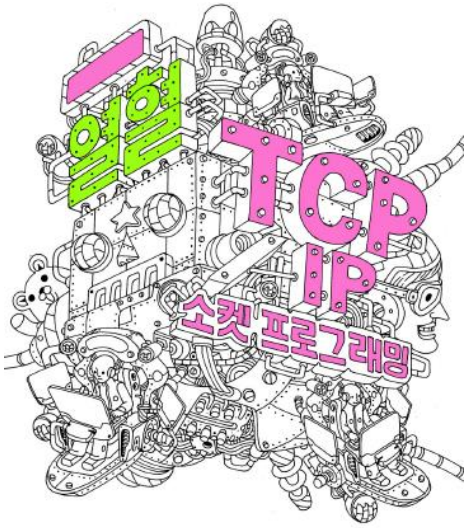
윤성우의 열혈 TCP/IP 소켓 프로그래밍

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

Chapter 05. TCP 기반 서버 / 클라이언트 2



Chapter 05-1. 에코 클라이언트의 완벽 구현

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

에코 클라이언트의 문제점 확인하기

에코 서버의 코드

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

서버는 데이터의 경계를 구분하지 않고 수신된 데이터를 그대로 전송할 의무만 갖는다. TCP는 데이터 경계가 없는 프로토콜이므로, 두 번이나 세 번의 write 함수 호출을 통해서 데이터를 전송하더라도 문제 되지 않는다.

에코 클라이언트의 코드

```
write(sock, message, strlen(message));
str_len=read(sock, message, BUF_SIZE-1);
```

반면, 클라이언트는 문장 단위로 데이터를 송수신하기 때문에, 데이터의 경계를 구분해야 한다. 때문에 이와 같은 데이터 송수신 방식은 문제가 된다. TCP의 read 및 write 함수 호출은 데이터의 경계를 구분하지 않기 때문이다.

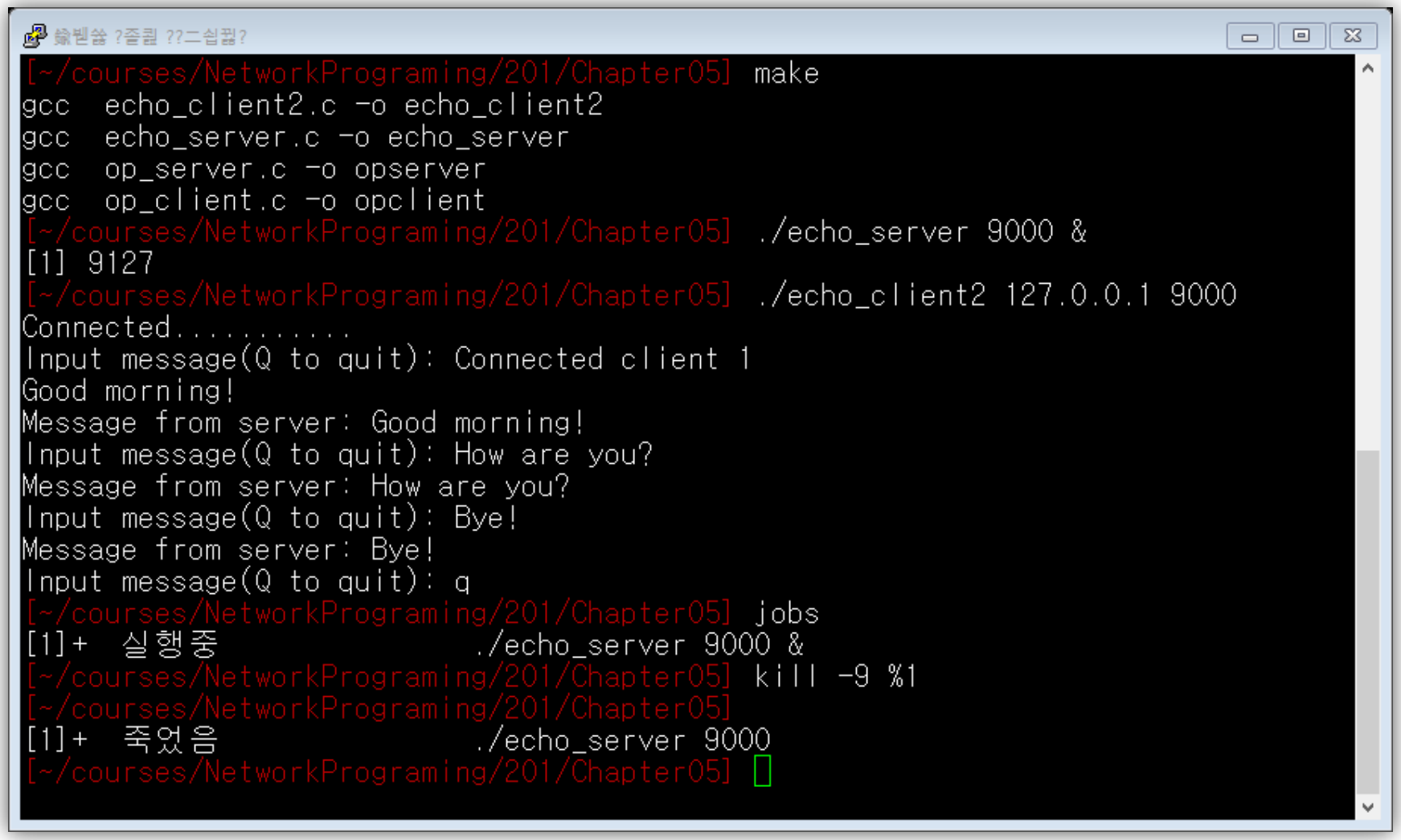
에코 클라이언트2의 해결책!

```
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;
}
message[recv_len]=0;
printf("Message from server: %s", message);
```

write 함수 호출할 때 전송한 데이터 길이만큼 읽어 들일 수 있는 반복문의 삽입이 필요하다. 이것이 TCP를 기반으로 데이터를 구분지어 읽어 들이는데 부가적으로 필요한 구분이다.

에코 클라이언트2 실행 결과 → 차이를 알수 있나요?



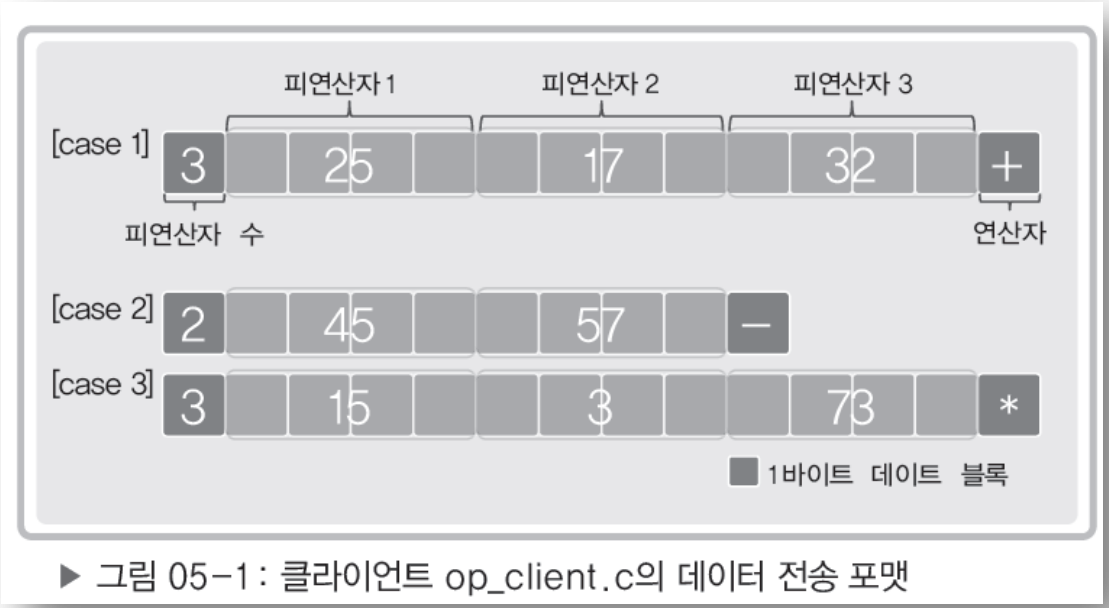
계산기 프로그램 구현하기(어플리케이션 프로토콜)

서버는 클라이언트로부터 여러 개의 숫자와 연산자 정보를 전달받는다. 그러면 서버는 전달받은 숫자를 바탕으로 덧셈, 뺄셈 또는 곱셈을 계산해서 그 결과를 클라이언트에게 전달한다. 예를 들어서 서버로 3, 5, 9가 전달되고 덧셈연산이 요청된다면 클라이언트에는 3+5+9의 연산결과가 전달되어야 하고, 곱셈연산이 요청된다면 클라이언트에는 3×5×9의 연산결과가 전달되어야 한다. 단, 서버로 4, 3, 2가 전달되고 뺄셈연산이 요청되면 클라이언트에는 4-3-2의 연산결과가 전달되어야 한다. 즉, 뺄셈의 경우에는 첫 번째 정수를 대상으로 뺄셈이 진행되어야 한다.

이와 같은 서버 클라이언트 사이에서의 데이터 송수신 명세가 바로 응용프로그램 프로토콜이다!

- 클라이언트는 서버에 접속하자마자 피연산자의 개수정보를 1바이트 정수형태로 전달한다.
- 클라이언트가 서버에 전달하는 정수 하나는 4바이트로 표현한다.
- 정수를 전달한 다음에는 연산의 종류를 전달한다. 연산정보는 1바이트로 전달한다.
- 문자 +, −, * 중 하나를 선택해서 전달한다.
- 서버는 연산결과를 4바이트 정수의 형태로 클라이언트에게 전달한다.
- 연산결과를 얻은 클라이언트는 서버와의 연결을 종료한다.

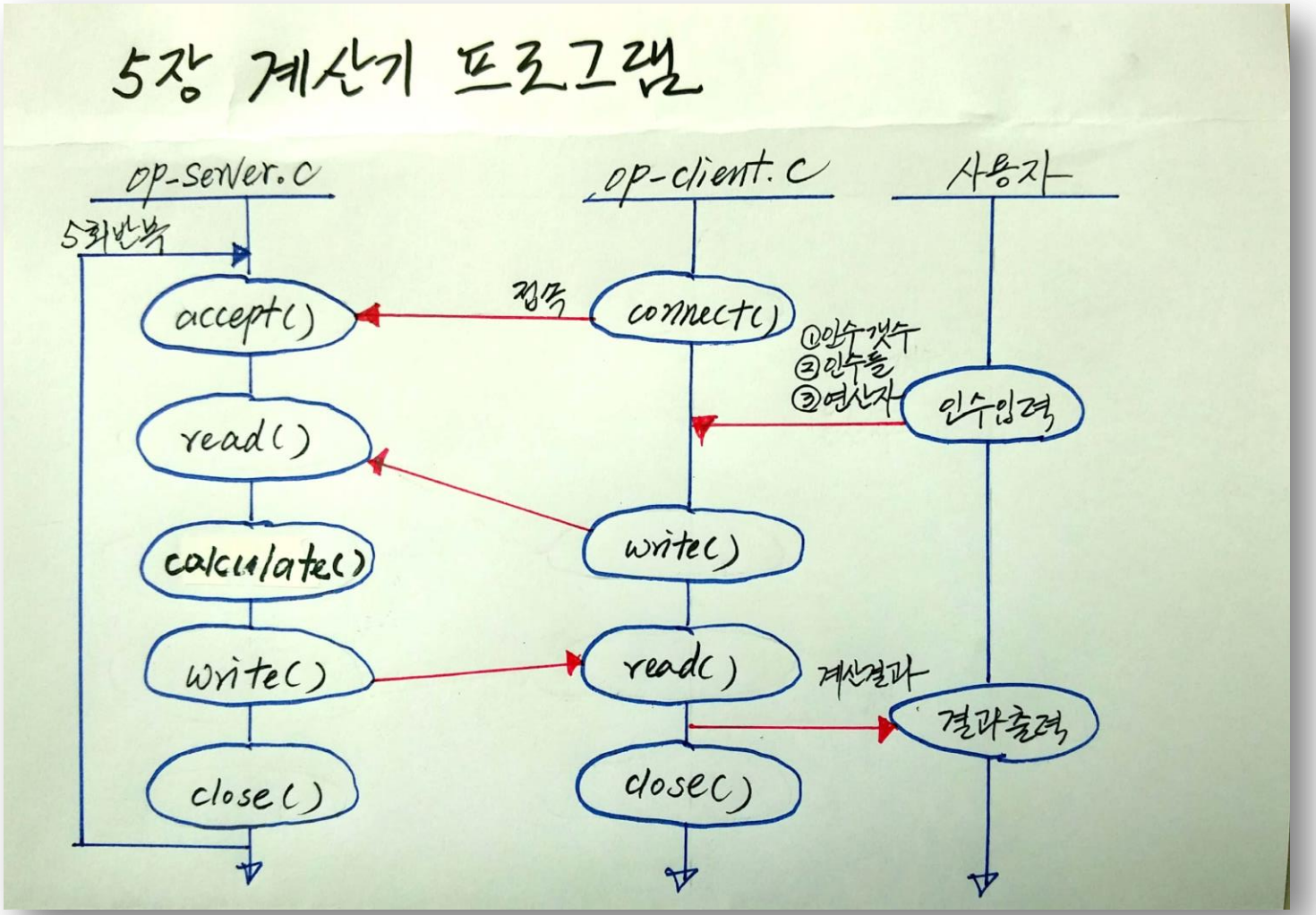
서버, 클라이언트의 구현(어플리케이션 프로토콜)



▶ 그림 05-1 : 클라이언트 op_client.c의 데이터 전송 포맷

프로토콜은 위와 같이(그 이상으로) 명확히 정의해야 한다.

서버, 클라이언트 동작


5장 계산기 프로그램
op-server.c
op-client.c
사용자
5회반복
accept()
접속
connect()
①인수갯수
②인수들
③연산자
인수입력
read()
write()
calculate()
write()
read()
계산결과
결과출력
close()
close()

서버, 클라이언트 컴파일 및 실행 화면

```
$ make
gcc  op_server.c -o opserver
gcc  op_client.c -o opclient
$ ./opserver 9190 &
[1] 13521
$ ./opclient 127.0.0.1 9190
Connected.....
Operand count: 3
Operand 1: 12
Operand 2: 24
Operand 3: 36
Operator: +
Operation result: 72
$ kill -9 1351
```

서버 코드 검토

- op_server.c

```
41    for(i=0; i<5; i++)          // 5명의 클라이언트 순차 접속과 처리 가능
42    {
43        opnd_cnt=0;
44        clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_adr,  &clnt_adr_sz);
45        read(clnt_sock,  &opnd_cnt, 1);
46
47        recv_len=0;
48        while((opnd_cnt*OPSZ+1)>recv_len)
49        {
50            recv_cnt=read(clnt_sock, &opinfo[recv_len], BUF_SIZE-1);
51            recv_len+=recv_cnt;
52        }
53        result=calculate(opnd_cnt, (int*)opinfo, opinfo[recv_len-1]);
54        write(clnt_sock, (char*)&result, sizeof(result));
55        close(clnt_sock);
56    }
```

서버 코드 검토

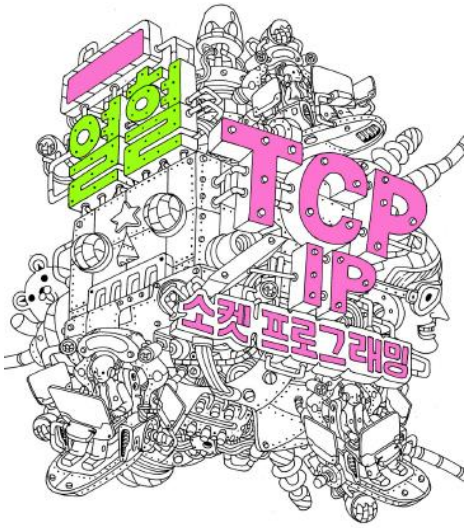
- op_server.c

```
61 int calculate(int opnum, int opnds[], char op)
62 {
63     int result=opnds[0], i;
64
65     switch(op)
66     {
67     case '+':
68         for(i=1; i<opnum; i++) result+=opnds[i];
69         break;
70     case '-':
71         for(i=1; i<opnum; i++) result-=opnds[i];
72         break;
73     case '*':
74         for(i=1; i<opnum; i++) result*=opnds[i];
75         break;
76     }
77     return result;
78 }
```

클라이언트 코드 검토

- op_client.c

```
38    fputs("Operand count: ", stdout);
39    scanf("%d", &opnd_cnt);
40    opmsg[0]=(char)opnd_cnt;
41
42    for(i=0; i<opnd_cnt; i++)
43    {
44        printf("Operand %d: ", i+1);
45        scanf("%d", (int*)&opmsg[i*OPSZ+1]);
46    }
47    fgetc(stdin);
48    fputs("Operator: ", stdout);
49    scanf("%c", &opmsg[opnd_cnt*OPSZ+1]);
50    write(sock, opmsg, opnd_cnt*OPSZ+2);
51    read(sock, &result, RLT_SIZE);
52
53    printf("Operation result: %d \n", result);
```



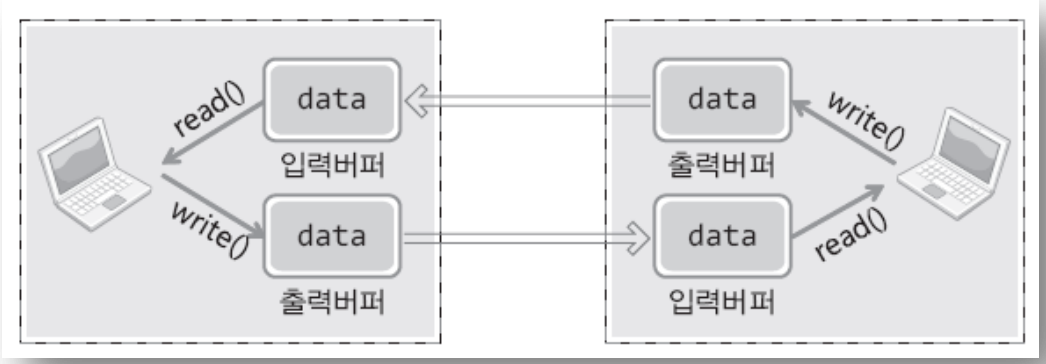
Chapter 05-2. TCP의 이론적인 이야기

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판

TCP 내부 동작원리 1. : 소켓에 존재하는 입출력 버퍼

- 입출력 버퍼는 소켓 생성 시 자동으로 생성된다.
- 입출력 버퍼는 TCP 소켓과 연결되어 각각 존재한다.
- 소켓을 닫아도 출력버퍼에 남아있는 데이터는 계속해서 전송이 이뤄진다.
- 소켓을 닫으면 입력버퍼에 남아있는 데이터는 소멸된다.

별개의 버퍼가 존재하기 때문에 데이터의 슬라이딩 윈도우 프로토콜 적용이 가능하고, 이로 인해서 버퍼가 차고 넘치는 상황은 발생하지 않는다.



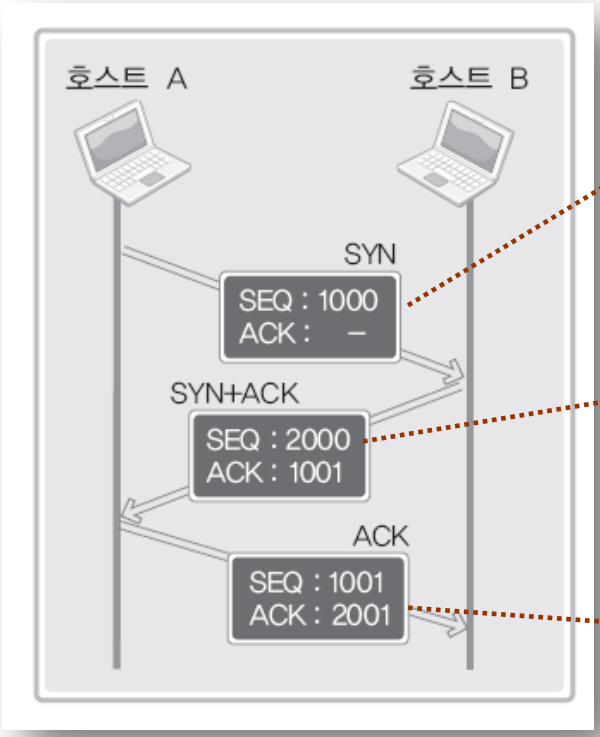
소켓 A : 야 50바이트까지는 보내도 괜찮아!
소켓 B : OK!

소켓 A : 내가 20바이트 비웠으니까 70바이트까지 괜찮아!
소켓 B : OK!

슬라이딩 윈도우 프로토콜의 데이터 송수신 절차

TCP 내부 동작원리 2. : 상대 소켓과의 연결

- [Shake 1] 소켓 A Hi! 소켓 B, 내가 전달할 데이터가 있으니 우리 연결 좀 하자.
- [Shake 2] 소켓 B Okay! 지금 나도 준비가 되었으니 언제든지 시작해도 좋다.
- [Shake 3] 소켓 A Thank you! 내 요청을 들어줘서 고맙다.



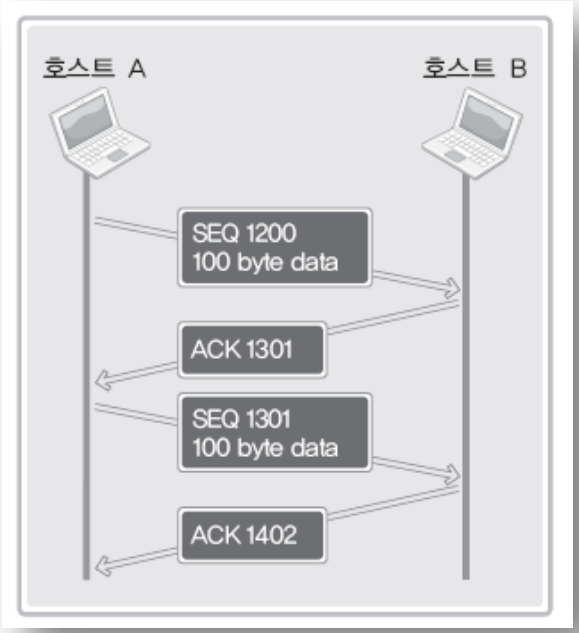
3-Way Handshake

"제가 지금 보내는 이 패킷에 1000번이라는 번호를 부여하니, 잘 받았다면 다음 패킷인 1001번 패킷을 전송해 달라고 내게 말해주세요."

"당신이 보내 준 1000번 패킷 잘 받았으니 다음 1001번 패킷 보내주세요. 그리고 제가 지금 보내는 이 패킷에 2000이라는 번호를 부여하니, 잘 받았다면 다음 패킷인 2001번 패킷을 전송해 달라고 내게 말해주세요."

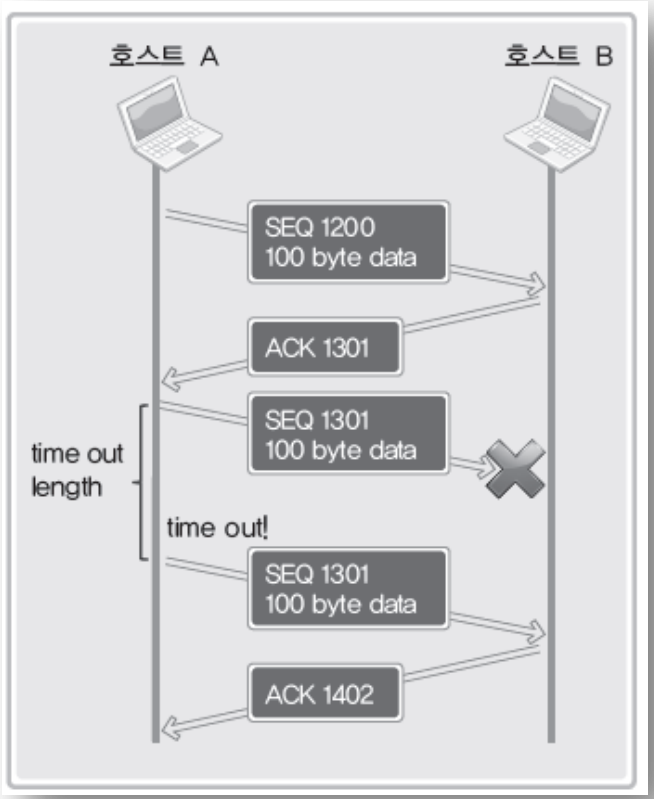
"당신이 보내 준 2000번 패킷 잘 받았으니 다음의 2001번 패킷 보내주세요. 그리고 좀 전에 제가 전송한 SEQ가 1000인 패킷도 잘 받았다고 하니, 이번에는 SEQ가 1001인 다음 패킷을 전송합니다."

TCP 내부 동작원리 3. : 상대 소켓과 데이터 송수신



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

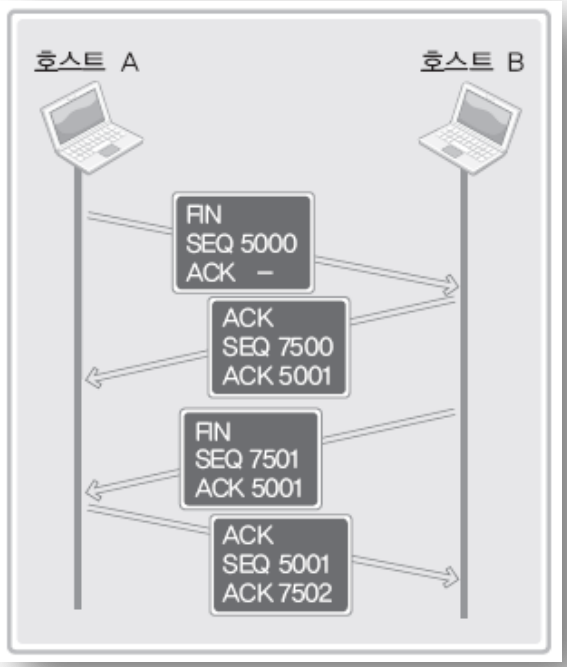
ACK 번호 → SEQ 번호 + 전송된 바이트 크기 + 1



SEQ 전송 시 송신측에서 타이머 작동, 그리고 SEQ에 대한 ACK가 설정된 시간 이내에 수신되지 않을 경우 데이터 재전송

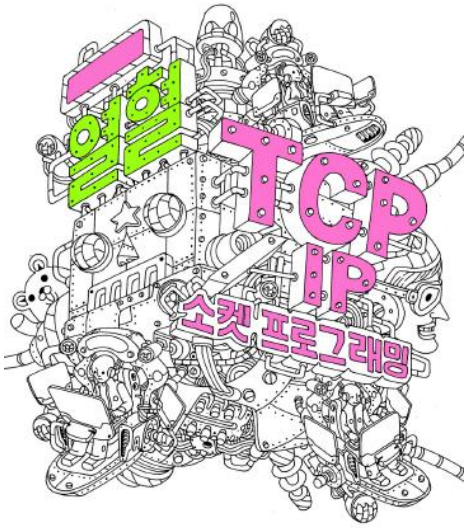
TCP 내부 동작원리 4. : 상대 소켓과의 연결 종료

- 소켓 A　　전 연결을 끊고자 합니다.
- 소켓 B　　아! 그러세요? 잠시만 기다리세요.
- 소켓 B　　네 저도 준비가 끝났습니다. 그럼 연결을 끊으시지요.
- 소켓 A　　네! 그 동안 즐거웠습니다.



four-way handshaking 과정을 거쳐서 연결을 종료하는 이유는 일방적 종료로 인한 데이터의 손실을 막기 위함이다.

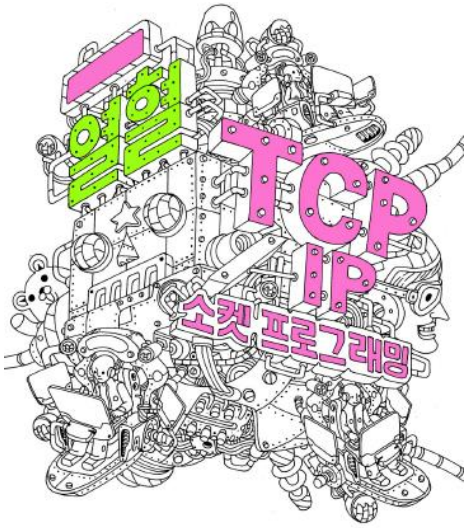
- 신뢰성있는 데이터 송수신을 위한 TCP 패킷 전송 과정의 상세 내용은 3학년 2학기 컴퓨터 네트워크 과목에서 공부하세요!!!
- 이런 내용은 네트워크 프로그래밍 공부할 때 알면 좋고 몰라도 크게 문제되지 않습니다!!!

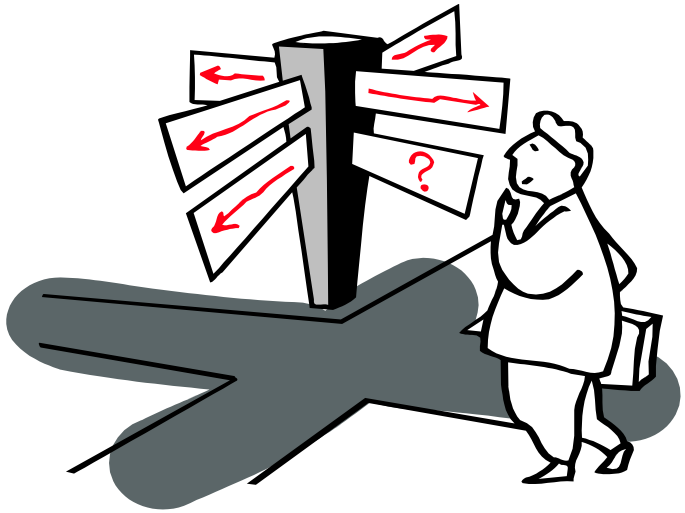


op_server_win.c와 op_client_win.c는
윈도우용 코드로서 별도 강의를 생략합니다.

Chapter 05-3. 윈도우 기반으로 구현하기

윤성우 저 열혈강의 TCP/IP 소켓 프로그래밍 개정판





Chapter 05가 끝났습니다. 질문 있으신지요?