8장. TCP 소켓 프로그래밍

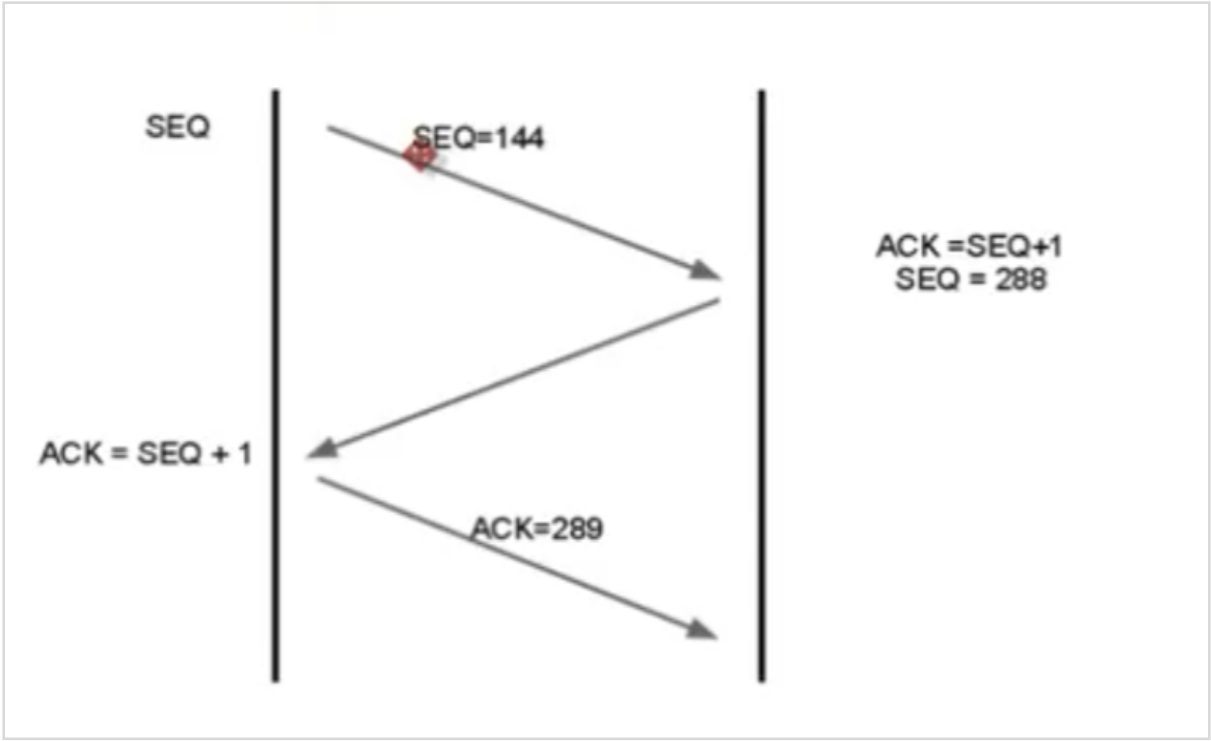
- 인터넷 : 프로그램과 프로그램의 연결
- 커뮤니케이션 방법 : TCP, UDP

TCP 통신을 쓰는 이유

- 패킷통신
- 경로와 회선상태의 다양함
  - 데이터 흐름을 제어하는 프로토콜 필요 = TCP가 이 역할을 함

TCP 프로토콜의 특징

- 연결 지향 프로토콜
- Three-way handshake
  - 세번의 패킷 교환으로 쌍방을 확인

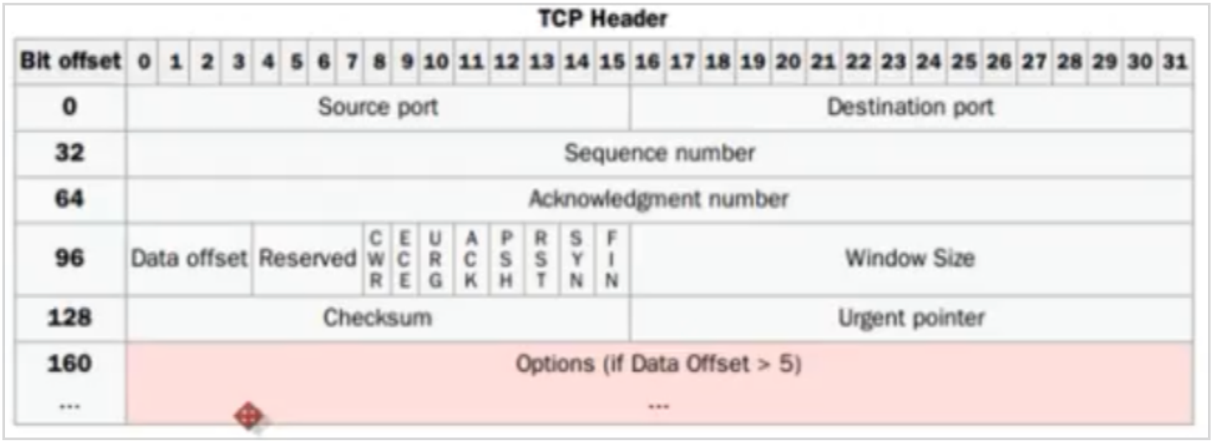


  - SEQ : 시퀀스 넘버
    - 연결을 요청한 측에서 정함
  - ACK : 시퀀스 패킷 + 1
    - 연결을 받은 쪽에서 +1 해서 보냄

➡ 이와 같이 상대 확인을 통해 신뢰성을 확보

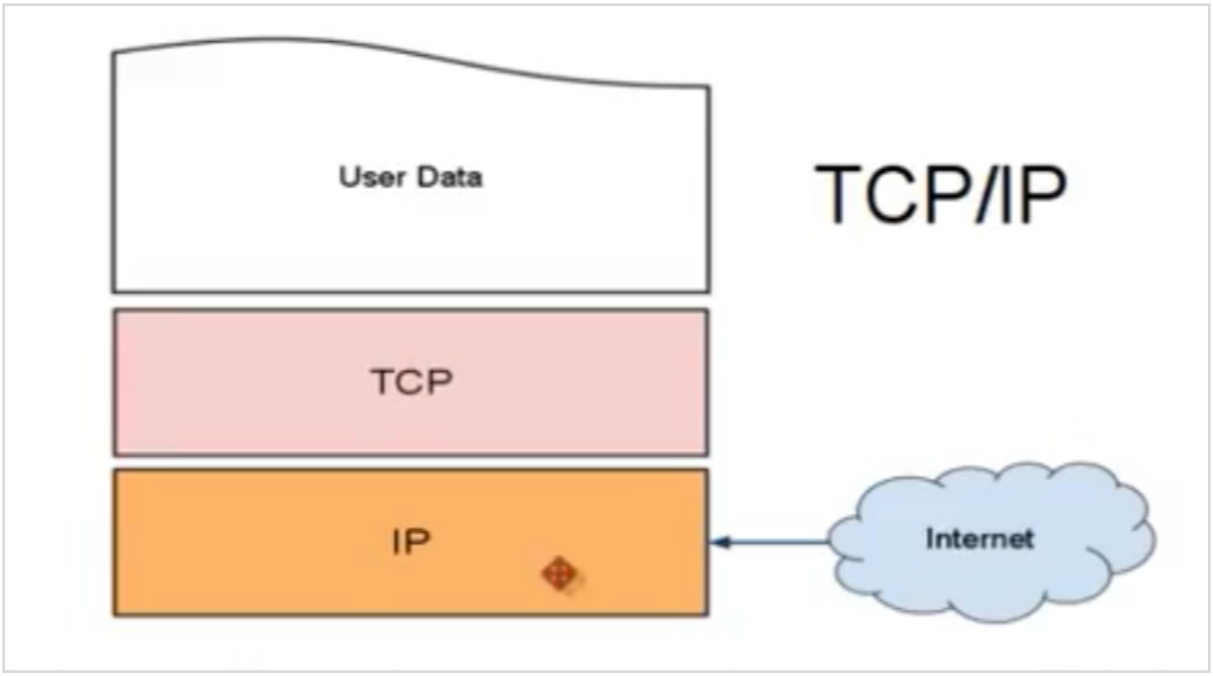
➡ 또한 시퀀스 번호를 통해 데이터 순서를 재조정 할 수 있다

TCP 헤더



- Source port, destination port
  - 데이터를 보낸 소프트웨어와 받는 소프트웨어 연결
- Sequence number
  - 패킷의 일련 번호를 준다
  - 이 데이터를 통해 나중에 순서를 재조정 할 수 있음
- Acknowledgment number
  - 응답번호
  - TCP는 모든 데이터에 대해서 응답 메세지를 보낸다
- Data offset
  - 유저 데이터가 시작되는 위치를 알려주는 데이터 오프셋
- 이후로는 패킷의 종류를 알려주기 위한 필드들..
- Window Size
- Checksum
  - 패킷의 잡음과 변조를 확인하기 위함

TCP의 경로 확인

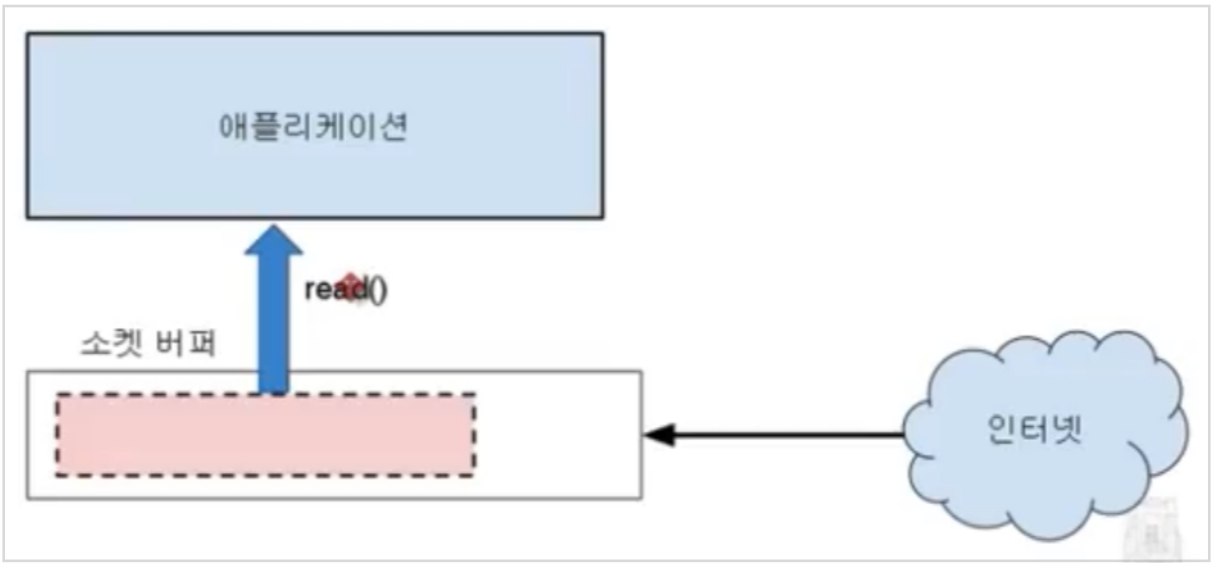


- IP가 경로를 가지고 있다 = 그래서 항상 TCP는 IP와 함께 있음
- IP로 컴퓨터의 위치를 찾고, TCP로 데이터의 흐름을 관리하고, 데이터를 처리할 소프트웨어를 찾음
  ➜ TCP/IP 프로토콜

소켓 버퍼

- 소켓은 읽기와 쓰기를 위한 보퍼를 따로 관리한다

소켓버퍼 - 읽기 예시



- 인터넷으로 부터 데이터가 쌓이면, 소켓 버퍼에 쌓이게 된다
- 입출력 함수를 이용해서 소켓 버퍼의 데이터를 유저 버퍼로 복사하게 된다.
  - 버퍼를 두는 이유는 프로그램이 실행되는 유저 영역과 소켓이 작용하는 커널 영역이 서로 분리되어있기 때문 ➜ 데이터 완충 지대 필요 = 버퍼

소켓 버퍼와 유저 버퍼가 분리되어 발생되는 문제

문제점

- 데이터를 얼마만큼 읽어야 할지를 모른다
- 왜냐면 데이터에는 경계가 명시되어있지 않다
  - 구조체 처럼 읽어야할 데이터 크기가 명확하면 몰라도 가변 데이터의 경우는 알 수 없다
  - 두번 데이터가 버퍼에 쌓였을 때 읽어야 한다고 하면 가변 데이터를 주고 받는 경우 이 기준을 정해야 한다

해결점

- 데이터의 크기 / 끝 부분을 표기

```
// EX HTTP 데이터
HTTP/1.1 200 OK
Date: Thu, 13 Jul 2011 10:10:10 GMT
Content-Length: 2291 // 데이터 크기 명시
Content-Type: text/html
```

TCP 소켓 프로그래밍 개발

TCP 속성의 소켓 생성

- 소켓이 TCP 기반으로 사용될 것이라고 명시

```
socket(AF_INET, SOCK_STREAM, 0);
```

서버 측 설정

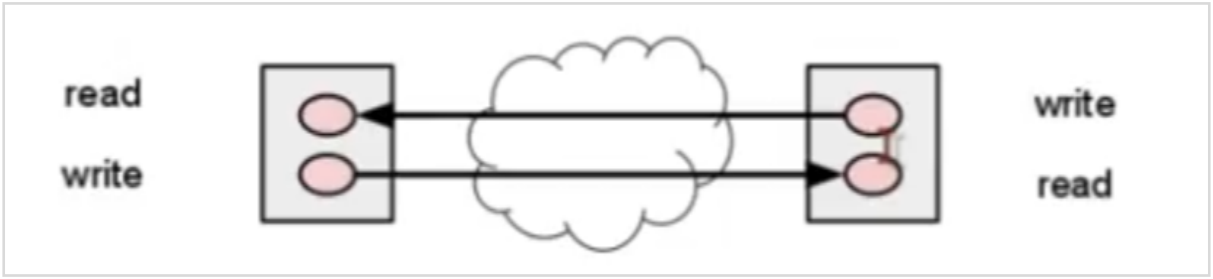
```
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = INADDR_ANY;
Addr.sin_port = htons(8080)
bind(sockfd, (struct sockaddr *)&addr, sizeof(addr));//만들어진 소켓의 주소와 포트번호를 묶어주기
```

클라이언트 측 설정

```
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = 연결할 인터넷 주소;
addr.sin_port = htons(8080);
```

```
connect(sockfd, (struct sockaddr *)&addr, sizeof(addr));
```

이후 데이터 통신은 TCP를 따른다



TCP 클라이언트 프로그램 - 에코 프로그램

- 과정
  - socket() ➜ connect() ➜ read() / write() ➜ close()

연결 시도

```
int sockfd;
struct sockaddr_in addr;
sockfd = socket(AF_INET, SOCK_STREAM, 0);
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = inet_addr("111.111.111.111");
addr.sin_port = htons(8888);
connect(sockfd, (struct sockaddr *)&addr, sizeof(addr));
```

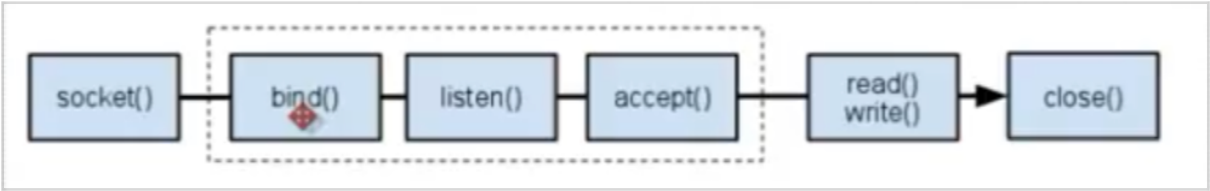
연결 성공 후

```
readn = read(0, buf, MAXLINE);//키보드로부터 입력을 읽음 (0은 표준 입력이라는 뜻)
write(sockfd, buf, readn);//키보드로부터 읽은 정보를 서버에 씀

memset(buf, 0x00, MAXLINE);//* memset함수는 어떤 메모리의 시작점부터 연속된 범위를 어떤 값으로(바이트 단위) 모두 지정하고 싶을 때 사용하는 함수
read(sockfd, buf, MAXLINE-1);//서버로부터 데이터를 읽음

close(sockfd);
```

TCP 서버 프로그램 - 에코 프로그램




```
int sockfd;
sockfd = socket(AF_INET, SOCK_STREAM, 0);
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = INADDR_ANY;
addr.sin_port = htons(8888);

bind(sockfd, (struct sockaddr *)&addr, sizeof(addr));//8888 포트에 묶음
listen(sockfd, 5);// 수신 대기열

for(;;) {
  clilen = sizeof(cliaddr);
  cli_sockfd = accept(addr, (struct sockaddr *)&cliaddr, &clilent); // 클라이언트에 연결 소켓을 받아서 clilen 소켓 (연결 소켓)을 만든다

  readn = read(cli_sockfd, buf, MAXLINE);//연결 소켓으로부터 데이터를 읽고, 버퍼에 복사
  write(cli_sockfd, buf, readn);// 복사한 내용을 클라이언트로 쓰며 됨
  close(cli_sockfd);
}
```

#Development/study