

TCP/IP 기반 서버 클라이언트

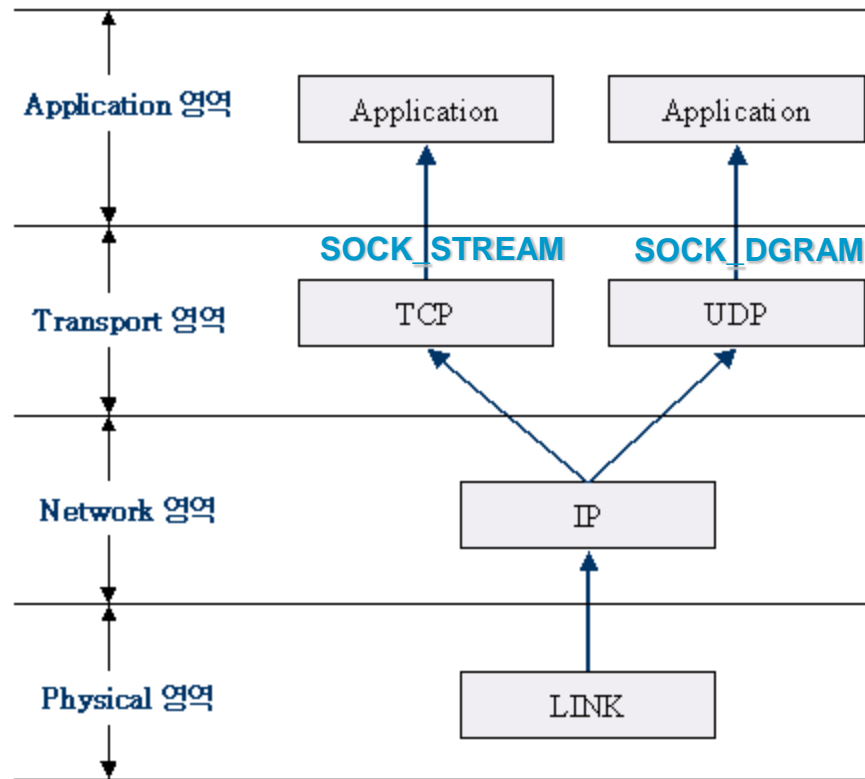
인하공업전문대학 컴퓨터정보과
최효현 교수

주요사항

- ❑ TCP와 UDP에 대한 이해
- ❑ TCP/IP 기반의 서버, 클라이언트의 이해
- ❑ Iterative서버의 구현
- ❑ TCP에서의 경계 (Boundary)를 처리하기 위한 방법
- ❑ TCP에서의 버퍼의 존재
- ❑ TCP의 내부 구조

4.1 TCP와 UDP에 대한 이해

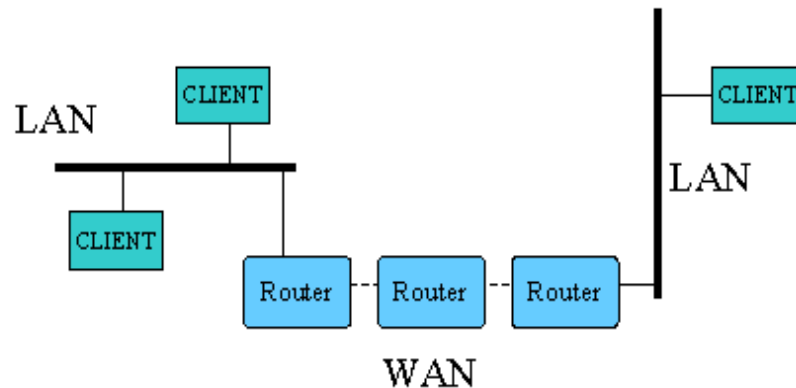
• TCP/IP 프로토콜 스택



4.1 TCP와 UDP에 대한 이해

• LINK 계층

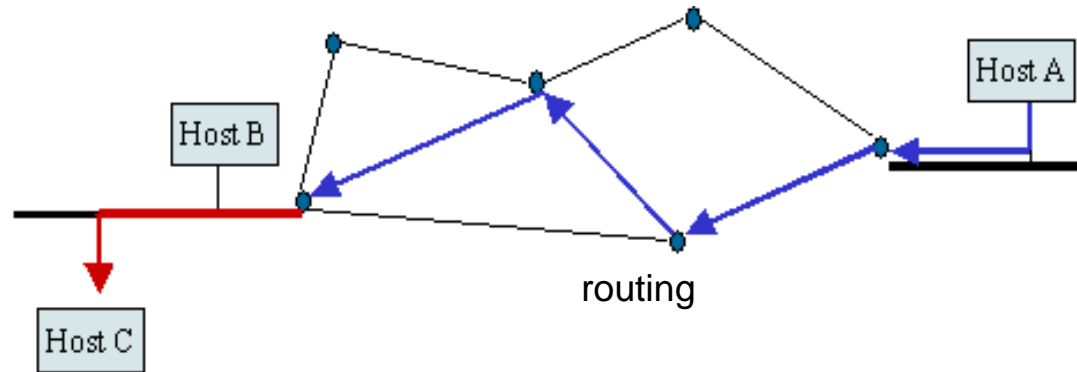
- 물리적인 영역을 담당.
- LAN의 **MAC** 프로토콜과 WAN의 **L2** 프로토콜 영역



4.1 TCP와 UDP에 대한 이해

• IP 계층

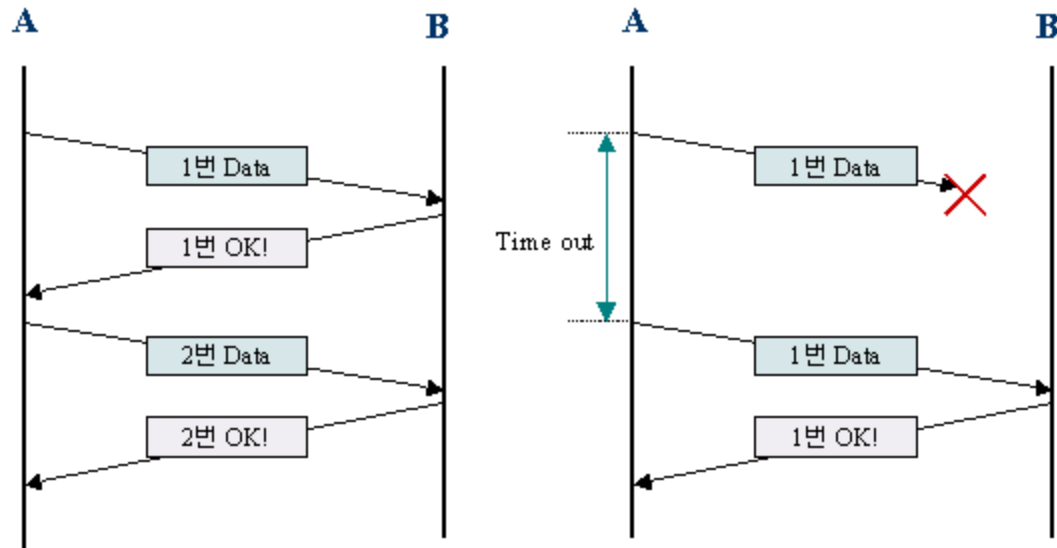
- 네트워크를 통한 데이터 전송을 담당한다.
- **IP** 주소를 사용, 경로를 선택해서 목적지로 라우팅 함



4.1 TCP와 UDP에 대한 이해

• TCP/UDP 계층

- **TCP와 IP의 관계**
 - : **TCP**에서 **IP**를 사용하여 데이터 전송
 - : **IP**는 오직 한 패킷의 전송에만 관심을 둬
 - : **TCP**는 데이터의 수신을 확인하는 메커니즘을 사용



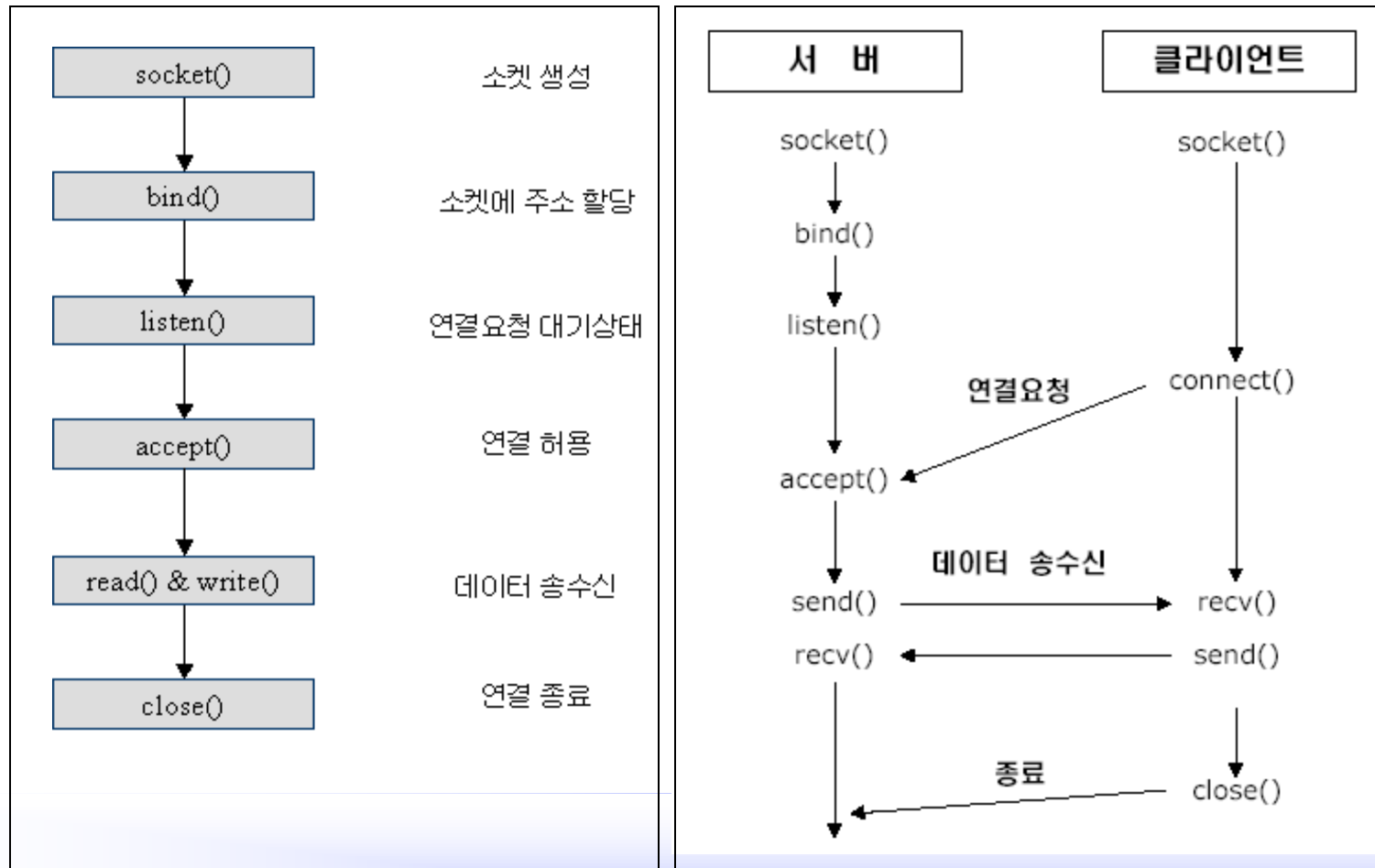
4.1 TCP와 UDP에 대한 이해

• 응용 계층

- 소켓을 이용한 프로그램의 구현을 의미한다.
- 일반적으로 소켓 프로그래밍이라고 하면 **Application** 계층의 프로토콜을 정의하고 구현하는 것을 말한다.
- “**Hello World**” 서버 / 클라이언트도 **Application** 프로토콜의 구현이다.
- 지금까지 이야기 해 온 내부 구조를 알지 못해도 소켓 프로그래밍이 가능하다.(소켓이 우리에게 제공하는 이점)

4.2 TCP 기반 서버, 클라이언트의 구현

• TCP 서버에서의 기본적인 함수 호출 순서



4.2 TCP 기반 서버, 클라이언트의 구현

- ‘연결 요청 대기 상태’로의 진입

1. **listen** 함수는 전달되는 인자의 소켓을 ‘서버 소켓’이 되게 한다.
2. **listen** 함수는 ‘연결 요청 대기 큐’를 생성 한다.

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

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

sock : 연결요청 대기상태에 두고자 하는 소켓의 파일 디스크립터,
디스크립터 소켓의 서버 소켓 (리스닝 소켓)이 된다.

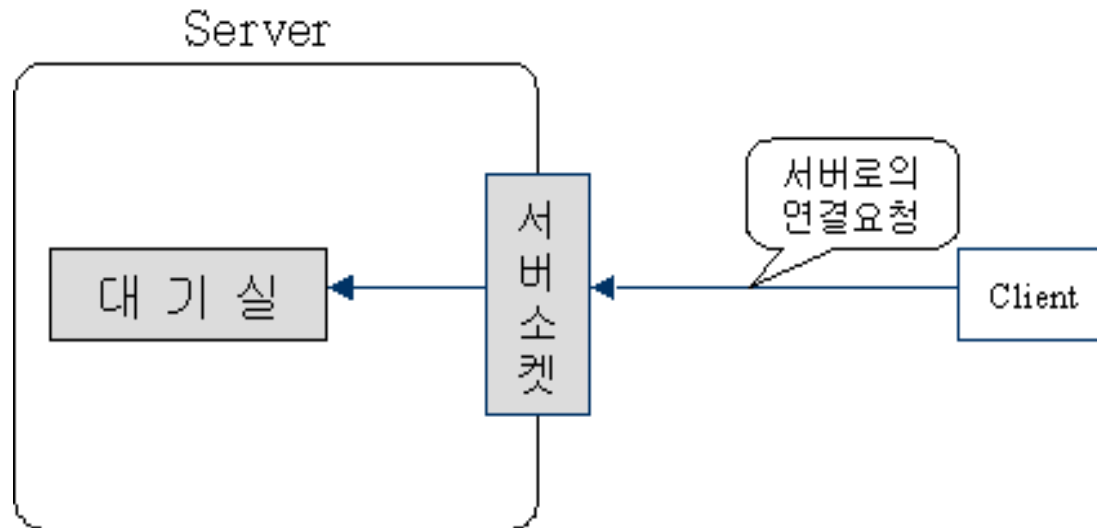
backlog: 연결요청 대기 큐의 크기 정보

5가 되면 큐의 크기가 5가 되어 클라이언트의 요청을 5개까지
대기시킬 수 있다.

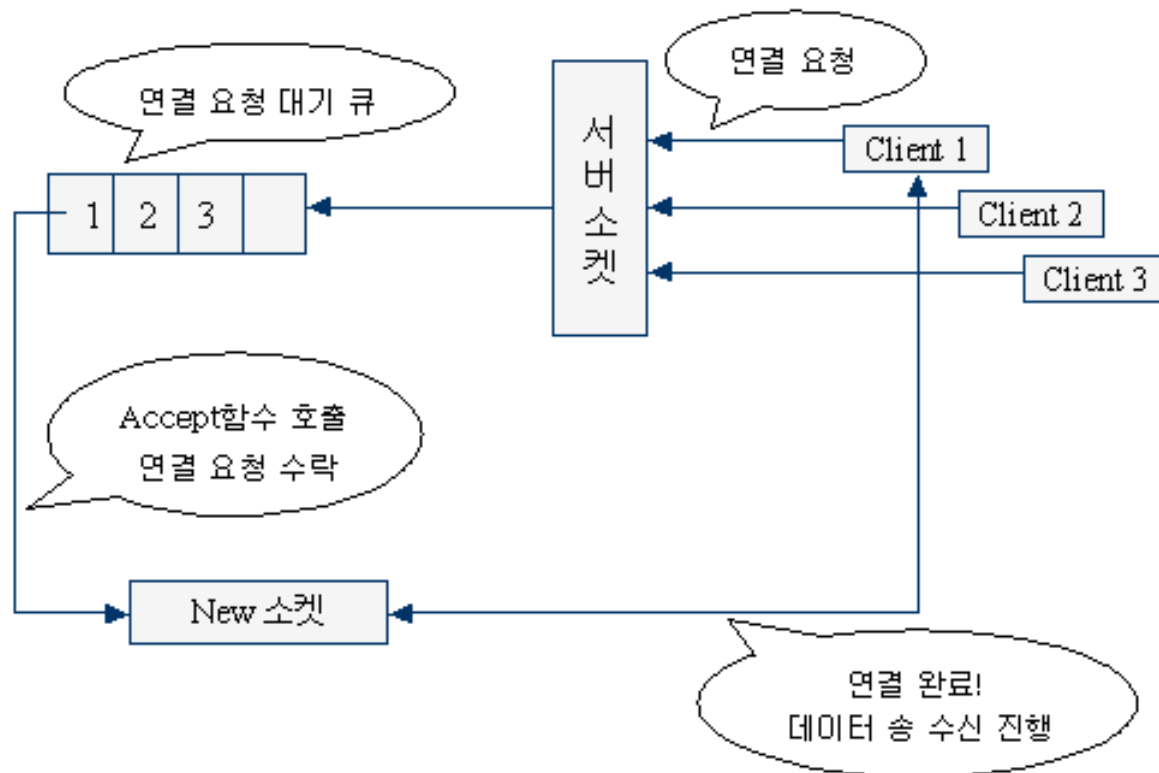
4.2 TCP 기반 서버, 클라이언트의 구현

- 서버의 역할과 연결요청 대기상태

1. 서버 소켓은 일종의 ‘문지기’ 이다.



4.2 TCP 기반 서버, 클라이언트의 구현



4.2 TCP 기반 서버, 클라이언트의 구현

- 연결요청 수락하기

1. 연결요청 대기 큐(queue)에 존재하는 클라이언트의 연결 요청 수락.

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

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

sock : 서버 소켓의 파일 디스크립터,

addr: 연결 요청한 클라이언트의 주소 정보를 담은 변수의 주소 값 전달

함수 호출이 완료되면 인자로 전달된 주소의 변수에는 클라이언트의 주소 정보가 채워짐

addrlen: **addr**에 전달된 주소의 변수 크기를 바이트 단위로 전달

4.2 TCP 기반 서버, 클라이언트의 구현

- Hello World 서버 다시 보기

1. 프로그램 예제
 - helloworld_server.c

2. 실행결과

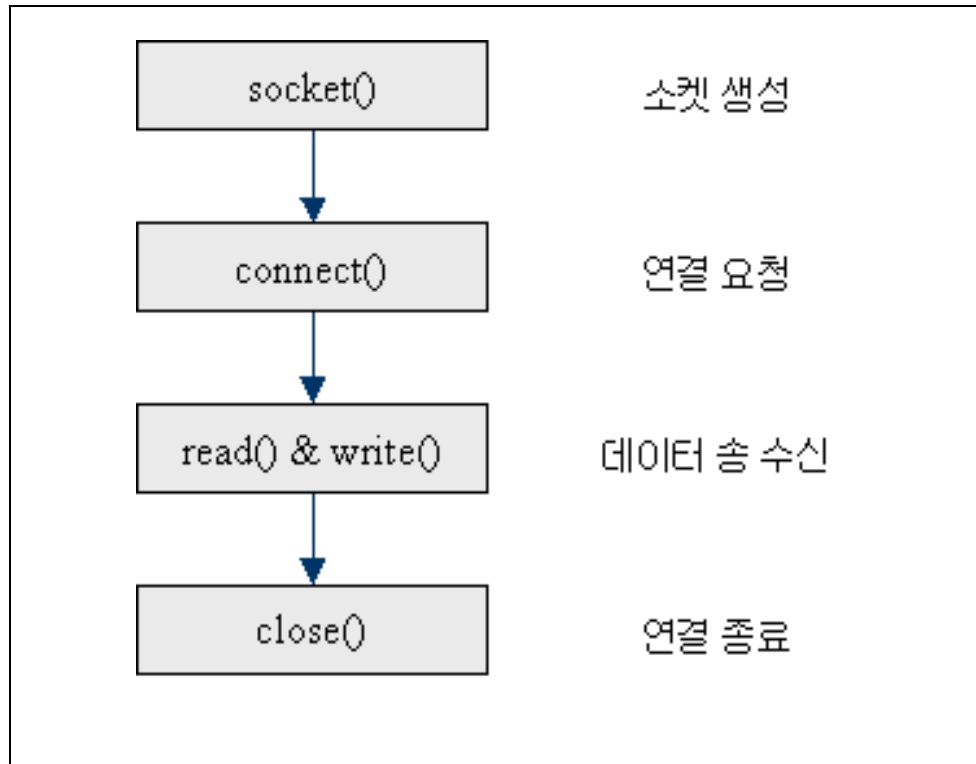


```
root@localhost.localdomain: /booksource/1jang/source
[root@localhost source]# gcc helloworld_server.c -o server
[root@localhost source]# ./server 9190
[root@localhost source]#
```

[영어] [완성] [두벌식]

4.2 TCP 기반 서버, 클라이언트의 구현

- 클라이언트의 기본적인 함수 호출 순서



4.2 TCP 기반 서버, 클라이언트의 구현

- 연결 요청 함수

1. 소켓과 목적지 주소에 대한 정보를 마련 해 두고 나서 연결 요청을 시도 한다.

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

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

sock : 클라이언트 소켓의 파일 디스크립터,

servaddr: 연결 요청할 서버의 주소 정보를 담은 변수의 주소 값

addrlen: **servaddr**에 전달된 주소의 변수 크기를 바이트 단위로 전달

4.2 TCP 기반 서버, 클라이언트의 구현

- Hello World 클라이언트 다시 보기

1. 프로그램 예제

- helloworld_client.c

2. 실행결과

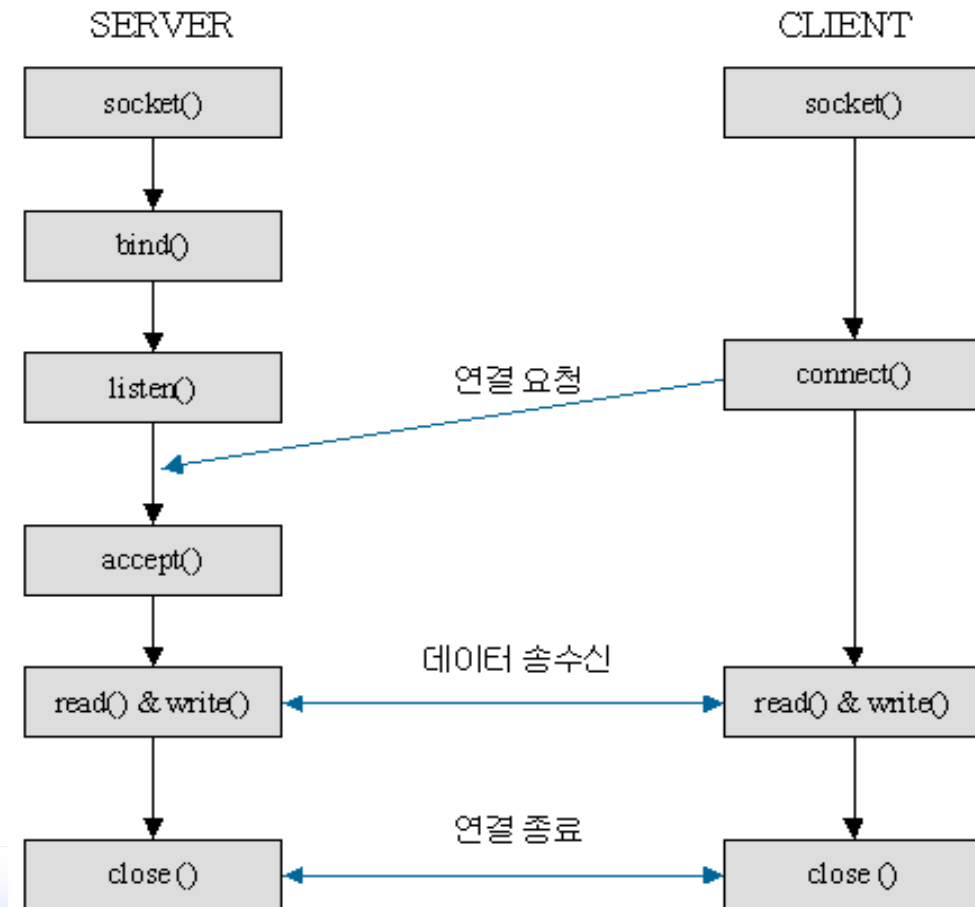
```
root@localhost.localdomain: /booksource/1jang/source
[root@localhost source]# gcc helloworld_client.c -o client
[root@localhost source]# ./client 127.0.0.1 9190
Message from server : Hello World!

[root@localhost source]#
```

[영어][완성][두벌식]

4.3 Iterative 기반의 서버, 클라이언트의 구현

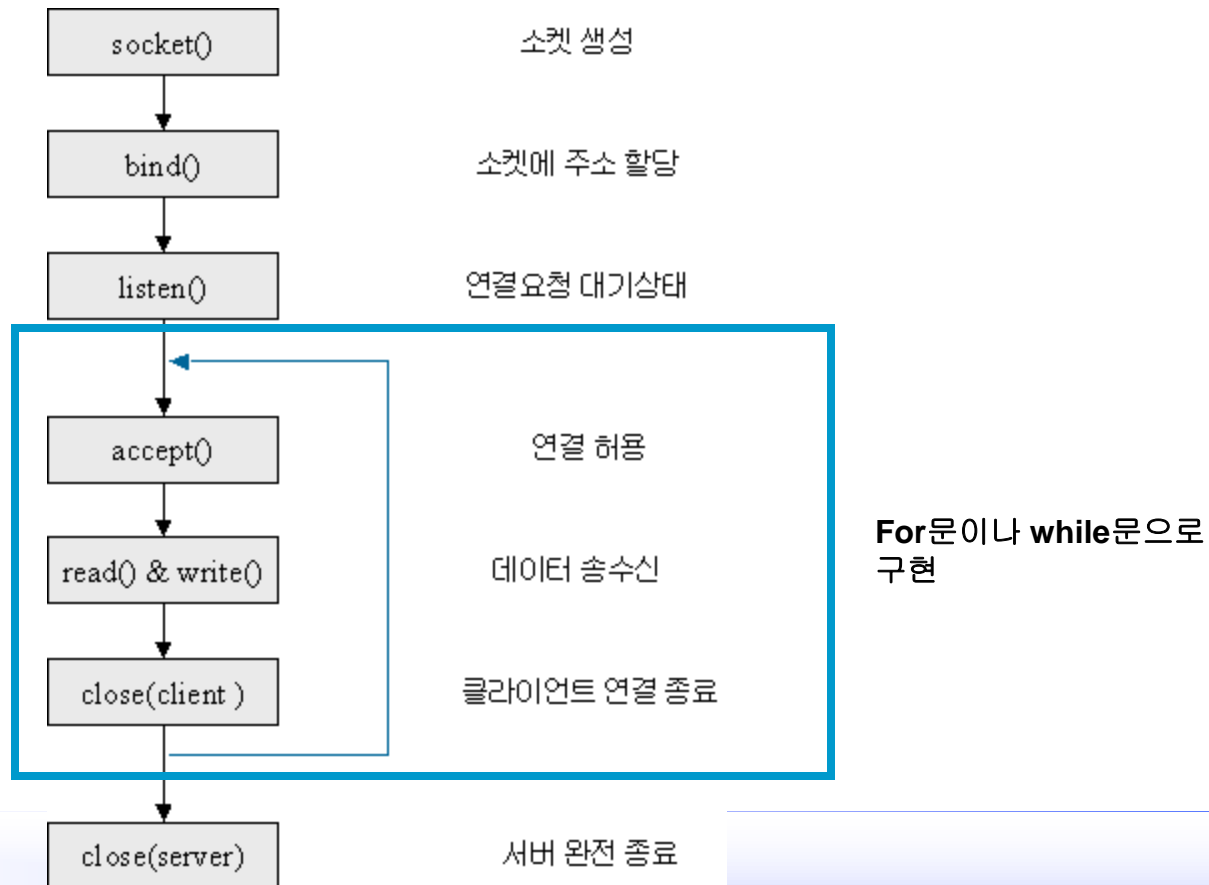
- TCP 서버/클라이언트 함수호출 관계



4.3 Iterative 기반의 서버, 클라이언트의 구현

• Iterative 서버의 구현

1. Iterative 서버 : 반복해서 클라이언트의 요청을 처리한다.



4.3 Iterative 기반의 서버, 클라이언트의 구현

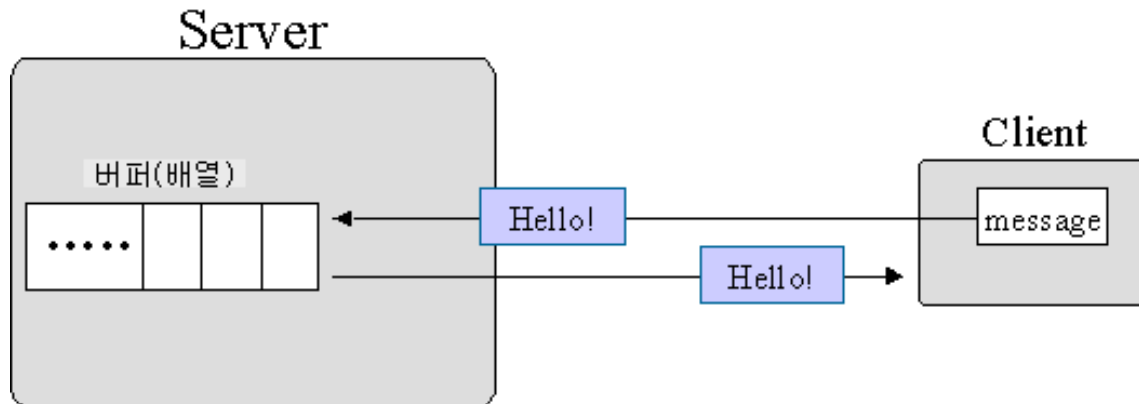
• Iterative 서버

```
for( i = 0; i < 5; i++){  
    clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_addr, &clnt_addr_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 );  
}
```

4.3 Iterative 기반의 서버, 클라이언트의 구현

- 에코(echo) 서버/클라이언트의 구현

1. 에코 서버/클라이언트의 기능 : 클라이언트가 전송해 주는 데이터를 그대로 되돌려 전송해주는 기능의 서버



4.3 Iterative 기반의 서버, 클라이언트의 구현

2. 서버에서 **read()** 함수의 종료

- 클라이언트가 **EOF**를 보낼 때까지 데이터를 받아서 화면에 표시
- 서버는 **EOF** 메시지를 받으면 **0**을 리턴
- **EOF** 전달은 **close()**로 종료시에 자동으로 전달

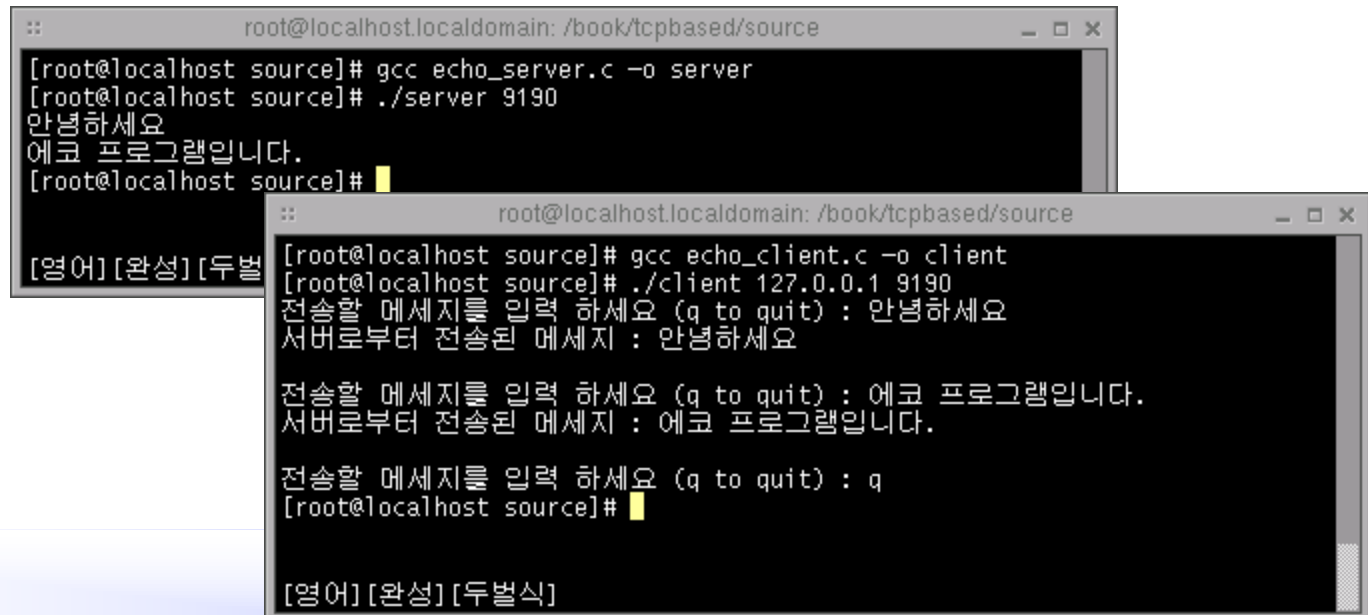
4.3 Iterative 기반의 서버, 클라이언트의 구현

- 예제 확인

1. 프로그램 예제

- echo_server.c, echo_client.c

2. 실행결과



```
root@localhost.localdomain: /book/tcpbased/source
[ root@localhost source ]# gcc echo_server.c -o server
[ root@localhost source ]# ./server 9190
안녕하세요
예코 프로그램입니다.
[ root@localhost source ]#

[영어][완성][두벌]

root@localhost.localdomain: /book/tcpbased/source
[ root@localhost source ]# gcc echo_client.c -o client
[ root@localhost source ]# ./client 127.0.0.1 9190
전송할 메시지를 입력 하세요 (q to quit) : 안녕하세요
서버로부터 전송된 메시지 : 안녕하세요

전송할 메시지를 입력 하세요 (q to quit) : 예코 프로그램입니다.
서버로부터 전송된 메시지 : 예코 프로그램입니다.

전송할 메시지를 입력 하세요 (q to quit) : q
[ root@localhost source ]#

[영어][완성][두벌식]
```

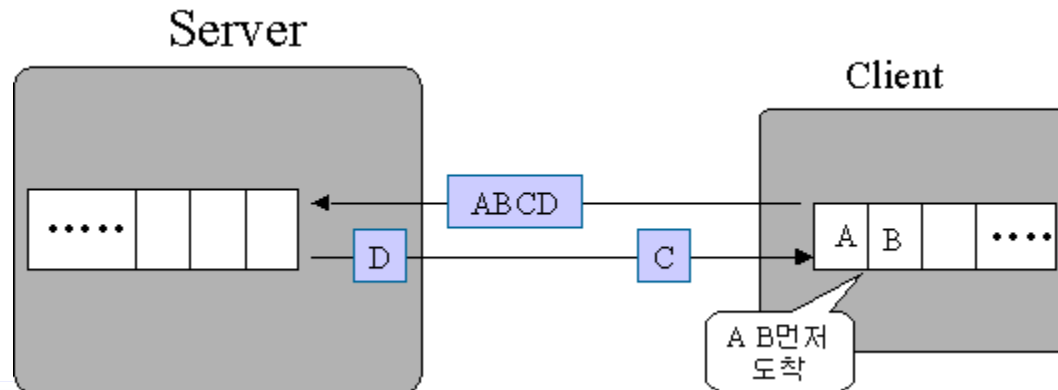
5.1 에코 클라이언트의 완벽구현

• TCP 기반의 데이터 전송 특징

- 한번의 데이터 전송함수 호출(**write, send**)이 늘 하나의 패킷을 형성하는 것은 아님
- **TCP**는 연결지향 프로토콜로 전송되는 데이터의 경계(**boundary**)가 없음
- 한번에 **write()**를 사용하여 “**ABCD**” 문자를 전송할지라도, 그 데이터들이 하나의 패킷을 형성하여 전송 되는것은 아님

(**AB**가 실린 패킷, **C**가 실린 패킷, **D**가 실린 패킷 순으로 3개 패킷으로 전송도 가능함)

=> 일부 프로그램 변경이 필요함



5.1 에코 클라이언트의 완벽구현

• 다시 구현하는 TCP기반의 에코 클라이언트

- 이전 예에서는 “**AB**” 한 패킷만을 수신하고, 모든 패킷을 수신한 것으로 인식하여 **read()** 함수를 호출하는 오류가 발생할 수 있음
- 이전 에코 프로그램 : **read()**와 **write()**함수를 한번씩 만 호출하도록 작성됨
 - => **write()** 함수로 전송시 반드시 하나의 패킷으로 구성되어야 정상적으로 동작
 - => **TCP**에서는 이것을 보장 안함
- 에코 부분을 수정해야 함

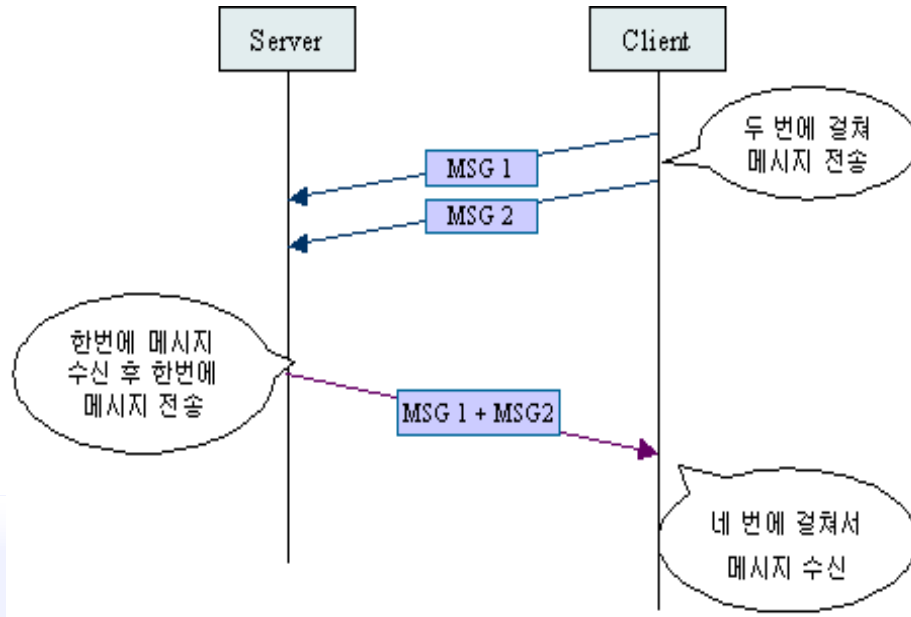
```
str_len = write(sock, message, strlen(message));
recv_len = 0;
while(recv_len < str_len) {
    recv_cnt = read(sock, &message[recv_len], BUF_SIZE-1);
    if(recv_cnt == -1)
        error_handling("read() error !.");
    recv_len += recv_cnt;
}
```


5.2 TCP의 이론적인 이야기

1. **TCP** 서버/클라이언트가 주고받는 데이터에 경계가 없다는 사실을 확인
2. 제시되는 예제 시나리오 <그림>

- 방법 1 : 클라이언트가 두 번에 걸쳐 메시지를 전송(즉, **write()**를 두 번 호출), 서버는 전송되어 오는 메시지를 한번의 **read()**로 모두 읽음
- 방법 2 : 서버는 한번의 **write()**로 클라이언트로 전송, 클라이언트는 네 번의 **read()** 호출로 모든 메시지를 수신함

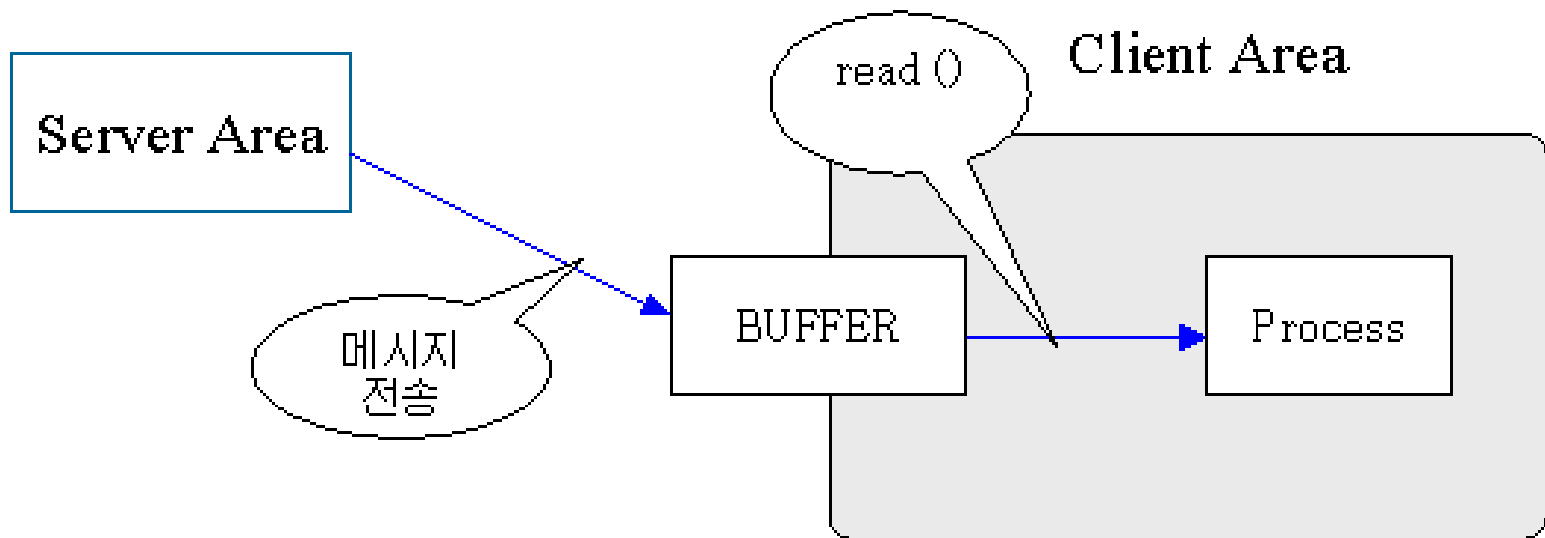
**** TCP는 데이터 송수신 함수의 호출 횟수는 큰 의미를 지니지 않는다.(UDP는 반대)**



5.2 TCP의 이론적인 이야기

• 버퍼의 존재 1

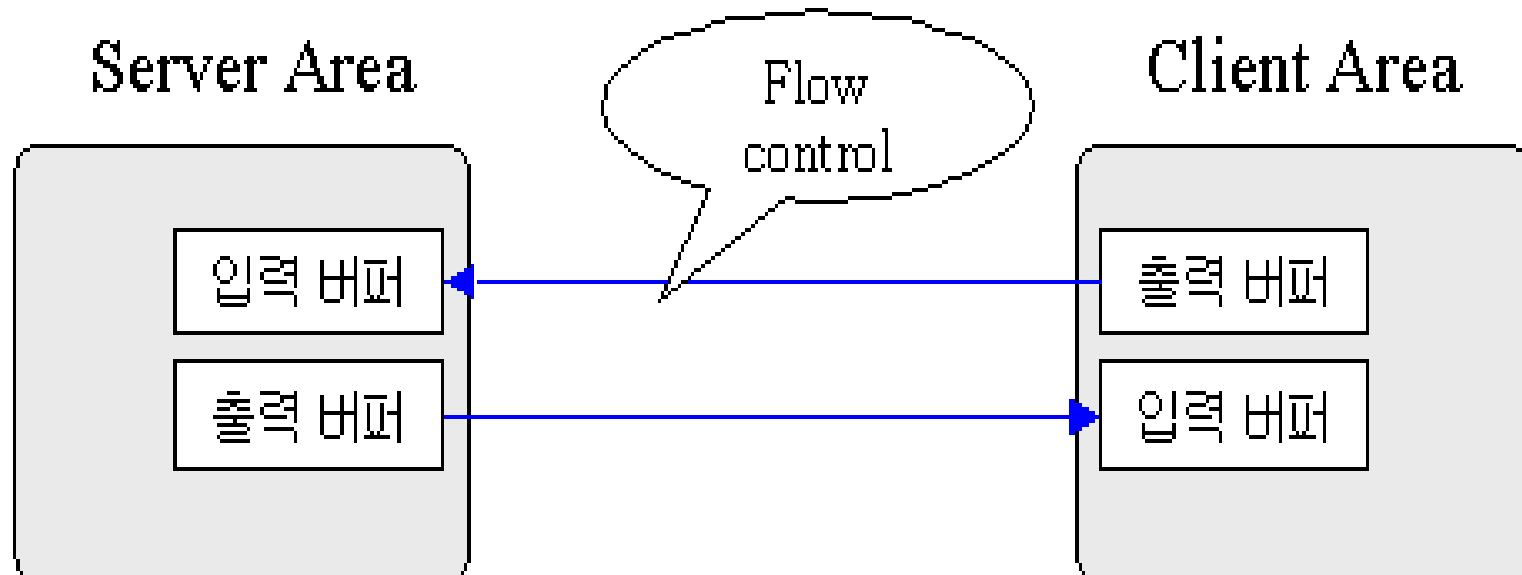
1. 이미 전송된 데이터는 어디에서 존재 하고 있었는가?



5.2 TCP의 이론적인 이야기

• 버퍼의 존재 2

1. 입력 버퍼와 출력 버퍼의 역할.



5.2 TCP의 이론적인 이야기

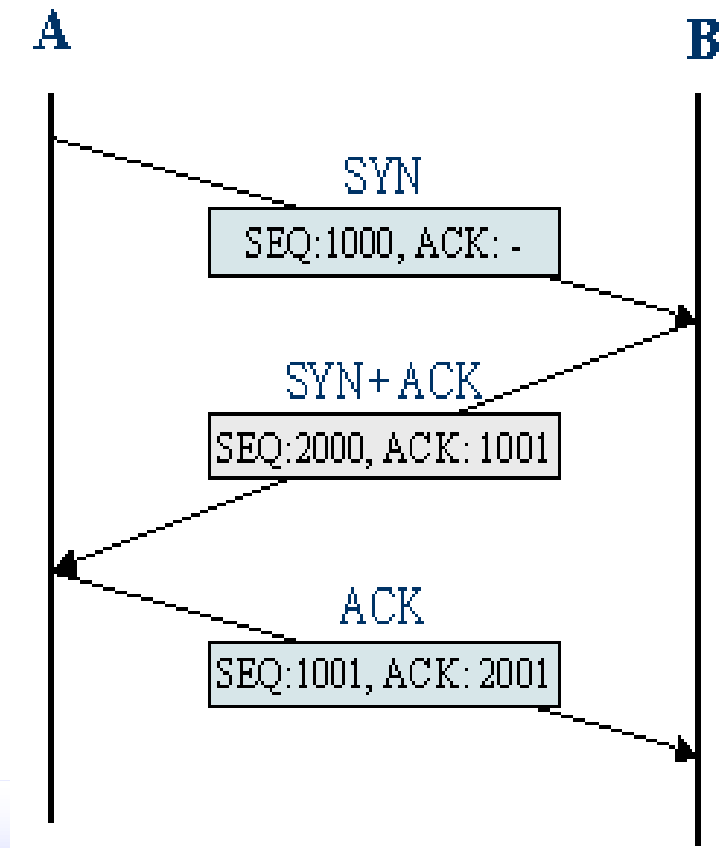
• TCP의 데이터 전송 과정.

1. 첫 번째 : 연결 설정 단계
 - 클라이언트가 `connect` 함수 호출 시 진행.
2. 두 번째 : 데이터 송 수신 단계
 - 서버/클라이언트간 데이터 송 수신 함수 호출 과정에서 진행.
3. 세 번째 : 연결 종료 단계
 - 클라이언트 혹은 서버가 `close()` 함수호출 시 진행.

5.2 TCP의 이론적인 이야기

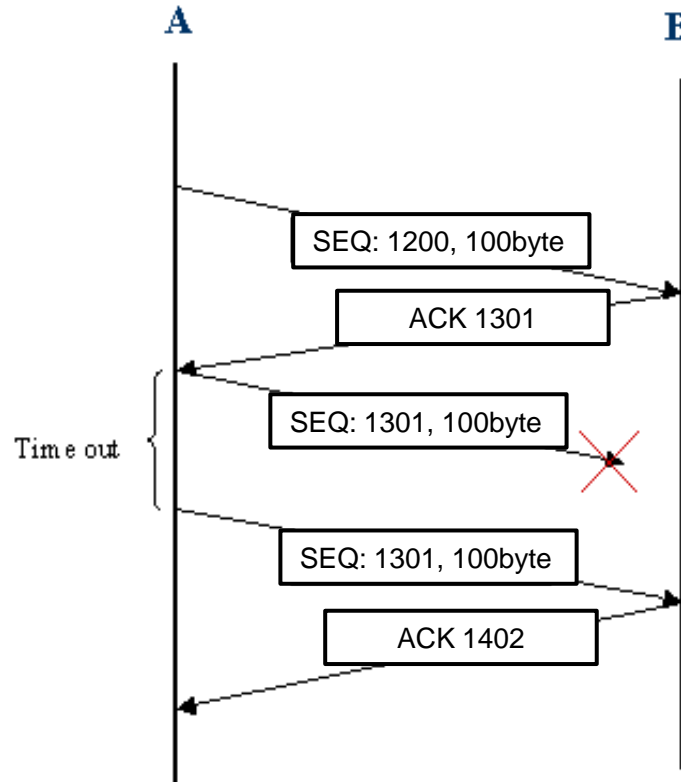
- 연결 설정 단계

1. Three-way handshaking



5.2 TCP의 이론적인 이야기

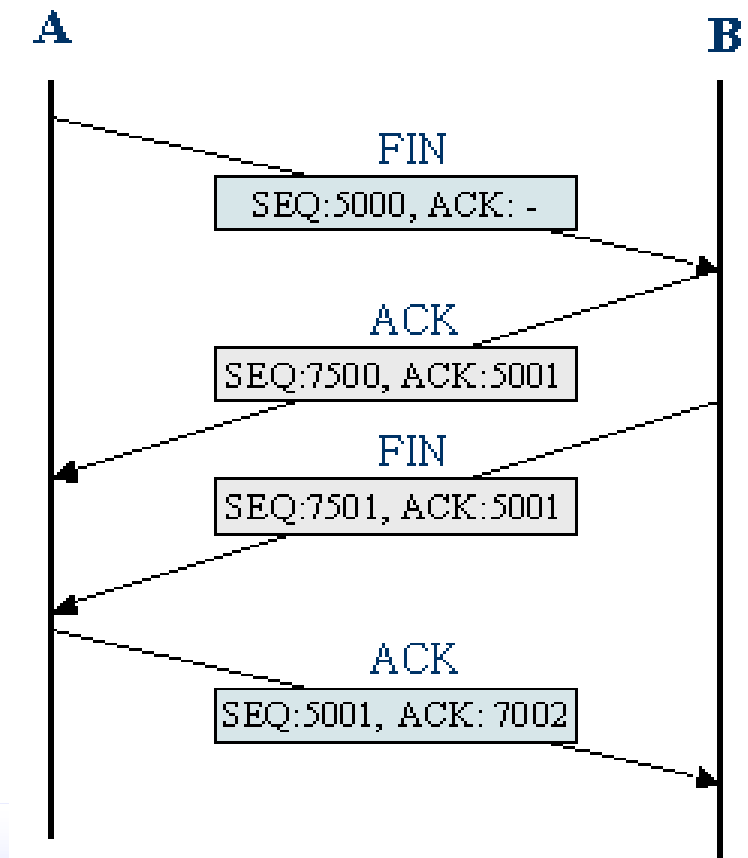
- 데이터 송수신 단계.



<그림 5-8>

5.2 TCP의 이론적인 이야기

- 연결 종료 단계.



Q&A

