[CYDF321-TP] 2021350225 - 오정민

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 5개의 TCP 클라이언트만 accept한다.
   - TCP 연결이 된 이후, 클라이언트의 IP 주소와 포트 번호가 화면에 표준 출력으로 출력된다.

     (예 : “Connection from 163.152.162.144:12345”)
   - connection 이후, 서버는 클라이언트의 닉네임을 받는다.
   - 다른 클라이언트가 접속하면, 서버는 클라이언트의 닉네임을 사용하여 연결된 다른 클라이언트에 새 연결을 통지하고 서버의 화면에도 출력한다.

     (예: “User2 is connected”)
   - 클라이언트에서 받은 메시지는 화면에 출력되어야 한다.

     메시지가 출력될 때, 해당 메시지를 보낸 사용자의 이름도 같이 출력된다.
   - 서버는 input을 받지 않고 메시지를 받기만 한다.

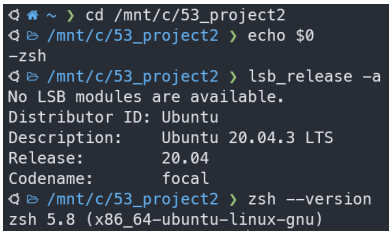
     또한, 받은 메시지를 다른 클라이언트에게 모두 전달한다.
   - 서버는 multi-threading을 통해 많은 클라이언트들과 동시에 소통한다.

     따라서, thread들이 공유하는 변수들은 임계구역을 설정해 동시에 접근하지 못하도록 해야한다.
   - 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 -lpthread
  gcc -o client client.c -lpthread
clear :
  rm client server
```

위와 같이 Makefile을 만들고 make 명령어를 이용해 컴파일 하였다.

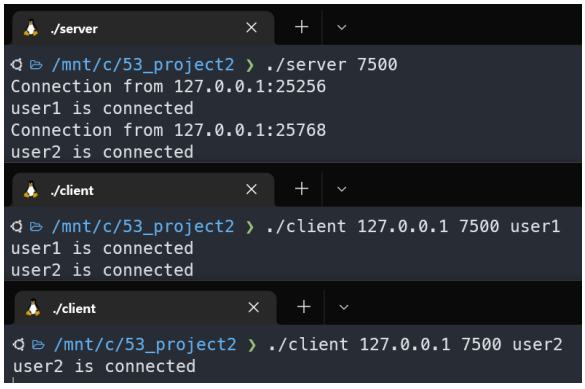
```
[실행 방법]
1. server
  ./server <port number>
2. client
  ./client <server ip> <port number> <nickname>
```

위와 같이 적절한 인자를 이용해 실행한다.

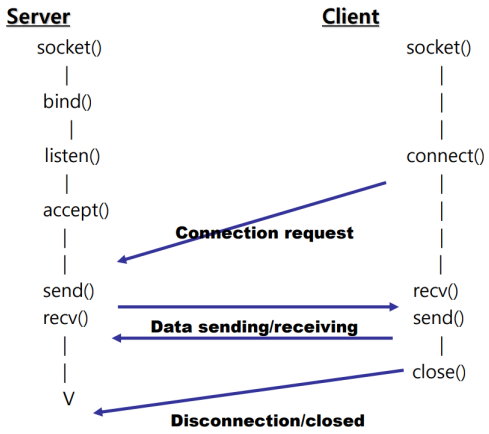
가상 머신을 이용해 두 환경을 이용하여 실행하는 경우 server를 실행하는 환경의 ip 주소를 client 실행시에 인자로 넘겨주면 된다.

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

아래와 같이 여러 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 넣어준다.

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 넣어준다.

<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_in구조체의 주소

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

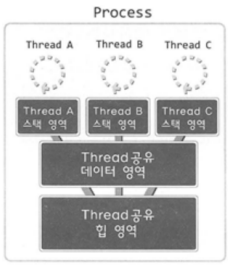
6. Multi-threaded application

thread는 멀티프로세스의 특징을 유지하면서 단점을 보완하기 위해 등장했다.

thread는 하나의 프로세스 내부에서 별도의 실행흐름을 유지하기 위해 각 thread가 스택 영역만 독립적으로 유지한다.

따라서, 프로세스의 data 영역과 heap 영역은 thread들 모두에 공유된다.

메모리를 그림으로 표현하면 아래와 같다.



이러한 thread를 생성하고 관리하기 위해 우리는 c언어의 `pthread.h` 헤더파일의 내부 함수들과 자료형들을 이용할 것이다.

- pthread_create - thread 생성

```
int pthread_create(pthread_t *restrict thread, const pthread_attr_t* restrict attr, void*(*start_routine)(void*), void* restrict a
```

- 반환값

  성공 : 0 / 실패 : 0 이외의 값

- 인자

  1. thread : 생성되는 thread의 id를 저장하기 위한 주소값.
  2. attr : 해당 thread의 main함수 역할을 할 함수포인터.
  3. arg : attr 함수가 호출될 때 전달할 인자를 담은 주소 값.

thread가 생성되어 실행될 때 메인 프로세스의 흐름을 제어해야한다.

- pthread_join - 프로세스 제어

```
int pthread_join(pthread_t thread, void** status);
```

- 반환값

  성공 : 0 / 실패 : 0 이외의 값

- 인자

1. thread : 진행되는 thread의 id
2. status : thread의 함수가 반환하는 값이 저장될 포인터 변수의 주소.

join 함수를 이용하면, thread가 종료될 때까지 함수를 반환하지 않는다. 따라서, process가 대기상태로 유지된다.

여러개의 thread가 공유된 메모리의 변수중 특정 변수에 동시에 접근하는것을 막기위해 thread를 동기화 시켜줘야한다.

우리는 thread의 동시접근을 막기위해 mutex라는 자물쇠와 같은 함수를 이용할 것이다.

- mutex

  우선 mutex를 이용하기 위해서는 아래와 같이 mutex형 변수가 선언되어야 한다.

  ```
  pthread_mutex_t mutex;
  ```

- mutex 생성 / 소멸

  ```
  int pthread_mutex_init(pthread_mutex_t* mutex, const pthread_mutexattr_t* attr);

  int pthread_mutex_destroy(pthread_mutex_t* mutex);
  ```

  mutex의 생성과 소멸에 관한 함수는 위와 같다.

  - 반환값

    성공 : 0 / 실패 : 0 이외의 값

  - 인자

    1. mutex : 뮤텍스 생성시에 뮤텍스의 참조 값 저장을 위한 주소값, 소멸시에 소멸하고자하는 뮤텍스의 참조값을 저장하고있는 주소값.
    2. attr : 생성하는 뮤텍스의 특성정보를 담고 있는 변수의 주소 값 전달, 별도의 특성을 지정하지 않을 경우 NULL.

- mutex lock / unlock

  ```
  int pthread_mutex_lock(pthread_mutex_t *mutex);
  int pthread_mutex_unlock(pthread_mutex_t *mutex);
  ```

  - 반환값

    성공 : 0 / 실패 : 0 이외의 값

  공유된 메모리를 보호하기 위해 `pthread_mutex_lock` 함수를 이용해 임계 영역을 만들고, 해당 임계 영역에는 하나의 thread만 진입가능하도록 한다.

  임계 영역의 이용이 끝나면 `pthread_mutex_unlock` 함수를 이용해 임계 영역을 해제한다.

thread의 소멸에는 2가지 case가 있다.

1. pthread_join

   join 함수는 thread의 종료를 대기할 뿐 아니라, thread의 소멸도 유도한다.

2. pthread_detach

   thread를 종료하고, 소멸시킴.

- pthread_detach

  ```
  int pthread_detach(pthread_t thread);
  ```

  - 반환값

    성공 : 0 / 실패 : 0 이외의 값

  - 인자

    thread : 종료와 동시에 소멸시킬 thread의 id.

5. Code detail

두 소스코드에서 사용된 함수인 socket, bind, listen, connect, accept, read, write 는 모두 비정상적인 종료시에 -1을 반환한다. 해당 경우에는 에러 처리를 해주었다.

1. server.c

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

#define MAXBUF 1024
#define MAXCLIENT 5
#define NAMESIZE 20

void *t_main(void *arg);
void send_msg(char* msg, int len, int client_socket);

pthread_mutex_t mutex;
int client[MAXCLIENT];
int cnt = 0;
int check = 0;

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

    if(argc != 2){
        printf("Usage: %s [port]\n", argv[0]);
        return 1;
    }

    pthread_t tid;

    int server_socket, client_socket;

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

    pthread_mutex_init(&mutex, NULL);

    server_socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(server_socket == -1){
        printf("SERVER : 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 = 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;
    }

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

        if(cnt == MAXCLIENT){
            if(check == 0){
                printf("client is full\n");
                check = 1;
            }
            close(client_socket);
            continue;
        }

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

        pthread_mutex_lock(&mutex);
        client[cnt++] = client_socket;
        pthread_mutex_unlock(&mutex);
```

```
        pthread_create(&tid, NULL, t_main, (void *)&client_socket);
        pthread_detach(tid);
    }

    close(server_socket);
    return 0;
}

void *t_main(void *arg){
    int i, len = 0, name_len = 0,client_socket = *((int *)arg);

    char name[NAMESIZE];
    char tmp[MAXBUF];
    char buf[MAXBUF + name_len];

    if((name_len = recv(client_socket, name, NAMESIZE, 0)) == -1){
        printf("SERVER : name recv error\n");
        return NULL;
    }

    memset(buf, 0x0, MAXBUF);
    sprintf(buf, "%s is connected\n", name);
    printf("%s is connected\n", name);
    send_msg(buf, MAXBUF, client_socket);

    while(1){
        if((len = recv(client_socket, tmp, MAXBUF, 0)) == -1){
            printf("SERVER : recv error\n");
            continue;
        }
        if(!strcmp(tmp, "QUIT"))
            break;
        sprintf(buf ,"%s: %s\n", name, tmp);
        printf("%s", buf);
        send_msg(buf, len, client_socket);
    }

    memset(buf, 0x0, MAXBUF);
    sprintf(buf, "%s is disconnected\n", name);
    printf("%s is disconnected\n", name);
    send(client_socket, buf, MAXBUF, 0);
    send_msg(buf, MAXBUF, client_socket);

    pthread_mutex_lock(&mutex);
    for(i = 0; i < cnt; i++){
        if(client_socket == client[i]){
            while(i++ < cnt-1)
                client[i] = client[i+1];
            break;
        }
    }

    check = 0;
    cnt--;
    pthread_mutex_unlock(&mutex);
    close(client_socket);
}

void send_msg(char* msg, int len, int client_socket){
    int i;
    pthread_mutex_lock(&mutex);
    for(i = 0; i < cnt; i++)
        if(client[i] == client_socket)
            continue;
        else
            send(client[i], msg, len, 0);
    pthread_mu
```

서버의 전체 소스코드는 위와 같다.

함수로 나눠서 코드의 detail을 살펴보자.

- 전역변수

```
pthread_mutex_t mutex;
int client[MAXCLIENT];
int cnt = 0;
int check = 0;
```

  - `pthread_mutex_t mutex` : 임계영역 보호를 위한 mutex 변수

- `int client[MAXCLIENT]` : client의 fd를 저장할 배열
- `int cnt = 0` : client의 수를 세줌.

- main
  - 변수 선언

```
pthread_t tid;

int server_socket, client_socket;

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

pthread_mutex_init(&mutex, NULL);
```

    - `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
    - `pthread_t tid;` : thread의 id를 저장해줄 변수
    - `pthread_mutex_init(&mutex, NULL);` : mutex 생성
  - 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.
  - while 문
    - 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;
}
if(cnt == MAXCLIENT){
```

```
    if(check == 0){
        printf("client is full\n");
        check = 1;
    }
    close(client_socket);
    continue;
}
pthread_mutex_lock(&mutex);
client[cnt++] = client_socket;
pthread_mutex_unlock(&mutex);
```

- listen 함수를 통해 확인한 접속 요청을 accept 함수로 승인한다.

  accept 함수는 소켓을 생성해준다. 해당 소켓을 client_socket이라고 하자.

- 연결된 client의 수가 MAXCLINET값과 같다면 새로 연결된 client_socket을 닫아준다.
- 전역변수인 client 배열에 접근하기 위해 mutex로 lock 걸고 client_socket 저장하고 unlock

  - thread

```
pthread_create(&tid, NULL, t_main, (void *)&client_socket);
pthread_detach(tid);
```

- `pthread_create` 를 이용해 tid 에 thread id 저장, t_main 호출.
- `pthread_detach` thread 종료시 소멸.

- t_main
  - 변수

```
int i, len = 0, client_socket = *((int *)arg);

char buf[MAXBUF];
```

매개변수로 오는 client_socket 받아줌.

send, recv 할 buf 선언.

  - recv

```
if((name_len = recv(client_socket, name, NAMESIZE, 0)) == -1){
    printf("SERVER : name recv error\n");
    return NULL;
}

while(1){
        if((len = recv(client_socket, tmp, MAXBUF, 0)) == -1){
            printf("SERVER : recv error\n");
            continue;
        }
        if(!strcmp(tmp, "QUIT"))
            break;
        sprintf(buf ,"%s: %s\n", name, tmp);
        printf("%s", buf);
        send_msg(buf, len, client_socket);
    }
```

recv로 이름 먼저 받아줌.

recv 계속 올 때까지 받아줌. 받은 문자열이 QUIT이면 break, 아니면 send_msg 함수를 통해 이름 붙여서 문자열 보내주고, 출력해줌.

  - 종료된 client 삭제

```
pthread_mutex_lock(&mutex);
for(i = 0; i < cnt; i++){
    if(client_socket == client[i]){
        while(i++ < cnt-1)
            client[i] = client[i+1];
        break;
    }
}
```

```
check = 0;
cnt--;
pthread_mutex_unlock(&mutex);
close(client_socket);
```

전역변수 접근하므로, mutex_lock

QUIT 문자열 받아서 종료되는 client_socket 배열에서 지워주고, cnt 하나 감소.

mutex_unlock 해줌.

client_socket 닫아줌.

- connect, disconnect 문자열 보내주기.

```
memset(buf, 0x0, MAXBUF);
sprintf(buf, "%s is connected\n", name);
printf("%s is connected\n", name);
send_msg(buf, MAXBUF, client_socket);

memset(buf, 0x0, MAXBUF);
sprintf(buf, "%s is disconnected\n", name);
printf("%s is disconnected\n", name);
send(client_socket, buf, MAXBUF, 0);
send_msg(buf, MAXBUF, client_socket);
```

- send_msg 함수.

```
void send_msg(char* msg, int len, int client_socket){
    int i;
    pthread_mutex_lock(&mutex);
    for(i = 0; i < cnt; i++)
        if(client[i] == client_socket)
            continue;
        else
            send(client[i], msg, len, 0);
    pthread_mutex_unlock(&mutex);
}
```

인자로 받은 msg 를 보낸 client를 제외한 나머지 모든 client에게 send 해줌.

client 배열에 접근하기전에 mutex_lock, 접근 끝나고 mutex_unlock

2. client.c

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

#define MAXBUF     1024
#define NAMESIZE   20

pthread_mutex_t mutex;
char name[NAMESIZE] = {0, };
int name_len;

void *send_msg(void *arg);
void *recv_msg(void *arg);

int main(int argc, char **argv){
    pthread_mutex_init(&mutex, NULL);
    int sock;
    struct sockaddr_in server_addr;
    pthread_t send_t, recv_t;
    void *thread_return;
    sprintf(name, "%s", argv[3]);
    name_len = strlen(name);

    //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;
    }

    if(send(sock, name, name_len, 0) == -1){
        printf("CLIENT : name send error\n");
        return 1;
    }
    printf("%s is connected\n",name);
    pthread_create(&send_t, NULL, send_msg, (void*)&sock);
    pthread_create(&recv_t, NULL, recv_msg, (void*)&sock);
    pthread_join(send_t, &thread_return);
    pthread_join(send_t, &thread_return);

    close(sock);
    return 0;
}

void *send_msg(void* arg){
    int sock = *((int*)arg);
    char buf[MAXBUF];

    while(1){
        scanf("%s",buf);

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

        if(!strcmp(buf, "QUIT")){
            printf("%s is disconnected", name);
            close(sock);
            return NULL;
        }
    }
}

void *recv_msg(void* arg){
    int sock = *((int*)arg);
    char buf[MAXBUF];
    int len;
    char str[100];
    sprintf(str, "%s is disconnnected\n",name);

    while(1){
        if(len = recv(sock, buf, MAXBUF, 0) == -1){
            printf("CLIENT : recv error\n");
        }
        if(!strcmp(buf, str)){
            close(sock);
            return NULL;
        }
        printf("%s", buf);
    }
}
```

client의 전체 소스코드는 위와 같다.

함수별로 나눠서 코드를 살펴보자.

- main 함수
  - 변수선언

```
pthread_mutex_init(&mutex, NULL);
int sock;
struct sockaddr_in server_addr;
pthread_t send_t, recv_t;
void *thread_return;
```

    - `int sock` : socket 함수의 반환값을 저장할 소켓 디스크립터

- `struct sockaddr_in server_addr` : 소켓에 주소와 포트 번호를 할당하기 위한 구조체
- `pthread_t send_t, recv_t` : send와 recv를 해줄 thread의 id 변수
- `void *thread_return;` : thread의 함수 반환을 저장해줄 포인터 변수

- 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;
}
if(send(sock, name, name_len, 0) == -1){
    printf("CLIENT : name send error\n");
    return 1;
}
printf("%s is connected\n",name);
```

connect 함수를 이용해 listen 상태인 서버에 연결해줌.

연결 되면 바로 이름 보내고, connected 문자열 출력.

- thread

```
pthread_create(&send_t, NULL, send_msg, (void*)&sock);
pthread_create(&recv_t, NULL, recv_msg, (void*)&sock);
pthread_join(send_t, &thread_return);
pthread_join(send_t, &thread_return);
```

pthread_create로 send_msg 함수와 recv_msg 함수가 main인 thread를 각각 생성.

각 thread가 끝날 때 까지 대기하도록 pthread_join 실행.

- send_msg

```
void *send_msg(void* arg){
    int sock = *((int*)arg);
    char buf[MAXBUF];

    while(1){
        scanf("%s",buf);

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

        if(!strcmp(buf, "QUIT")){
            printf("%s is disconnected", name);
            close(sock);
            return NULL;
        }

    }
}
```

socket fp를 인자로 받아와 sock에 저장.

send해줄 배열 buf 선언.

while문을 통해 buf 에 입력 받고 send 해준 뒤, QUIT 이면 sock close 해주고 함수 종료.

- recv_msg

```
void *recv_msg(void* arg){
    int sock = *((int*)arg);
    char buf[MAXBUF];
    int len;
    char str[100];
    sprintf(str, "%s is disconnnected\n",name);

    while(1){
        if(len = recv(sock, buf, MAXBUF, 0) == -1){
            printf("CLIENT : recv error\n");
        }
        if(!strcmp(buf, str)){
            close(sock);
            return NULL;
        }
        printf("%s", buf);
    }
}
```

socket fp를 인자로 받아와 sock에 저장.

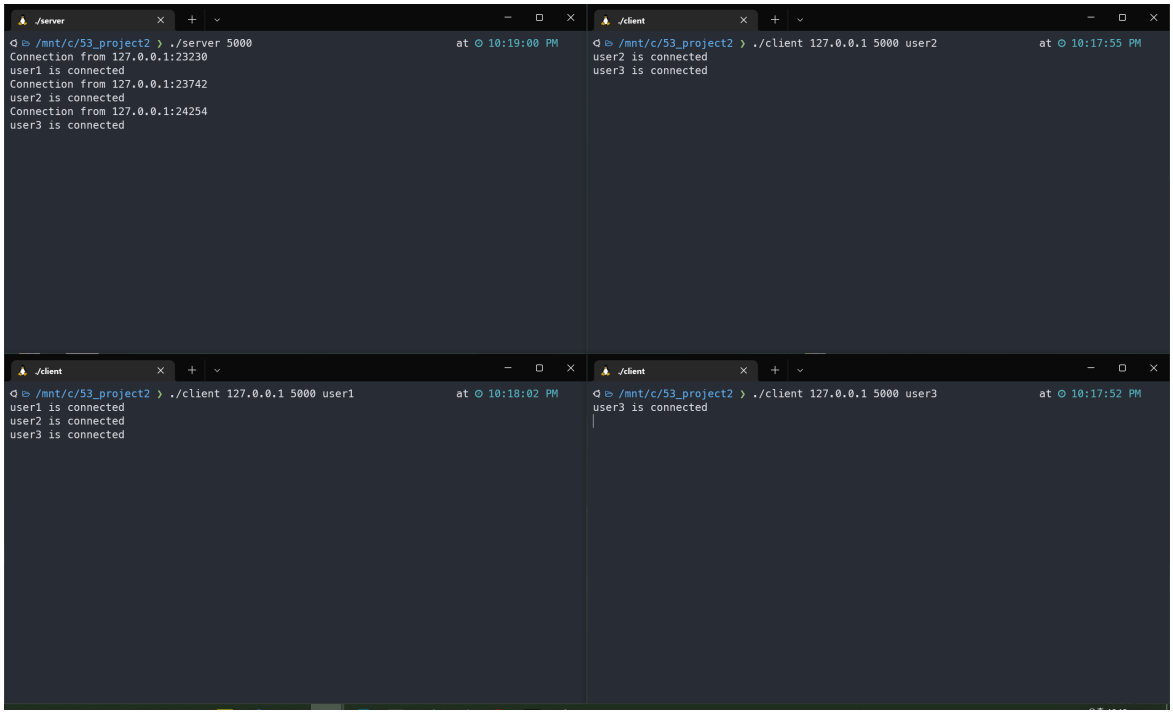
recv해줄 배열 buf 선언

while문을 통해 recv 해준 문자열 출력.

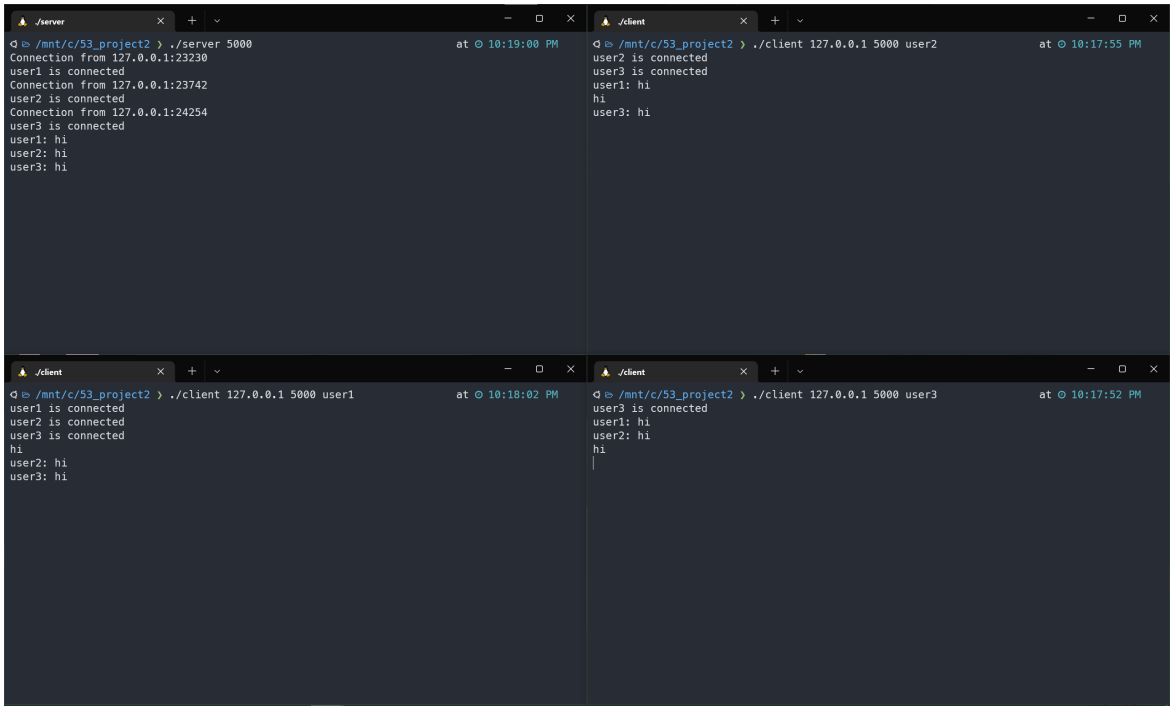
send에서 QUIT을 보내 <name> is disconnected 가 오면 소켓 닫고 리턴해줌.

6. 실행 예시

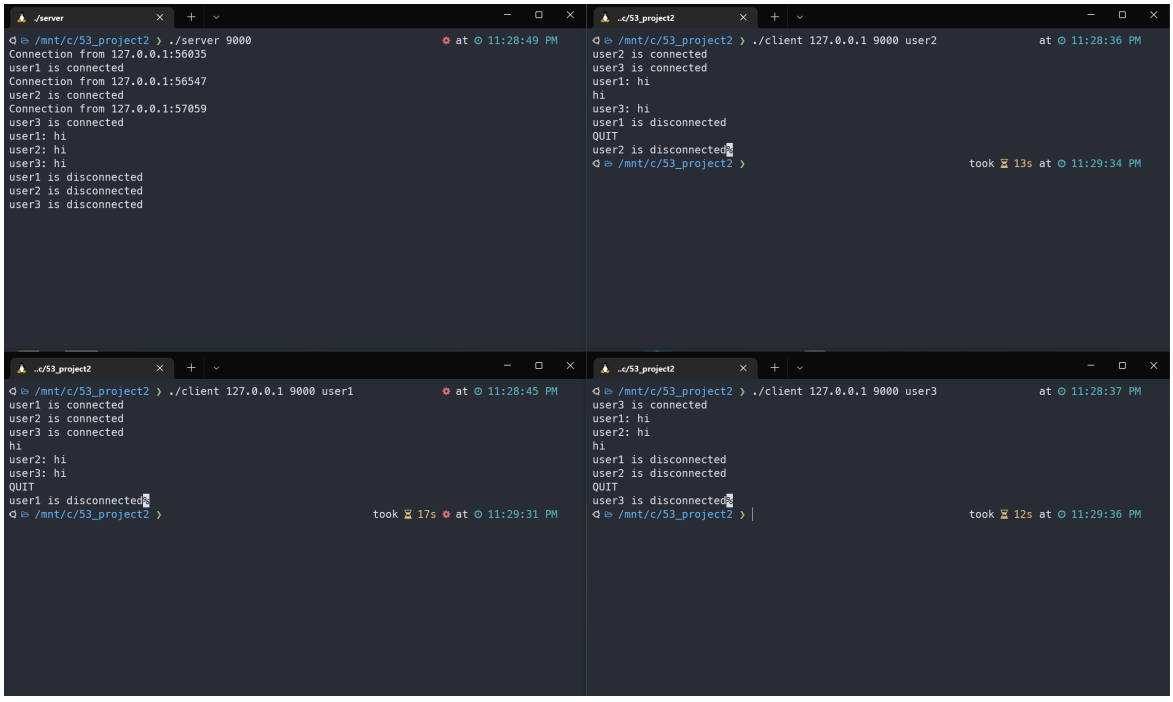
- 3명의 클라이언트 순서대로 접속



- 각 클라이언트가 메시지 보냄.


./server
/mnt/c/53_project2 ❯ ./server 5000
at 10:19:00 PM
Connection from 127.0.0.1:23230
user1 is connected
Connection from 127.0.0.1:23742
user2 is connected
Connection from 127.0.0.1:24254
user3 is connected
user1: hi
user2: hi
user3: hi
./client
/mnt/c/53_project2 ❯ ./client 127.0.0.1 5000 user2
at 10:17:55 PM
user2 is connected
user3 is connected
user1: hi
hi
user3: hi
./client
/mnt/c/53_project2 ❯ ./client 127.0.0.1 5000 user1
at 10:18:02 PM
user1 is connected
user2 is connected
user3 is connected
hi
user2: hi
user3: hi
./client
/mnt/c/53_project2 ❯ ./client 127.0.0.1 5000 user3
at 10:17:52 PM
user3 is connected
user1: hi
user2: hi
hi

- QUIT


./server
/mnt/c/53_project2 ❯ ./server 9000
at 11:28:49 PM
Connection from 127.0.0.1:56035
user1 is connected
Connection from 127.0.0.1:56547
user2 is connected
Connection from 127.0.0.1:57059
user3 is connected
user1: hi
user2: hi
user3: hi
user1 is disconnected
user2 is disconnected
user3 is disconnected
../53_project2
/mnt/c/53_project2 ❯ ./client 127.0.0.1 9000 user2
at 11:28:36 PM
user2 is connected
user3 is connected
user1: hi
hi
user3: hi
user1 is disconnected
QUIT
user2 is disconnected
/mnt/c/53_project2 ❯
took 13s at 11:29:34 PM
../53_project2
/mnt/c/53_project2 ❯ ./client 127.0.0.1 9000 user1
at 11:28:45 PM
user1 is connected
user2 is connected
user3 is connected
hi
user2: hi
user3: hi
QUIT
user1 is disconnected
/mnt/c/53_project2 ❯
took 17s at 11:29:31 PM
../53_project2
/mnt/c/53_project2 ❯ ./client 127.0.0.1 9000 user3
at 11:28:37 PM
user3 is connected
user1: hi
user2: hi
hi
user1 is disconnected
user2 is disconnected
QUIT
user3 is disconnected
/mnt/c/53_project2 ❯
took 12s at 11:29:36 PM