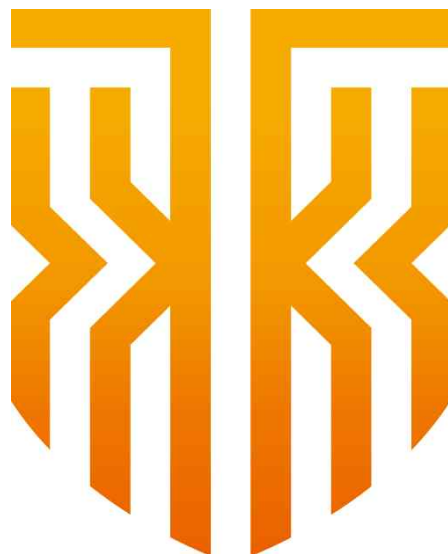


---

# 과제 3

## TCP 기반 서버/클라이언트1

---



# 목차

<b>1. 4장 정리</b>	3
가. TCP / IP 프로토콜 스택	3
나. TCP 소켓과 UDP 소켓의 스택 FLOW	3
다. LINK & IP계층	3
라. TCP/UDP 계층	4
마. APPLICATION 계층	4
바. 소켓을 이용한 TCP 통신	5
사. 반복적인 TCP 통신을 위한 Flow diagram	6
아. TCP 서버의 기본적인 함수호출 순서	6
자. 연결요청 대기 상태로의 진입 – listen( ) 메서드, 서버	7
차. 클라이언트의 연결요청 수락 – accept( ) 메서드, 서버	8
카. TCP 클라이언트의 기본적인 함수호출 순서 – connect( )	9
타. TCP 기반 서버, 클라이언트의 함수호출 관계	9
파. listen( ) 메서드 호출 이후	9
하. Iterative 서버의 구현	10
거. 에코 클라이언트의 문제점	10
<b>2. 프로그램 작성</b>	11
가. Hello world 프로그램 작성 ( 서버, 클라이언트 )	11
나. TCP Iterative echo 프로그램 작성 ( 서버, 클라이언트 )	15
다. 2.가 & 2.나 비교하기	20

# 1. 4장 정리

## 가. TCP / IP 프로토콜 스택

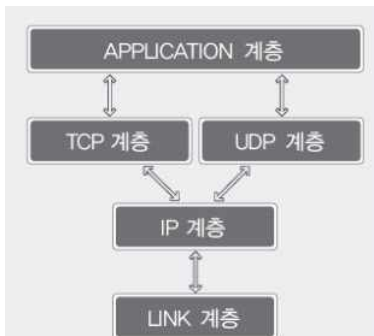
### 1) 인터넷 기반 데이터 송수신을 목적으로 설계된 스택

#### 가) 목적

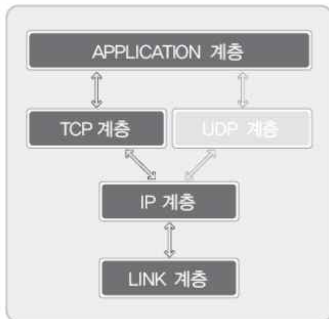
(1) "인터넷 기반의 효율적인 데이터 전송"

#### 나) 방식

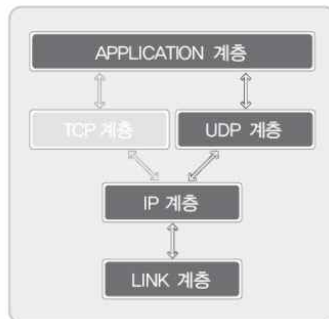
- (1) 계층화된 TCP/IP 프로토콜을 통해 구현
- 2) 큰 문제를 작게 나눠서 계층화한 결과
- 3) 데이터 송수신의 과정을 네 개의 영역으로 계층화 한 결과
- 4) 각 스택별 영역을 전문화하고 표준화함
- 5) 7계층으로 세분화되며, 4계층으로도 표현함



## 나. TCP 소켓과 UDP 소켓의 스택 FLOW



TCP 소켓의 스택 FLOW



UDP 소켓의 스택 FLOW

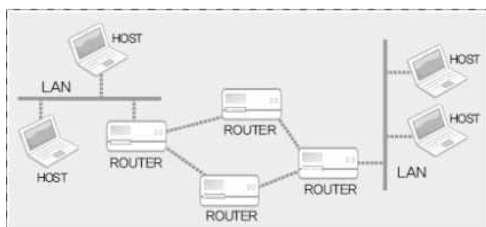
- 1) 각각의 계층을 담당하는 것은 OS, NIC와 같은 sw, hw

## 다. LINK & IP계층

### 1) LINK 계층

#### 가) 기능 및 역할

- (1) 물리적 영역의 표준화 결과
- (2) LAN, WAN, MAN과 같은 물리적인 네트워크 표준 관련 프로토콜이 정의된 영역
- (3) 아래의 그림과 같은 물리적인 연결의 표준이 된다.



- (4) 라우터의 경로에 따라 같은 호스트로 도달하더라도 시간차 이슈가 발생할 수 있다.

### 2) IP 계층

#### 가) 기능 및 역할

- (1) IP(Internet Protocol) : 라우팅 경로 설정과 관련이 있는 프로토콜

라. TCP/UDP 계층

1) 기능 및 역할

가) 전송(Transport) 계층 : 실제 데이터의 송수신과 관련 있는 계층

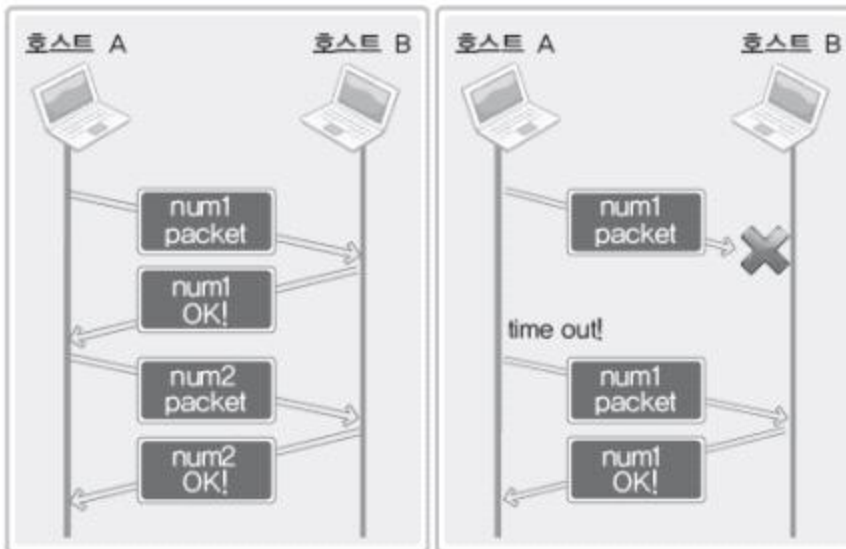
나) TCP vs UDP

(1) 신뢰성 여부

(가) TCP는 신뢰성을 보장

(2) 복잡성

(가) TCP는 신뢰성을 보장하기 때문에 UDP보다 복잡한 프로토콜이다



(나) 패킷이 보내지는데 TCP(왼쪽 그림) 시간이 더 오래 걸린다.

마. APPLICATION 계층

1) 기능 및 역할

가) 응용프로그램의 프로토콜을 구성하는 계층

나) 소켓을 기반으로 완성하는 프로토콜을 의미함

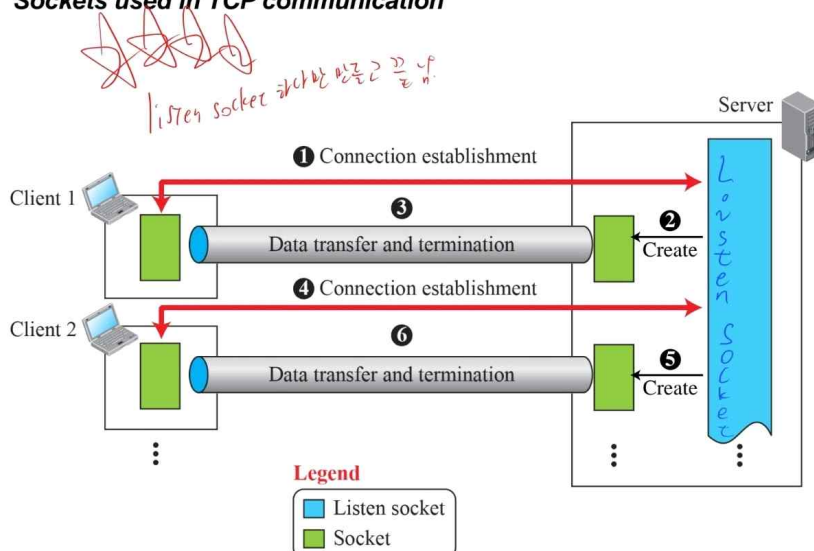
다) 소켓을 생성하면, 앞서 보인 LINK, IP, TCP/UDP 계층에 대한 내용은 감춰짐.

라) 그러니 응용 프로그래머는 APPLICATION 계층의 완성에 집중하게 됨.

## 바. 소켓을 이용한 TCP 통신

### 1) 상황 정리

#### Sockets used in TCP communication



#### Client1

- 가) ① Connection establishment => 양방향
- 나) ② Create => listen socket에서 client1와 연결할 코드를 만듦
- 다) ③ Data transfer and termin => 데이터 교환

#### Client2

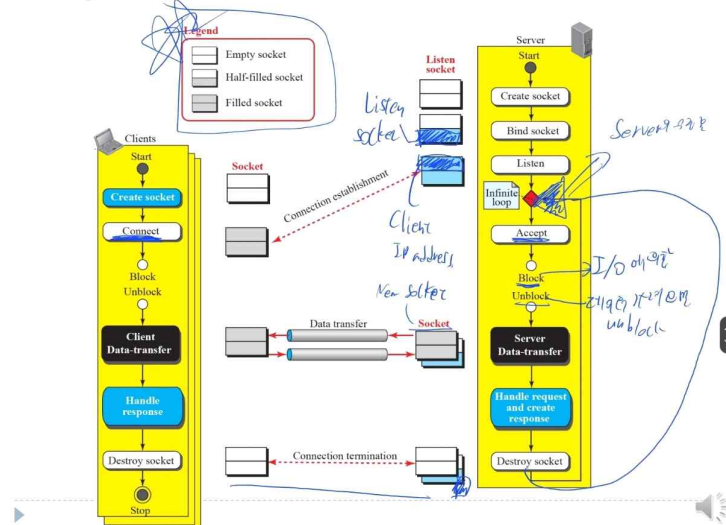
- 라) ④ Connection establishment=> 양방향
- 마) ⑤Create =>listen socket에서 client1와 연결할 코드를 만듦
- 바) ⑥Data transfer and termin

### 2) 내용 요약

- 가) 서버 입장 소켓 : 3개 ( Client Connection socket 2개, listen socket
- 나) 클라이언트 입장 소켓 : 1개
- 다) 위의 맹점? => 소켓이 1회용품이다. 여러번 할 수 있게 iterative하게 처리할 것

사. 반복적인 TCP 통신을 위한 Flow diagram

**Flow diagram for iterative TCP communication**



1) Listen과 Accept 사이에 iterative loop를 생성함으로 비워진 상태의 소켓을 재사용할 수 있다.

아. TCP 서버의 기본적인 함수호출 순서

1) socket( ) // 소켓 생성

// sock=socket( PF\_INET, SOCK\_STREAM, 0 );

2) bind( ) // 소켓 주소할당

// server : if(bind(serv\_sock, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr)) == -1)

3) listen( ) // 연결요청 대기상태

// server : if(listen(serv\_sock, 5) == -1)

// 연결 요청

// ①client : if( connect( sock, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr)) == -1)

4) accept( ) // 연결 허용

// server : clnt\_sock = accept(serv\_sock, struct sockaddr\*)&clnt\_addr, &clnt\_addr\_size);

// 연결 요청

// ②client : if( connect( sock, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr)) == -1)

5) read( ) // write( ) // 데이터 송수신

// server : write(clnt\_sock, message, sizeof(message));

// client : str\_len = read(sock, message, sizeof(message) -1);

6) close( ) // 연결 종료

// server : close(clnt\_sock); close(serv\_sock);

// client : close(sock);

bind 함수 : 호출이 되면 주소가 할당된 소켓을 얻게 됨

listen 함수 : 호출을 통해서 연결요청이 가능한 상태

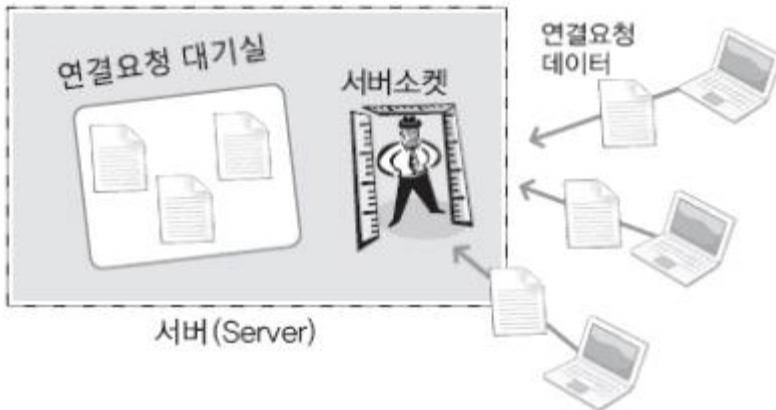
자. 연결요청 대기 상태로의 진입 - **listen( )** 메서드, 서버

```
#include <sys/types.h>
```

```
int listen(int sock, int backlog);
```

➔ 성공 시 0, 실패 시 -1 반환

- sock 연결요청 대기상태에 두고자 하는 소켓의 파일 디스크립터 전달, 이 함수의 인자로 전달된 디스크립터의 소켓이 서버 소켓(리스닝 소켓)이 된다.
- backlog 연결요청 대기 큐(Queue)의 크기정보 전달, 5가 전달되면 큐의 크기가 5가 되어 클라이언트의 연결요청을 5개까지 대기시킬 수 있다.



#### 1) 연결 요청

가) 일종의 데이터 전송

나) 리스닝 소켓

(1) 연결 요청을 받아 들이기 위한 소켓

(2) listen 함수 호출을 통해 이루어짐

## 차. 클라이언트의 연결요청 수락 - accept( ) 메서드, 서버

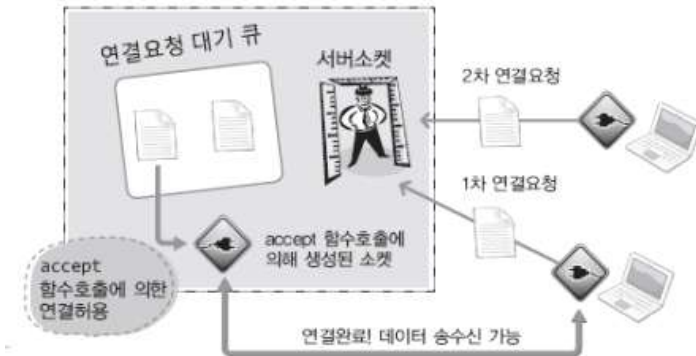
```
#include <sys/socket.h>
```

```
int accept(int sock, struct sockaddr * addr, socklen_t * addrlen);
```

➔ 성공 시 생성된 소켓의 파일 디스크립터, 실패 시 -1 반환

- sock 서버 소켓의 파일 디스크립터 전달.
- addr 연결요청 한 클라이언트의 주소정보를 담은 변수의 주소 값 전달, 함수호출이 완료되면 인자로 전달된 주소의 변수에는 클라이언트의 주소정보가 채워진다.
- addrlen 두 번째 매개변수 addr에 전달된 주소의 변수 크기를 바이트 단위로 전달, 단 크기정보를 변수에 저장한 다음에 변수의 주소 값을 전달한다. 그리고 함수호출이 완료되면 크기정보로 채워져 있던 변수에는 클라이언트의 주소정보 길이가 바이트 단위로 계산되어 채워진다.

서버 (Server)



### 1) 연결 요청

- 가) 서버 : 클라이언트 소켓과의 통신을 위한 별도의 소켓을 추가로 하나 더 생성
- 나) 생성된 소켓을 대상으로 데이터의 송수신이 진행됨
- 다) 서버의 코드를 보면 실제로 소켓이 추가로 생성됨



## 카. TCP 클라이언트의 기본적인 함수호출 순서 – connect( )

```
#include <sys/socket.h>

int connect(int sock, const struct sockaddr * servaddr, socklen_t addrlen);
```

➡ 성공 시 생성된 소켓의 파일 디스크립터, 실패 시 -1 반환

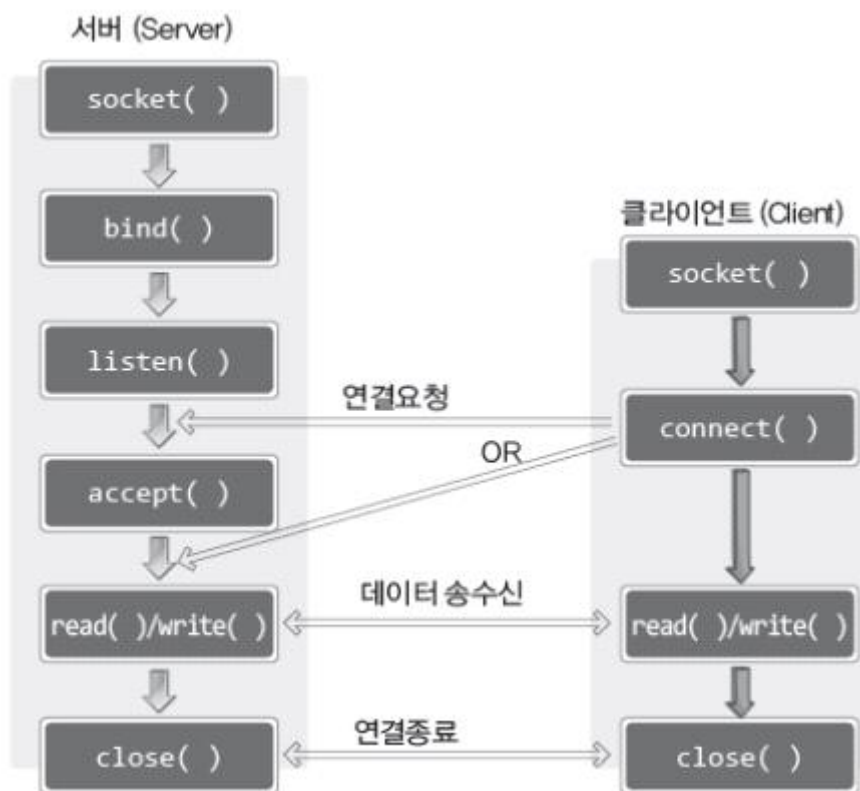
- sock     클라이언트 소켓의 파일 디스크립터 전달.
- servaddr   연결요청 한 클라이언트의 주소정보를 담은 변수의 주소 값 전달, 함수호출이 완료되면 인자로 전달된 주소의 변수에는 클라이언트의 주소정보가 채워진다.
- addrlen   두 번째 매개변수 servaddr에 전달된 주소의 변수 크기를 바이트 단위로 전달, 단, 크기정보를 변수에 저장한 다음에 변수의 주소 값을 전달한다. 그리고 함수호출이 완료되면 크기정보로 채워져 있던 변수에는 클라이언트의 주소정보 길이가 바이트 단위로 계산되어 채워진다.

- 1) socket( ) // 소켓생성
- 2) connect( ) // 연결요청
- 3) read( ) // write( ) // 데이터 송수신
- 4) close( ) // 연결 종료

클라이언트 : 소켓 생성, connect 함수 호출하는 것이 전부.

connect 함수 호출 : 연결할 서버의 주소 정보도 함께 전달한다.

타. TCP 기반 서버, 클라이언트의 함수호출 관계



파. listen( ) 메서드 호출 이후, 클라이언트 connect( ) 메서드 호출이 유효할 수 있음

- 1) connect( ) 메서드 호출

가) 서버에 의해 연결 요청 접수

(1) accept( ) 호출을 의미하는 것은 아님

(2) 클라이언트의 연결요청 정보가 서버의 대기 큐에 등록된 상황을 의미

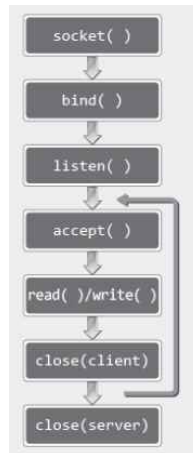
나) 네트워크 단절 등 오류상황이 발생해서 연결요청 중단

- 2) 파. 의 이유는?

가) connect( )가 반환되더라도 당장에 서비스가 이루어지지 않을 수 있기 때문이다.

#### 하. Iterative 서버의 구현

- 1) socket( )
- 2) bind( )
- 3) listen( )
- aa: // 돌아감을 표현
- 4) accept( )**
- 5) read( ) / write( )
- 6) close(client)
- goto aa: // aa: 로 돌아감을 표현
- 7) close(server)



반복적으로 accept 함수를 호출?

계속해서 클라이언트의 연결요청을 수락 가능

But, **동시에** 둘 이상의 클라이언트에게 서비스를 제공할 수 있는 모델은 **아니다**.

#### 거. 에코 클라이언트의 문제점

```
write(sock, message, strlen(message));
str_len=read(sock, message, BUF_SIZE-1);
message[str_len]=0;
printf("Message from server: %s", message);
```

- 1) PC가 여러대일 경우 등과 같은 상황에 오류를 일으킬 수 있다.
- 2) TCP의 데이터 송수신에는 경계가 존재하지 않음!

가) 위 코드의 가정

(1) "한 번의 read 함수호출로 앞서 전송된 문자열 전체를 읽어 들일 수 있다."

- 3) TCP에는 데이터의 경계가 존재하지 않기 때문에 서버가 전송한 문자열의 일부만 읽혀질 수도 있다.

## 2. 프로그램 작성

가. Hello world 프로그램 작성 ( 서버, 클라이언트 )

1) [hello\\_server.c](#)

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

void error_handling(char *message);
int main(int argc, char* argv[]){
    int serv_sock;                // 9190 setting
    int clnt_sock;                // New port birth

    struct sockaddr_in serv_addr;
    struct sockaddr_in clnt_addr;
    socklen_t clnt_addr_size;

    char message[] = "Hello World!";

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

    serv_sock=socket(PF_INET, SOCK_STREAM, 0);
    if(serv_sock == -1)
        error_handling("socket() error");

    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
    serv_addr.sin_port=htons(atoi(argv[1]));

    // while(1){
    if(bind(serv_sock, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) == -1)
        error_handling("bind() error");

    if(listen(serv_sock, 5) == -1)
        error_handling("listen() error");

    clnt_addr_size = sizeof(clnt_addr);
    clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_addr, &clnt_addr_size);
    if(clnt_sock == -1)
```

```

        error_handling("accept() error");
    // }
    printf("serv : %d, clnt : %d\n\n", serv_sock, clnt_sock);
    write(clnt_sock, message, sizeof(message));
    printf("serv : %d, clnt : %d\n\n", serv_sock, clnt_sock);
    printf("sin_family : %p\nsin_addr : %p\nsin_port : %p\n", &serv_addr.sin_family,
&serv_addr.sin_addr.s_addr, &serv_addr.sin_port);
    close(clnt_sock);
    close(serv_sock);
    return 0;
}

void error_handling(char *message){
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}

```

## 2) [hello\\_client.c](#)

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

void error_handling(char *message);

int main(int argc, char* argv[]){
    int sock;
    struct sockaddr_in serv_addr;
    char message[30];
    int str_len;

    if(argc != 3) {
        printf("Usage : %s <IP> <port>\n", argv[0]);
        exit(1);
    }

    sock= socket(PF_INET, SOCK_STREAM, 0);
    if ( sock == -1 )
        error_handling("socket() error");
    memset(&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family=AF_INET;
    serv_addr.sin_addr.s_addr=inet_addr(argv[1]);
    serv_addr.sin_port=htons(atoi(argv[2]));

    if(connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr))== -1)
        error_handling("connect() error!");

    str_len=read(sock, message, sizeof(message)-1);
    if(str_len== -1)
        error_handling("read() error!");

    printf("Message from server: %s\n", message);
    close(sock);
    return 0;
}

void error_handling(char *message){
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}
```

### 3) 실행결과

```
server@server-virtual-machine:~$ ./hserver 9190
serv : 3, clnt : 4

serv : 3, clnt : 4

sin_family : 0x7ffc993d48f0
sin_addr : 0x7ffc993d48f4
sin_port : 0x7ffc993d48f2

server@server-virtual-machine: ~
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)
server@server-virtual-machine:~$ ./hclient 127.0.0.1 9190
Message from server: Hello World!
```

나. TCP Iterative echo 프로그램 작성 ( 서버, 클라이언트 )

1) [echo\\_server.c](#)

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

#define BUF_SIZE 1024
void error_handling(char *message);

int main(int argc, char *argv[]){
    int serv_sock, clnt_sock;
    char message[BUF_SIZE];
    int str_len, i;

    struct sockaddr_in serv_adr, clnt_adr;
    socklen_t clnt_adr_sz;

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

    serv_sock=socket(PF_INET, SOCK_STREAM, 0);
    if(serv_sock==-1)
        error_handling("socket() error");

    memset(&serv_adr, 0, sizeof(serv_adr));
    serv_adr.sin_family=AF_INET;
    serv_adr.sin_addr.s_addr=htonl(INADDR_ANY);
    serv_adr.sin_port = htons(atoi(argv[1]));

    if(bind(serv_sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr))==-1)
        error_handling("bind() error");
    if(listen(serv_sock, 5) == -1)
        error_handling("listen() error");

    clnt_adr_sz = sizeof(clnt_adr);

    for(i = 0; i < 5; i++){
        clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_adr, &clnt_adr_sz);
        if(clnt_sock == -1)
            error_handling("accept() error");
        else
            printf("Connected client %d\n", i + 1);
    }
```

```

        while((str_len=read(clnt_sock, message, BUF_SIZE)) != 0 )
            write(clnt_sock, message, str_len);

        close(clnt_sock);
    }

    close(serv_sock);
    return 0;
}

void error_handling(char *message){
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}

```



## 2) [echo\\_client.c](#)

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

#define BUF_SIZE 1024
void error_handling(char *message);

int main(int argc, char *argv[]){
    int sock;
    char message[BUF_SIZE];
    int str_len;
    struct sockaddr_in serv_adr;

    if(argc != 3){                // ip와 포트 입력을 받았는지 여부 체크
        printf("Usage : %s <IP> <port>\n", argv[0]);
        exit(1);
    }

    sock = socket(PF_INET, SOCK_STREAM, 0);
    if (sock == -1)
        error_handling("socket() error");

    memset(&serv_adr, 0, sizeof(serv_adr));
    serv_adr.sin_family=AF_INET;
    serv_adr.sin_addr.s_addr=inet_addr(argv[1]);
    serv_adr.sin_port=htons(atoi(argv[2]));

    if(connect(sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr)) == -1)
        error_handling("connect() error!");
    else
        puts("Connected.....");

    while(1)                      // iterative echo
    {
        fputs("Input message(Q to quit): ", stdout);    // 멈춤 입력을 받기 위함
        fgets(message, BUF_SIZE, stdin);                // 메시지 입력을 위함

        if(!strcmp(message, "q\n") || !strcmp(message, "Q\n")) // 멈춤 진행
            break;

        write(sock, message, strlen(message));
    }
}
```

```

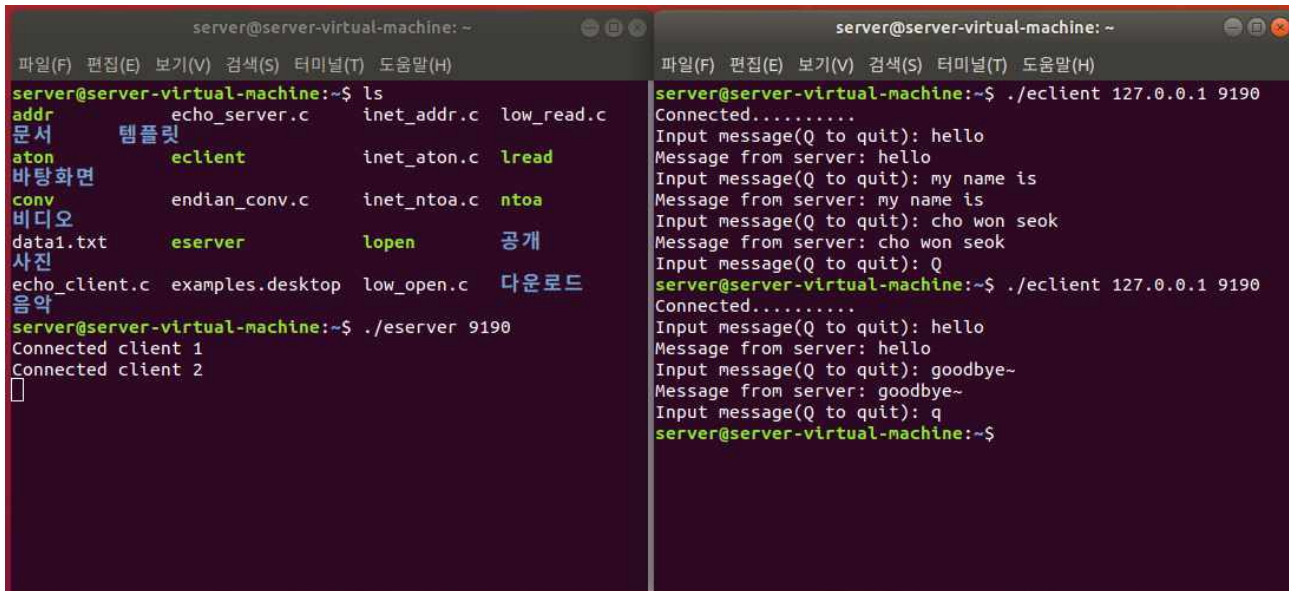
        str_len=read(sock, message, BUF_SIZE-1);
        message[str_len] = 0;
        printf("Message from server: %s", message);
    }

    close(sock);
    return 0;
}

void error_handling(char *message){
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}

```

### 3) 실행결과

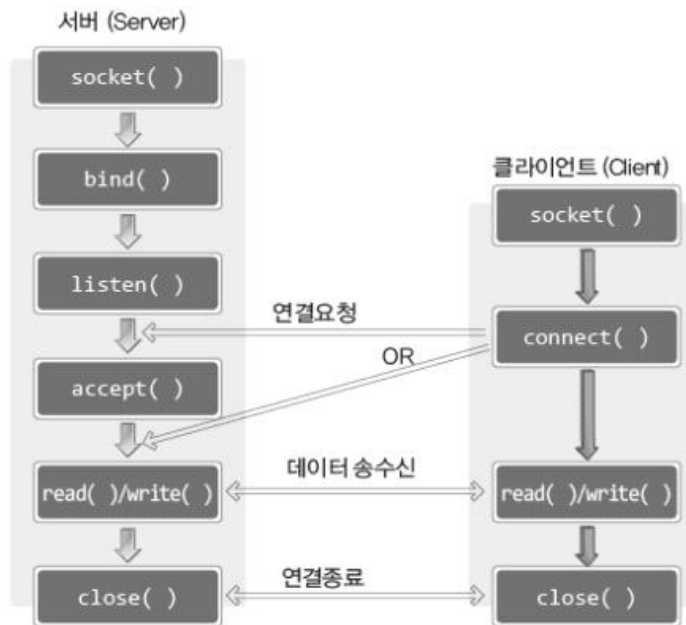


```
server@server-virtual-machine: ~  
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)  
server@server-virtual-machine:~$ ls  
addr      echo_server.c  inet_addr.c  low_read.c  
문서      템플릿  
aton      eclient        inet_aton.c  lread  
바탕화면  
conv      endian_conv.c  inet_ntoa.c  ntoa  
비디오  
data1.txt  eserver        lopen        공개  
사진  
echo_client.c  examples.desktop  low_open.c  다운로드  
음악  
server@server-virtual-machine:~$ ./eserver 9190  
Connected client 1  
Connected client 2  
□  
  
server@server-virtual-machine: ~  
파일(F) 편집(E) 보기(V) 검색(S) 터미널(T) 도움말(H)  
server@server-virtual-machine:~$ ./eclient 127.0.0.1 9190  
Connected.....  
Input message(Q to quit): hello  
Message from server: hello  
Input message(Q to quit): my name is  
Message from server: my name is  
Input message(Q to quit): cho won seok  
Message from server: cho won seok  
Input message(Q to quit): Q  
server@server-virtual-machine:~$ ./eclient 127.0.0.1 9190  
Connected.....  
Input message(Q to quit): hello  
Message from server: hello  
Input message(Q to quit): goodbye~  
Message from server: goodbye~  
Input message(Q to quit): q  
server@server-virtual-machine:~$
```

다. 2.가 & 2.나 비교하기

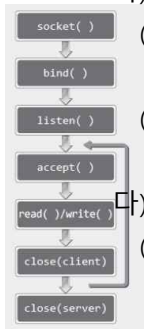
1) 2.가

가) 구조



- (1) hello\_server / hello\_client를 나타내며 server char message[]에 저장된 hello world! 라는 메시지를 9190포트를 통해 단순히 전달한다.
- (2) listen( )부분에서 connect를 받으면 accept( )에서 수용한다
- (3) write( )를 통해 server는 메시지를 보내고 read( )를 통해 client는 메시지를 받는다.
- (4) close( )를 통해 소켓 통신을 종료한다.

나) 2.나



- (1) 위의 1) 2.가에서 server의 listen( )이후와 close(server)이전 부분을 for문으로 5번 반복하고 client에서 q(멈춤) 입력을 받을 때까지 서버의 연결을 유지한다.
- (2) client는 while문에서 q(멈춤) 입력을 받기 위한 부분과 client의 메시지를 보내기 위한 write( ). client 본인의 메시지를 받기 위한 read( )를 통해 echo( 메아리 ) 작업을 진행한다.

다) 결론

(1) 2.가와 2.나의 차이

- (가) client에서의 메시지 입력 유무 ( write )
  - (나) client와 server의 반복적인 접속 가능 여부
    - ① client : while문
    - ② server : for문
- 정도의 차이가 있다고 할 수 있습니다.