**를 사용하면 서버는 모든 네트워크 인터페이스에 바인딩되므로, 서버는 시스템에 있는 모든 네크워크 인터페이스에서 들어오는 연결을 수락할 수 있게 된다.

연결 대기 상태 설정

```
listen(server_sockfd, 5);
```

listen 함수를 사용하여 클라이언트의 연결을 대기한다. 최대 5개의 연결을 허용한다.

무한 루프로 클라이언트 연결 대기

```
while(1) {
    char ch;

    printf("server waiting\n");

    client_len = sizeof(client_address);
    client_sockfd = accept(server_sockfd, (struct sockaddr *)&client_address,
&client_len);
```

루프 내에서 클라이언트의 연결을 대기하고, 클라이언트가 연결되면 수락한다.

데이터 송/수신, 클라이언트에 응답

```
  read(client_sockfd, &ch, 1);
    ch++;
    write(client_sockfd, &ch, 1);
    close(client_sockfd);
}
```

read 함수를 사용해 클라이언트로부터 값을 읽어오고, 읽어온 문자의 값을 증가시켜 **write** 함수로 전송한다. 이후 **close** 함수로 클라이언트 소켓을 닫는다.

client.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <pthread.h>

int main()
{
    int sockfd;
    int len;
    struct sockaddr_in address;
    int result;
    char ch = 'A';

    sockfd = socket(AF_INET, SOCK_STREAM, 0);

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = inet_addr("127.0.0.1");
    address.sin_port=htons(9734);
    len = sizeof(address);

    result = connect(sockfd, (struct sockaddr *)&address,
len);
    if (result == -1){
        perror("oops: client3");
        exit(1);
    }

    write(sockfd, &ch, 1);
    read(sockfd, &ch, 1);
    printf("char from server = %c\n", ch);
    close(sockfd);
    exit(0);
}
```

소켓 생성

```
int sockfd;
sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

인터넷 주소 패밀리, 스트림 소켓, 프로토콜을 지정하여 소켓을 생성한다.

AF_INET은 IPv4를 사용할 때 사용하는 네트워크 주소 체계이다. 소켓을 만들 때는 주소의 패밀리를 지정해야하며, 해당 유형의 주소만 소켓과 함께 사용할 수 있다.

SOCK_STREAM은 TCP 프로토콜로 통신할 때 사용한다. UDP 프로토콜로 통신하고자 한다면 **SOCK_DGRAM**을 사용해야 한다.

서버 주소 설정

```
struct sockaddr_in address;
address.sin_family = AF_INET;
address.sin_addr.s_addr = inet_addr("127.0.0.1");
address.sin_port=htons(9734);
len = sizeof(address);
```

IPv4 주소 패밀리 **AF_INET**을 사용하고, 서버 IP 주소 '**127.0.0.1**'의 포트 **9734**번으로 서버 주소를 설정한다.

서버 연결

```
result = connect(sockfd, (struct sockaddr *)&address, len);
```

connect 함수를 사용하여 서버에 연결을 시도한다. 연결에 실패하면 오류를 출력하고 프로그램을 종료한다.

데이터 송/수신

```
write(sockfd, &ch, 1);
read(sockfd, &ch, 1);
```

클라이언트가 **write** 함수를 사용하여 서버에 문자를 보내고, **read** 함수를 사용하여 서버로부터 문자를 읽어온다.

결과 출력 후 소켓 종료

```
printf("char from server = %c\n", ch);
close(sockfd);
exit(0);
```

서버로부터 받은 결과를 출력하고, **close** 함수로 소켓을 닫고 프로그램을 종료한다.

동작 확인

```
jo@jo-960XFG:~/Downloads/hw$ ./server
server waiting
server waiting
server waiting
```

```
jo@jo-960XFG:~/Downloads/hw$ ./client
char from server = B
jo@jo-960XFG:~/Downloads/hw$ ./client
char from server = B
```

client에서 'A'를 전송하면 server에서 받은 값에 +1을 하고 client 쪽으로 값을 전송한다.
따라서 client가 응답 받는 값은 'B'가 된다.

2. TCP 소켓 통신 : server - n clients 채팅

server.c

초기 작업

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <pthread.h>
#include <time.h>

#define CLNT_MAX 10
#define BUFFSIZE 200

int g_clnt_socks[CLNT_MAX];
int g_clnt_count =0;

pthread_mutex_t g_mutex;
```

헤더 파일을 다음과 같이 추가하고, 전역 변수와 상수를 선언했다. 또한 전역 변수에 스레드 다중 접근 시 임계영역의 문제가 발생할 여지가 있기 때문에 뮤텍스를 생성했다.

send_all_clnt()

```
void send_all_clnt(char * msg,int my_sock){


        pthread_mutex_lock(&g_mutex);
    for(int i = 0 ; i <g_clnt_count ; i++){
        if(g_clnt_socks[i] != my_sock){
            printf("send msg : %s",msg);
            write(g_clnt_socks[i],msg,strlen(msg)+1);
        }
    }

        pthread_mutex_unlock(&g_mutex);

}
```

server에서 client로 메시지를 전송하는 함수이다. 함수 호출 시 뮤텍스 락을 획득한다. 이후 메시지를 server에게 전송한 client를 제외한 모든 client에게 전달받은 메시지를 전송한다. 이후 뮤텍스를 반환한다.

clnt_connection()

```
void * clnt_connection(void * arg){

    int clnt_sock = (int)arg;
    int str_len=0;

    char msg[BUFFSIZE];
    int i;

    while(1){
    str_len = read(clnt_sock,msg,sizeof(msg));
        if(str_len == -1){
            printf("clnt[%d] close\n",clnt_sock);
            break;
        }
        send_all_clnt(msg,clnt_sock);
        printf("%s\n",msg);
    }


    pthread_mutex_lock(&g_mutex);

        for(i=0; i<g_clnt_count; i++){
                if(clnt_sock == g_clnt_socks[i]){
                        for(;i<g_clnt_count-1;i++)
                                g_clnt_socks[i]=g_clnt_socks[i+1];
                        break;
                }
        }

    pthread_mutex_lock(&g_mutex);
    close(clnt_sock);

    pthread_exit(0);
    return NULL;
}
```

client가 server에 접속했을 때 스레드를 생성 후 실행되는 함수이다. client로부터 받은 메시지를 **read** 함수로 읽어서 send_all_clnt 함수를 호출해 모든 client에 전송한다. 또한 읽어온 메시지를 출력한다. 이후 연결이 종료되면 뮤텍스를 획득해 전역 변수 소켓 배열에서 이를 제거한 뒤 뮤텍스를 반환하고, 스레드를 종료한다.

main()

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

    int serv_sock;
    int clnt_sock;

    pthread_t t_thread;

    struct sockaddr_in clnt_addr;
        int clnt_addr_size;

    struct sockaddr_in serv_addr;


    pthread_mutex_init(&g_mutex,NULL);

    serv_sock = socket(PF_INET,SOCK_STREAM,0);

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port=htons(8934);

    if(bind(serv_sock,(struct sockaddr *)&serv_addr,sizeof(serv_addr)) == -1){
        printf("Error : Bind\n");
    }


    if(listen(serv_sock,5) == -1){
        printf("Error : Listen\n");
    }

    char buff[200];
    int recv_len =0;
    while(1){

        clnt_addr_size=sizeof(clnt_addr);
        clnt_sock = accept(serv_sock,(struct sockaddr
*)&clnt_addr,&clnt_addr_size);

        pthread_mutex_lock(&g_mutex);
        g_clnt_socks[g_clnt_count++] = clnt_sock;
        pthread_mutex_unlock(&g_mutex);

        pthread_create(&t_thread,NULL,clnt_connection,(void *)clnt_sock);


    }
}
```

기본적인 설정(client 소켓 설정, 연결 대기 상태 설정 등)을 하고 **bind**를 사용해 server 소켓에 주소를 할당하여 client 연결을 수락하는 작업을 한다. 이후 무한 루프를 돌며 client 연결을 기다리고, **accept** 함수에서 client의 연결을 수락하여 client 소켓을 반환한다. 이후 뮤텍스를 획득하여 전역 변수인 client 소켓 배열에 반환된 client 소켓을 추가하고, 스레드를 생성해 **clnt_connection** 함수를 전달하여 client와 통신을 한다.

client.c

초기 작업

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <pthread.h>



#define BUFFSIZE 100
#define NAMESIZE 20

int StoHEX(char fi, char sc);
void error_handling(char *msg);
void * send_message(void * arg);
void * recv_message(void * arg);


char message[BUFFSIZE];
```

헤더 파일을 다음과 같이 추가하고, 전역 변수와 상수를 선언했다.

rcv()

```
void * rcv(void * arg){

    printf("rcv thread created\n");
    int sock = (int)arg;
    char buff[500];
    int len;
    while(1){
        len = read(sock,buff,sizeof(buff));

        if(len == -1){
            printf("sock close\n");
            break;
        }
        printf("%s",buff);
    }

    pthread_exit(0);
    return 0;
}
```

client의 소켓을 받아와서 **read** 함수로 메시지를 읽어온다. 수신한 메시지를 출력하고 스레드를 종료한다.

main()

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

        int sock;
        struct sockaddr_in serv_addr;
        pthread_t rcv_thread;
        void* thread_result;

    char id[100];
    printf("argc : %d\n",argc);
    if(argc < 2){
        printf("you have to enter ID\n");
        return 0;
    }
    strcpy(id,argv[1]);
    printf("id : %s\n",id);

        sock=socket(PF_INET, SOCK_STREAM, 0);
        if(sock==-1){
        printf("socket error");
    }else{

        printf("socket ok\n");
    }

        memset(&serv_addr, 0,sizeof(serv_addr));
        serv_addr.sin_family=AF_INET;
        serv_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
        serv_addr.sin_port=htons(7989);

        if(connect(sock, (struct sockaddr *)&serv_addr,sizeof(serv_addr)) ==
-1){
    printf("connect error\n");
    }else{

    printf("connection success\n");
    }

    pthread_create(&rcv_thread,NULL,rcv,(void *)sock);

     char chat[100];
     char msg[1000];


    printf("while before\n");
    while(1){
        printf("채팅 입력 : ");
        gets(chat);
        sprintf(msg,"[%s] : %s\n",id,chat);
        printf("send : %s",msg);
        write(sock,msg,strlen(msg)+1);
        sleep(1);
    }

    printf("while end\n");
        close(sock);
        return 0;


}
```

client.o 실행 시 사용할 id를 입력받는다. client 소켓을 생성하고 서버 주소 설정, 서버 연결 과정을 거쳐 **connect** 함수로 server 소켓에 이를 연결시킨다. 스레드를 생성하여 **rcv** 함수를 호출한다.. 이후 무한 루프를 돌며 채팅을 입력받고, **write** 함수로 server에 메시지를 전송한다. 연결이 종료되면 소켓을 반환한다.

동작 확인