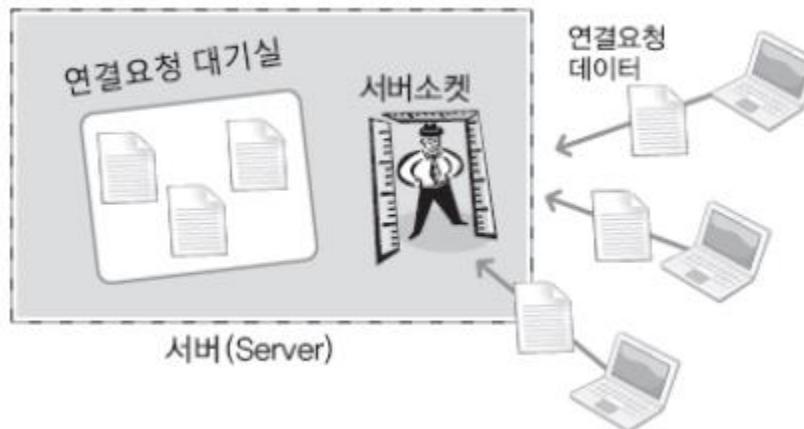


TCP 서버

```
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

→ 성공 시 0, 실패 시 -1 반환

소켓에 할당된 IP와 PORT번호로 연결요청이 가능한 상태가 된다.



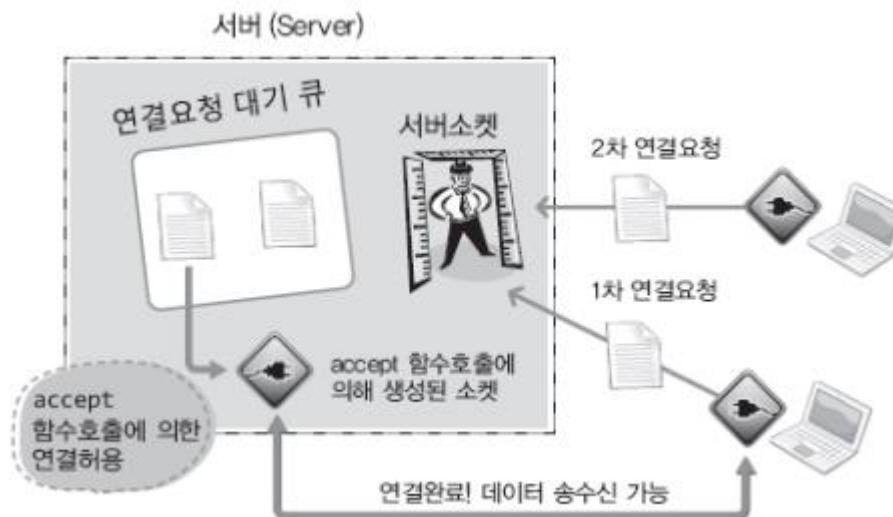
1. 연결요청도 일종의 데이터 전송(연결요청 데이터 전송)
2. 연결요청을 받아들이기 위해서도 하나의 소켓이 필요
→ **서버소켓 or 리스닝 소켓**

연결요청 대기실에 **연결요청 데이터**를 넣어둔다.

```
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

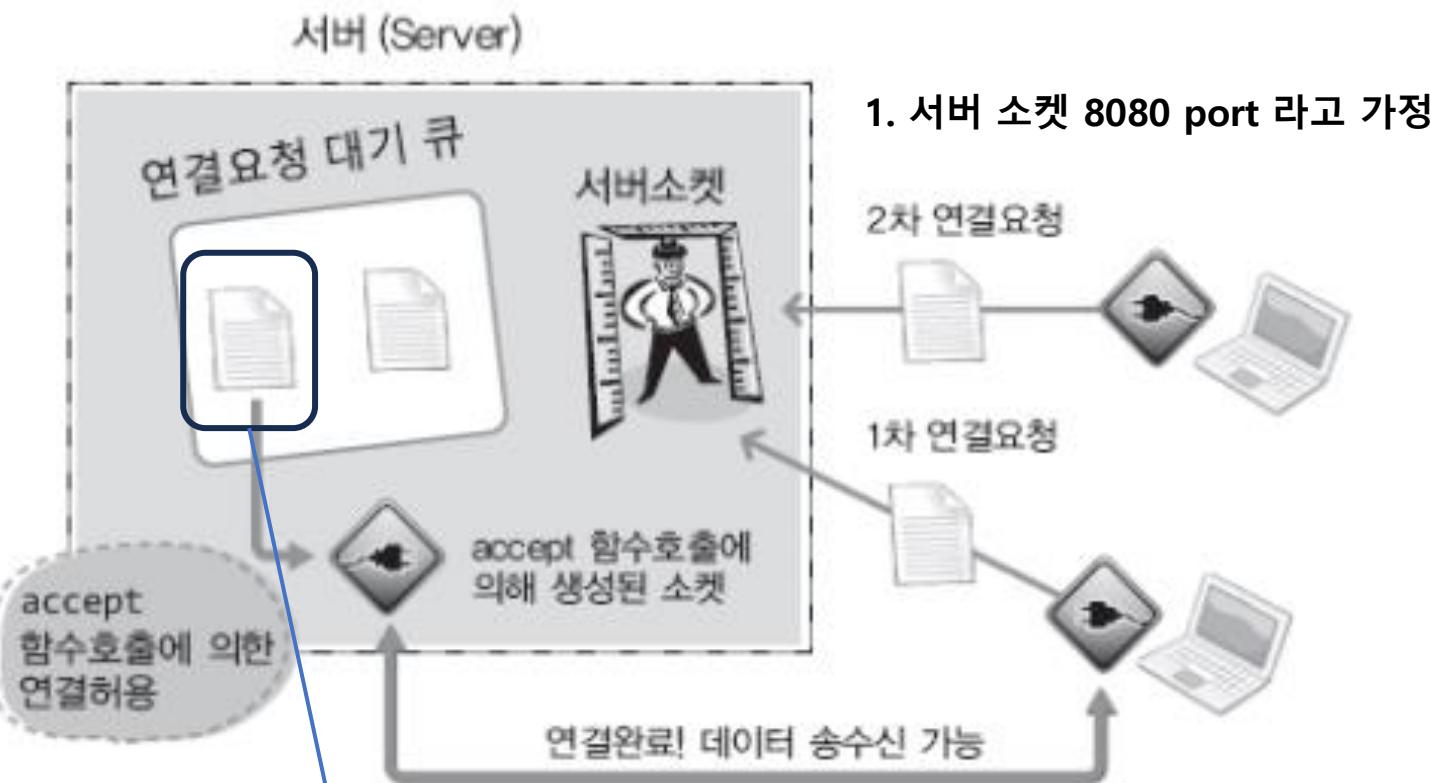
→ 성공 시 파일 디스크립터, 실패 시 -1 반환

accept 함수호출 이후에는 데이터의 송수신이 가능하다. 단, 연결요청이 있을 때에만 accept 함수가 반환을 한다.



연결요청 정보 참고 → 별도의 소켓을 추가 생성해당 소켓을 통해 통신한다.

TCP 서버



2. 이 요청에 매핑되는 port는?
❖ 똑같이 8080 port이다.

즉, 모든 socket이 8080 port를 가진다.

질문) 하지만, 전부다 8080포트면 어떻게 구분하는가?

답) TCP socket은 서버의 정보 뿐만 아니라 클라이언트의 정보도 포함해서 구분된다.
따라서 클라이언트가 다르면 서버 소켓의 정보가 같더라도 운영체제가 구분할 수 있다.

TCP 입출력 버퍼

버퍼는 왜 필요한가?

→ 만약 버퍼가 없다면, 데이터 전송 즉시 바로 사용하지 않으면 사라진다.

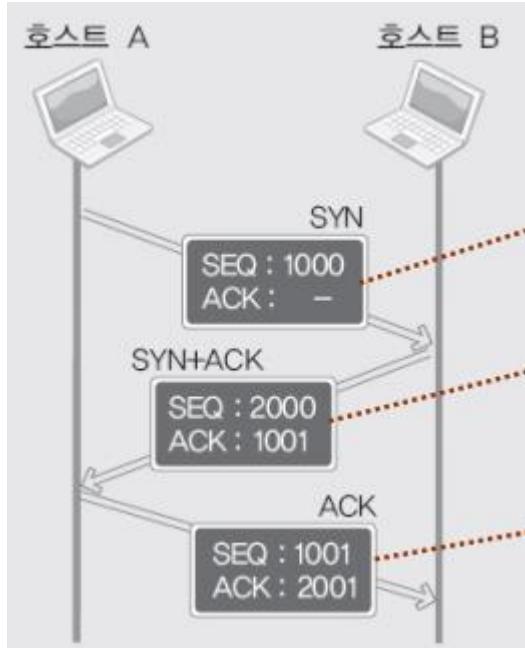
소켓을 닫아도 출력 버퍼에 남아있는 데이터는 계속해서 전송이 이뤄진다.



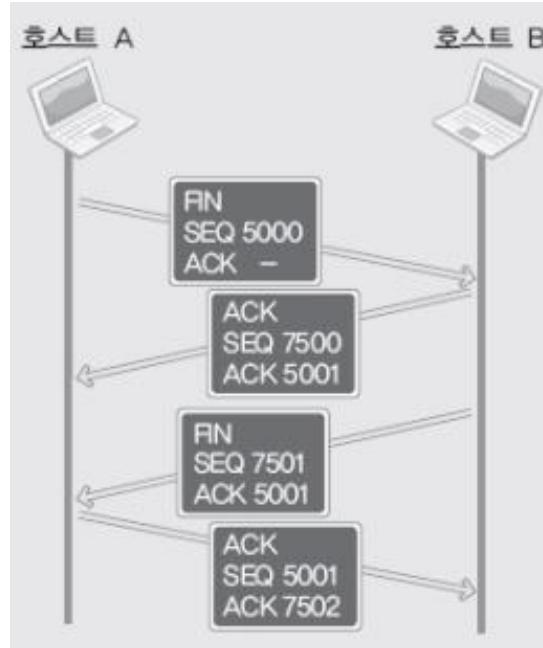
소켓을 닫으면 입력 버퍼에 남아있는 데이터는 소멸되어버린다.

TCP 연결 / 통신 / 종료

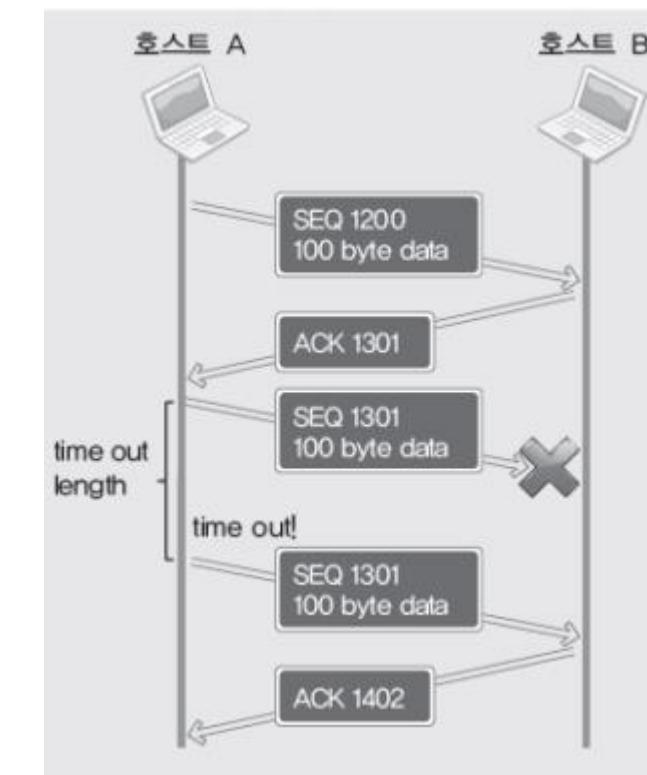
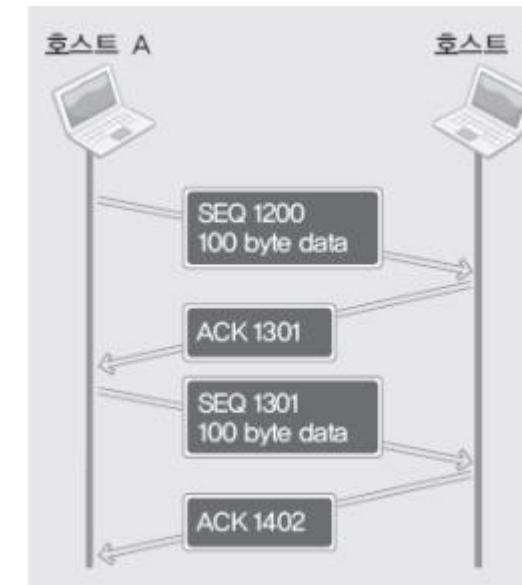
연결



종료



통신



ACK의 값을 전송된 바이트 크기만큼 증가시키는 이유는 패킷의 전송유무 뿐만 아니라, 데이터의 손실유무까지 확인하기 위함이다.

UDP

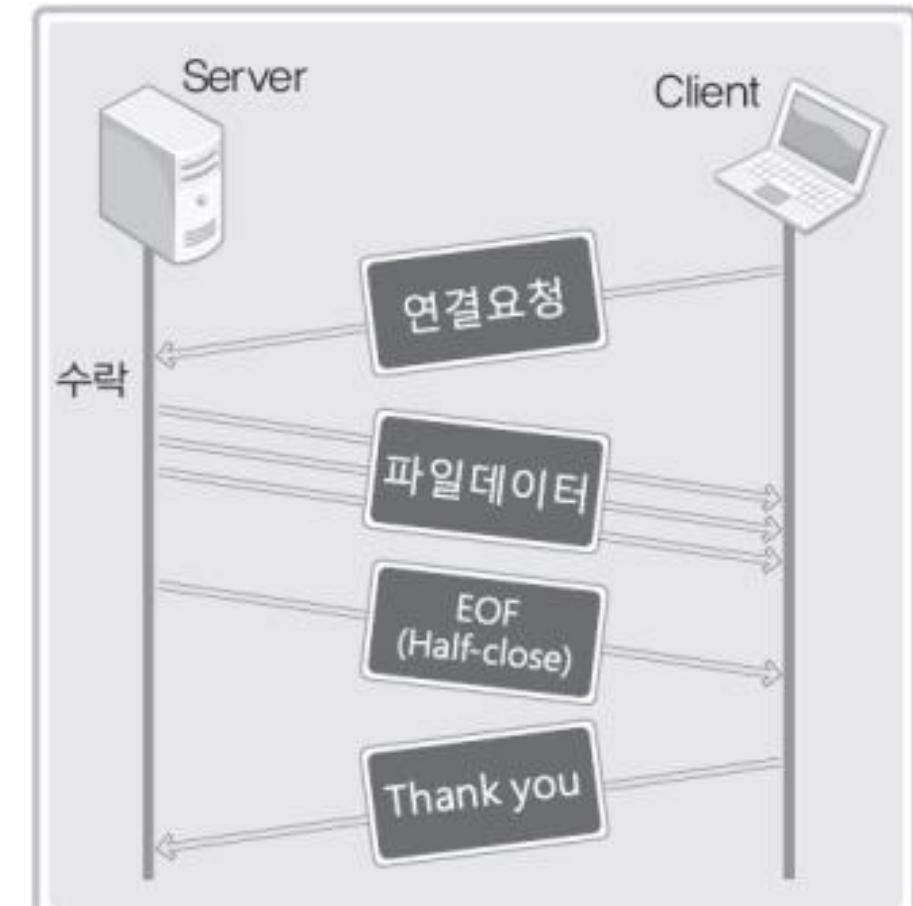
- 데이터의 경계가 존재한다.
→ 상대가 보낸 데이터 단위대로 읽는다.
- 연결 개념이 존재하지 않는다.
 - 소켓 하나로 여러 다른 소켓과 통신 가능
 - 연결이 없으므로, 전송 시마다 어디로 전송하는지 같이 보내주어야 한다.
 - 여러 목적지에 데이터 전송 가능

Half-Close

- 자신의 “출력 버퍼”만 종료한다. 왜냐하면 전송할 Data의 완료 여부는 확신할 수 있다.
- 하지만, 상대방이 전부 전송했는지는 모르기 때문에 입력버퍼를 함부로 종료할 수 없다.



Half-Close가 필요한 상황 예시



DNS 서버

- 인터넷에 연결된 네트워크 A는 DNS서버를 가지고 있을 필요는 없지만,
→ 물론, Localhost에 존재해도 되지만, 하는 일이 가벼우므로 하나를 공유
- 인터넷에 존재하는 DNS서버에 질의를 전송할 수는 있어야 한다.
- Default DNS 서버에 IP정보가 없으면 상위 DNS 서버에 질의 해서 변환

