

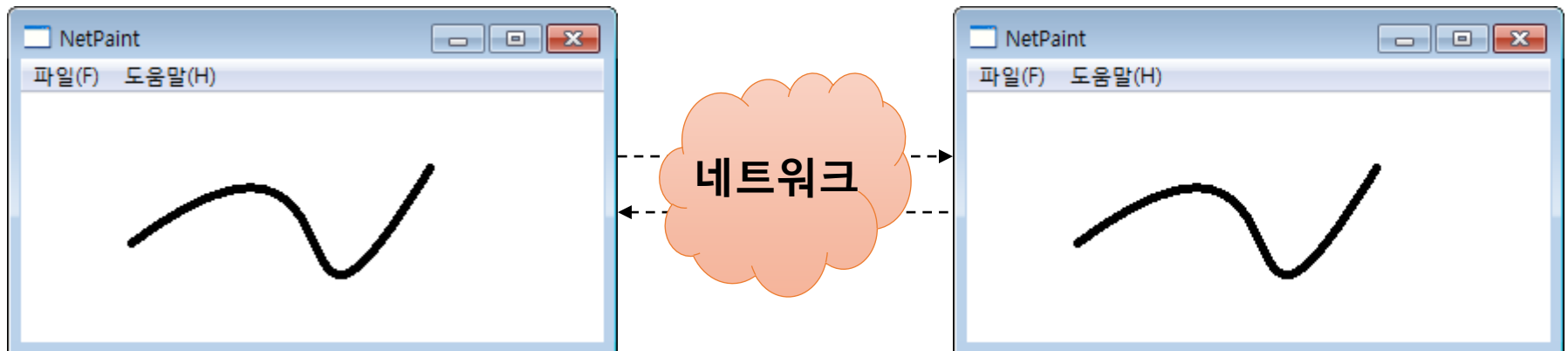
---

# 데이타전송하기

편집: 김혜영

# 응용 프로그램 프로토콜 (1)

- 응용 프로그램 프로토콜
  - 응용 프로그램 수준에서 주고받는 데이터의 형식과 의미 그리고 처리 방식을 정의한 프로토콜
- 응용 프로그램 프로토콜의 예
  - 네트워크 그림판 프로그램
    - 직선의 시작과 끝점
    - 선의 두께와 색상



# 응용 프로그램 프로토콜 (2)

---

- 메시지 정의 ①

```
struct DrawingMessage1
{
    int x1, y1;    // 직선의 시작점
    int x2, y2;    // 직선의 끝점
    int width;     // 선의 두께
    int color;     // 선의 색상
};
```

- 메시지 정의 ②

```
struct DrawingMessage2
{
    int x, y;      // 원의 중심 좌표
    int r;         // 원의 반지름
    int fillcolor; // 내부 채우기 색상
    int width;     // 테두리 두께
    int color;     // 테두리 색상
};
```

# 응용 프로그램 프로토콜 (3)

---

- 메시지 정의 ③

```
struct DrawingMessage1
{
    int type;        // = LINE
    int x1, y1;      // 직선의 시작점
    int x2, y2;      // 직선의 끝점
    int width;       // 선의 두께
    int color;       // 선의 색상
};

struct DrawingMessage2
{
    int type;        // = CIRCLE
    int x, y;        // 원의 중심 좌표
    int r;           // 원의 반지름
    int fillcolor;   // 내부 채우기 색상
    int width;       // 테두리 두께
    int color;       // 테두리 색상
};
```

# 데이터 전송 (1)

---

- 메시지 경계 구분 방법

- ① 송신자는 항상 고정 길이 데이터를 보냄. 수신자는 항상 고정 길이 데이터를 읽음
- ② 송신자는 가변 길이 데이터를 보내고 끝 부분에 특별한 표시(EOR, End Of Record)를 붙임. 수신자는 EOR이 나올 때까지 데이터를 읽음
- ③ 송신자는 보낼 데이터 크기를 고정 길이 데이터로 보내고, 이어서 가변 길이 데이터를 보냄. 수신자는 고정 길이 데이터를 읽어서 뒤따라올 가변 데이터의 길이를 알아내고, 이 길이만큼 데이터를 읽음
- ④ 송신자는 가변 길이 데이터 전송 후 접속을 정상 종료. 수신자는 `recv()` 함수의 리턴 값이 0(=정상 종료)이 될 때까지 데이터를 읽음

# 데이터 전송 (2)

---

- 메시지 경계 구분 방법

- 방법 ①

- 주고받을 데이터의 길이 변동폭이 크지 않을 경우에 적합

- 방법 ②

- 생성될 데이터의 길이를 미리 알 수 없을 때 적합

- 방법 ③

- 일반적으로 권장. 구현의 편의성과 처리 효율면에서 유리

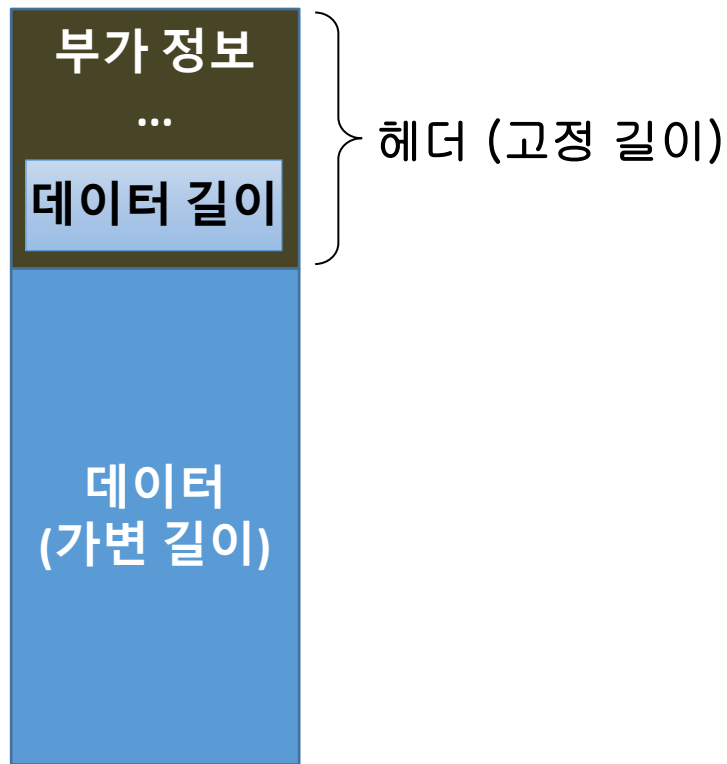
- 방법 ④

- 한쪽에서 일방적으로 데이터를 보내는 경우에 적합

# 데이터 전송 (3)

---

- 메시지 구조의 예 - 방법 ③을 사용할 경우



# 데이터 전송 (4)

---

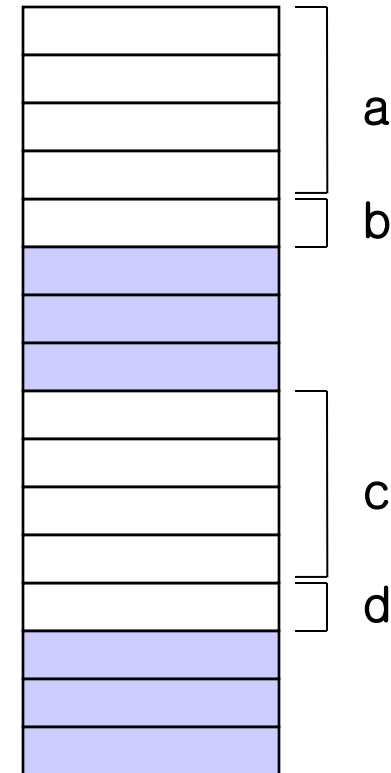
- 바이트 정렬
  - 빅 엔디안 방식으로 통일
- 구조체 멤버 맞춤
  - 구조체(C++의 클래스도 포함) 멤버의 메모리 시작 주소를 결정하는 컴파일러의 규칙
  - `#pragma pack` 지시자 사용



# 데이터 전송 (5)

- 구조체 멤버 맞춤의 예 - #pragma pack 적용 전

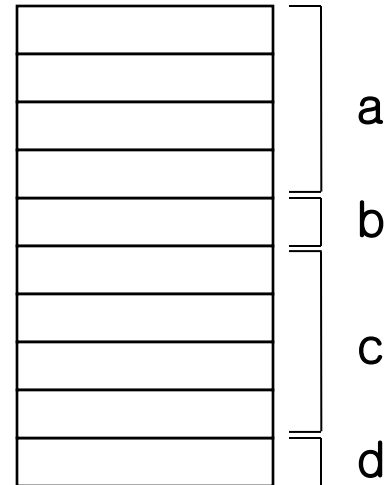
```
struct MyMessage
{
    int a;    // 4바이트
    char b;   // 1바이트
    int c;    // 4바이트
    char d;   // 1바이트
};
MyMessage msg;
...
send(sock, (char *)&msg, sizeof(msg), 0);
```



# 데이터 전송 (6)

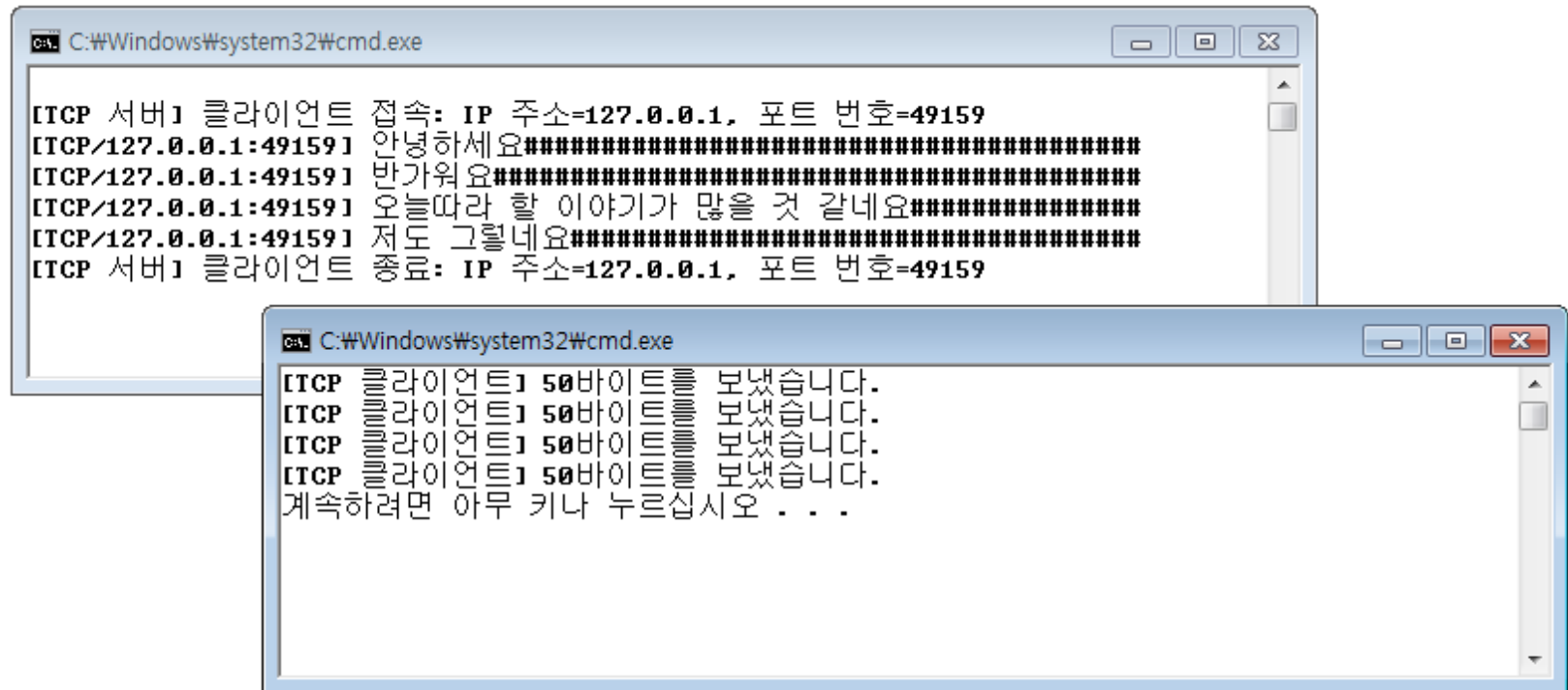
- 구조체 멤버 맞춤의 예 - #pragma pack 적용 후

```
#pragma pack(1)
struct MyMessage
{
    int a;    // 4바이트
    char b;   // 1바이트
    int c;    // 4바이트
    char d;   // 1바이트
};
#pragma pack()
MyMessage msg;
...
send(sock, (char *)&msg, sizeof(msg), 0);
```



# 다양한 데이터 전송 방식 (1)

- 고정 길이 데이터 전송
  - 서버와 클라이언트 모두 크기가 같은 버퍼를 정의해두고 데이터를 주고받음



```
C:\Windows\system32\cmd.exe

[TCPIP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=49159
[TCPIP/127.0.0.1:49159] 안녕하세요#####
[TCPIP/127.0.0.1:49159] 반가워요#####
[TCPIP/127.0.0.1:49159] 오늘따라 할 이야기가 많을 것 같네요#####
[TCPIP/127.0.0.1:49159] 저도 그렇네요#####
[TCPIP 서버] 클라이언트 종료: IP 주소=127.0.0.1, 포트 번호=49159

C:\Windows\system32\cmd.exe

[TCPIP 클라이언트] 50바이트를 보냈습니다.
[TCPIP 클라이언트] 50바이트를 보냈습니다.
[TCPIP 클라이언트] 50바이트를 보냈습니다.
[TCPIP 클라이언트] 50바이트를 보냈습니다.
계속하려면 아무 키나 누르십시오 . . .
```

# 다양한 데이터 전송 방식 (2)

- 가변 길이 데이터 전송
  - 가변 길이 데이터 경계를 구분하기 위해 EOR로 사용할 데이터 패턴을 정해야 함
    - 흔히 '\n'이나 '\r\n'을 사용
  - 예) '\n'을 검출하는 가상 코드

```
while(1){
```

```
    소켓 수신 버퍼에서 1바이트 데이터를 읽는다.  
    읽은 데이터가 '\n'이 아니면 응용 프로그램 버퍼에 저장한다.  
    읽은 데이터가 '\n'이면 루프를 빠져나온다.
```

```
}
```

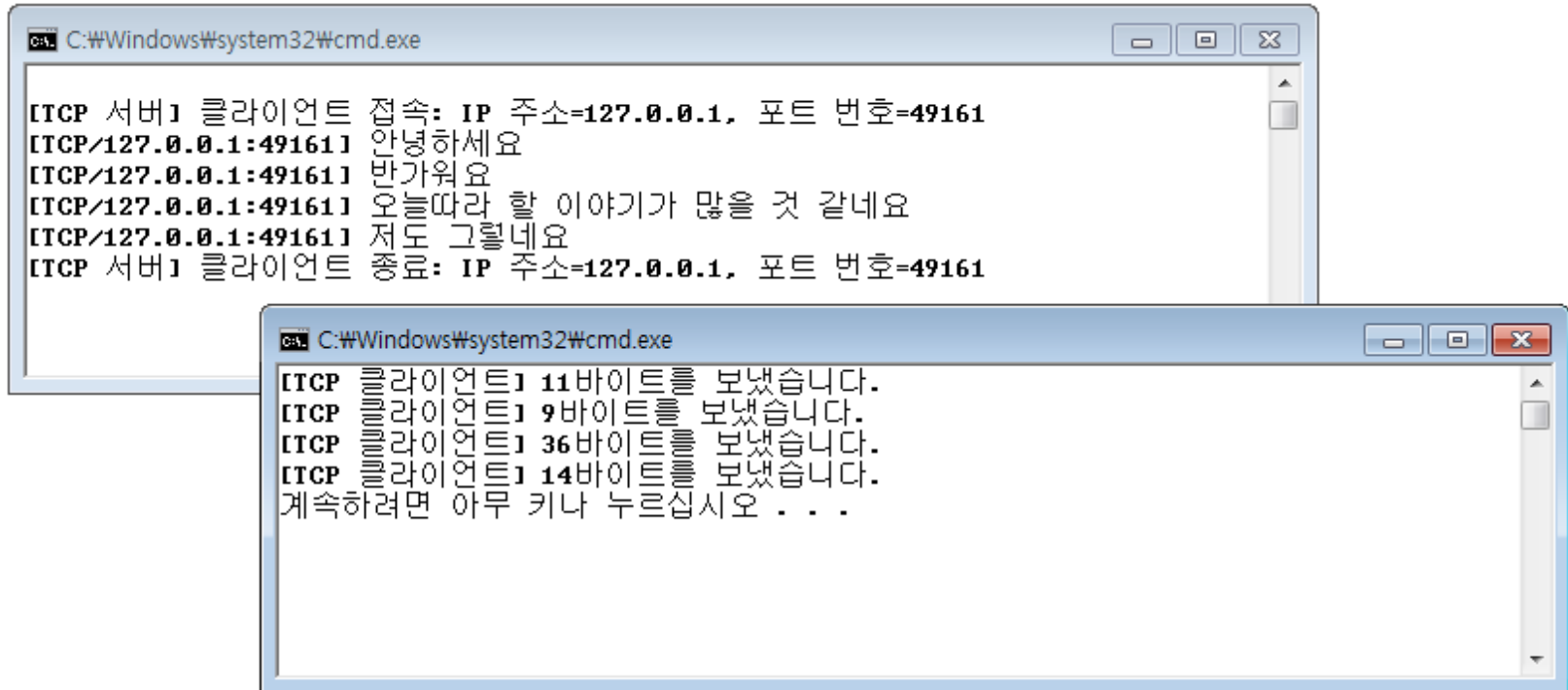
```
응용 프로그램 버퍼에 저장된 데이터를 사용한다.
```

성능 저하 요인!

- 소켓 수신 버퍼에서 데이터를 한 번에 많이 읽어 1바이트씩 리턴해주는 사용자 정의 함수가 필요!

# 다양한 데이터 전송 방식 (3)

- 가변 길이 데이터 전송



```
C:\Windows\system32\cmd.exe

[TCPP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=49161
[TCPP/127.0.0.1:49161] 안녕하세요
[TCPP/127.0.0.1:49161] 반가워요
[TCPP/127.0.0.1:49161] 오늘따라 할 이야기가 많을 것 같네요
[TCPP/127.0.0.1:49161] 저도 그럴네요
[TCPP 서버] 클라이언트 종료: IP 주소=127.0.0.1, 포트 번호=49161

C:\Windows\system32\cmd.exe

[TCPP 클라이언트] 11바이트를 보냈습니다.
[TCPP 클라이언트] 9바이트를 보냈습니다.
[TCPP 클라이언트] 36바이트를 보냈습니다.
[TCPP 클라이언트] 14바이트를 보냈습니다.
계속하려면 아무 키나 누르십시오 . . .
```

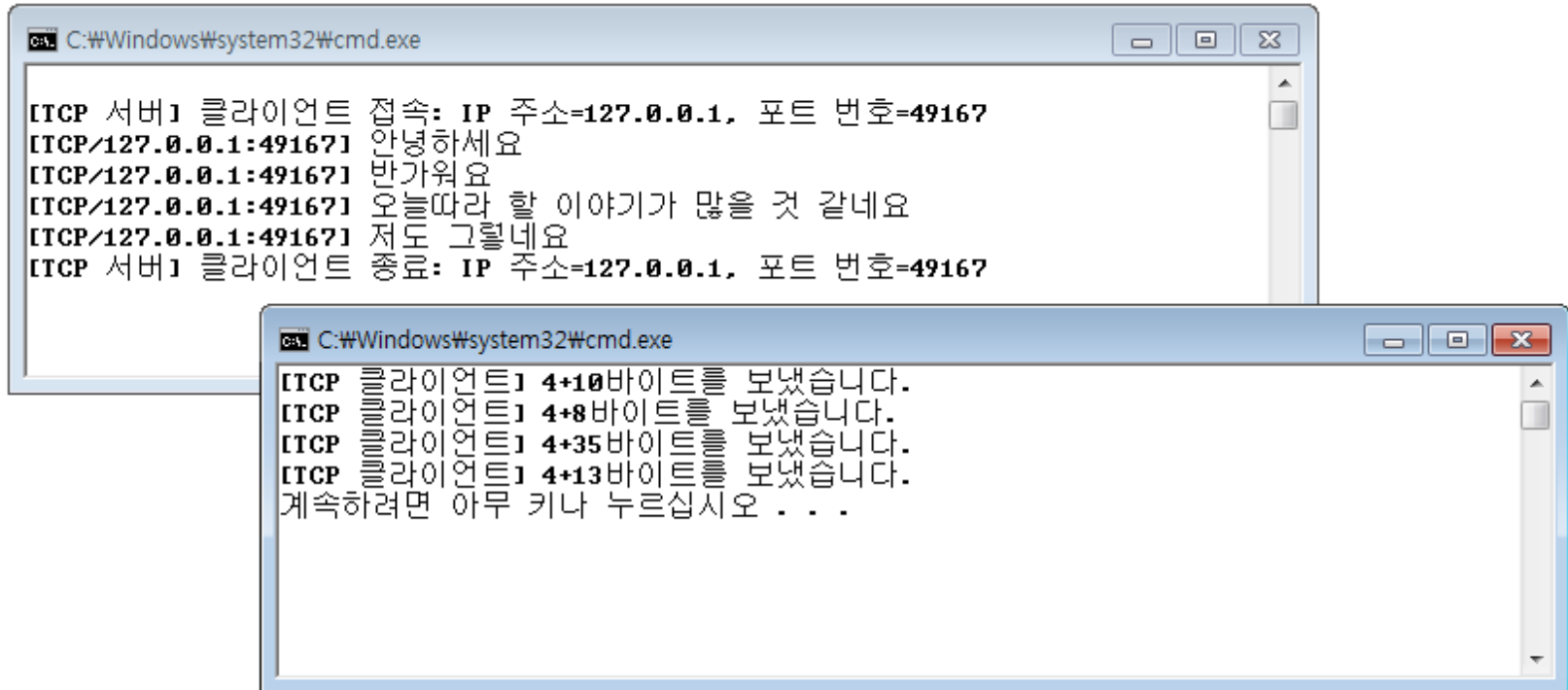
# 다양한 데이터 전송 방식 (4)

---

- 고정 길이+가변 길이 데이터 전송
  - 송신 측에서 가변 길이 데이터의 크기를 곧바로 계산할 수 있다면 고정 길이 + 가변 길이 전송이 효과적
  - 수신 측에서는 [① 고정 길이 데이터 수신 ② 가변 길이 데이터 수신] 두 번의 데이터 수신으로 가변 길이 데이터의 경계를 구분해 읽을 수 있음

# 다양한 데이터 전송 방식 (5)

- 고정 길이+가변 길이 데이터 전송



```
C:\Windows\system32\cmd.exe

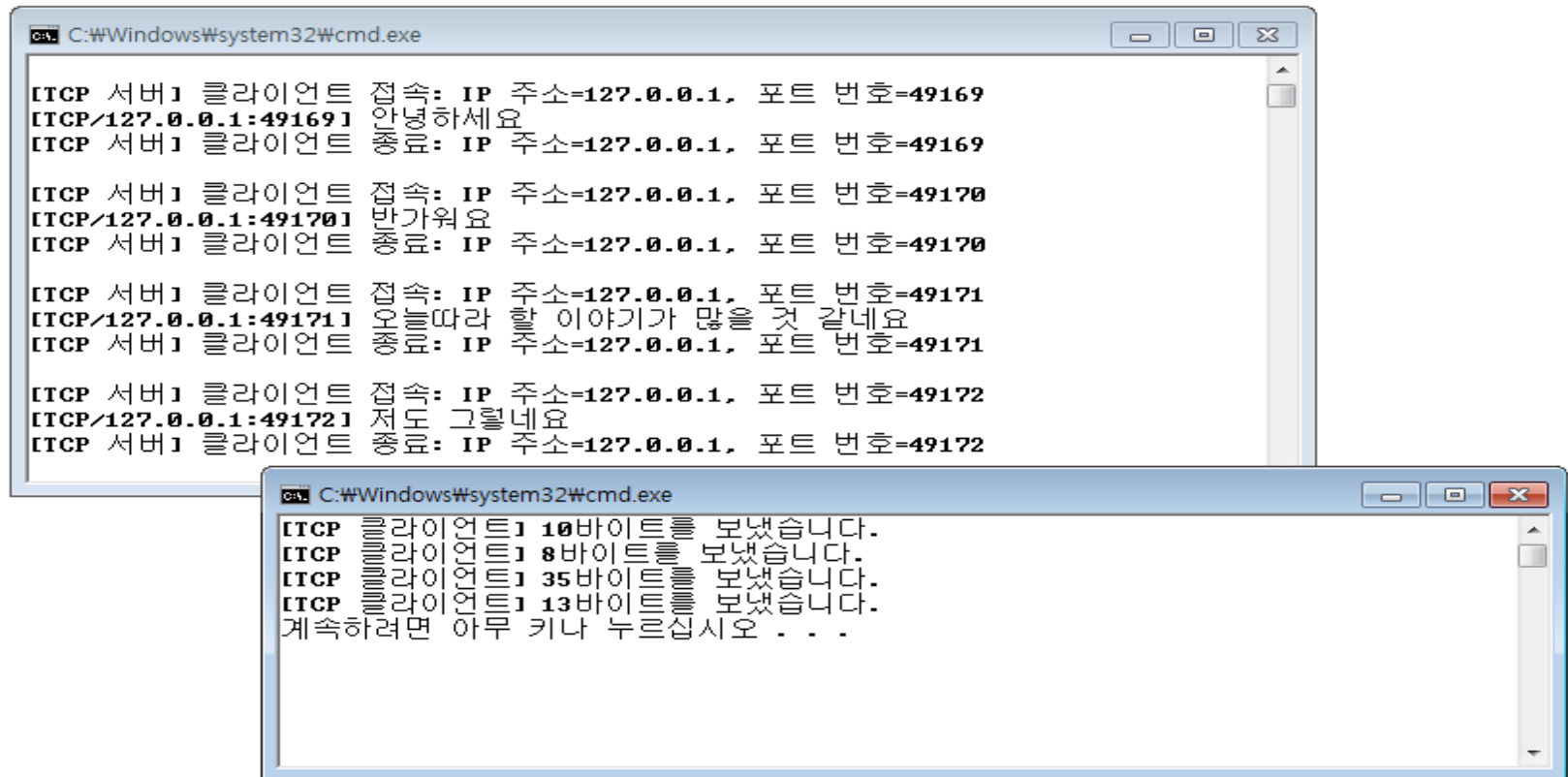
[ITCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=49167
[ITCP/127.0.0.1:49167] 안녕하세요
[ITCP/127.0.0.1:49167] 반가워요
[ITCP/127.0.0.1:49167] 오늘따라 할 이야기가 많을 것 같네요
[ITCP/127.0.0.1:49167] 저도 그럴네요
[ITCP 서버] 클라이언트 종료: IP 주소=127.0.0.1, 포트 번호=49167

C:\Windows\system32\cmd.exe

[ITCP 클라이언트] 4+10바이트를 보냈습니다.
[ITCP 클라이언트] 4+8바이트를 보냈습니다.
[ITCP 클라이언트] 4+35바이트를 보냈습니다.
[ITCP 클라이언트] 4+13바이트를 보냈습니다.
계속하려면 아무 키나 누르십시오 . . .
```

# 다양한 데이터 전송 방식 (6)

- 데이터 전송 후 종료
  - 일종의 가변 길이 데이터 전송 방식
    - EOR로 특별한 데이터 패턴 대신 연결 종료를 사용



```
C:\Windows\system32\cmd.exe

[ITCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=49169
[ITCP/127.0.0.1:49169] 안녕하세요
[ITCP 서버] 클라이언트 종료: IP 주소=127.0.0.1, 포트 번호=49169

[ITCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=49170
[ITCP/127.0.0.1:49170] 반가워요
[ITCP 서버] 클라이언트 종료: IP 주소=127.0.0.1, 포트 번호=49170

[ITCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=49171
[ITCP/127.0.0.1:49171] 오늘따라 할 이야기가 많을 것 같네요
[ITCP 서버] 클라이언트 종료: IP 주소=127.0.0.1, 포트 번호=49171

[ITCP 서버] 클라이언트 접속: IP 주소=127.0.0.1, 포트 번호=49172
[ITCP/127.0.0.1:49172] 저도 그럴네요
[ITCP 서버] 클라이언트 종료: IP 주소=127.0.0.1, 포트 번호=49172

C:\Windows\system32\cmd.exe

[ITCP 클라이언트] 10바이트를 보냈습니다.
[ITCP 클라이언트] 8바이트를 보냈습니다.
[ITCP 클라이언트] 35바이트를 보냈습니다.
[ITCP 클라이언트] 13바이트를 보냈습니다.
계속하려면 아무 키나 누르십시오 . . .
```



# 실습 1

---

- 강의록에 있는 TCPServer\_Fixed.cpp 와 TCPClient\_Fixed.cpp 분석하기

# 실습

---

- 본인의 기본 서버에 다음의 데이터 구조로 패킷을 만들어 데이터를 송수신하시오.

```
int id; //use id
```

```
int x;
```

```
int y;
```

```
int z;
```

```
char *message [ ] = {"hellow"} //가변길이로 처리
```

- 클라이언트에서 빅엔디안 형식으로 임의의 x,y,z 위치값과 id값, 그리고 읽어들이는 \*message 값을 서버에 보내고 서버에서는 수신한 내용을 화면에 출력하시오.