



인하공업전문대학
INHA TECHNICAL COLLEGE

TCP/IP 네트워크 프로그래밍 9주차

인하공업전문대학 컴퓨터 정보과
김한결 강사

- 복습
 - ✓ inet_addr
 - ✓ inet_aton
 - ✓ inet_ntoa
 - ✓ 인터넷주소
 - ✓ INADDR_ANY
- 소켓 인터넷주소할당
- Iterative 서버의 구현
- 에코 서버 클라이언트 구현
- TCP 소켓에 존재하는 입출력 버퍼
- 상대소켓과의 연결
- 어플리케이션 프로토콜
- 소켓의 Half-close

바이트 변환의 예

```
~/socket_programming
1 #include <stdio.h>
2 #include <arpa/inet.h>
3
4 int main(int argc, char *argv[])
5 {
6     unsigned short host_port=0x1234;
7     unsigned short net_port;
8     unsigned long host_addr=0x12345678;
9     unsigned long net_addr;
10
11     net_port=htons(host_port);
12     net_addr=htonl(host_addr);
13
14     printf("Host ordered port: %#x \n", host_port);
15     printf("Network ordered port: %#x \n", net_port);
16     printf("Host ordered address: %#lx \n", host_addr);
17     printf("Network ordered address: %#lx \n", net_addr);
18     return 0;
19 }
```

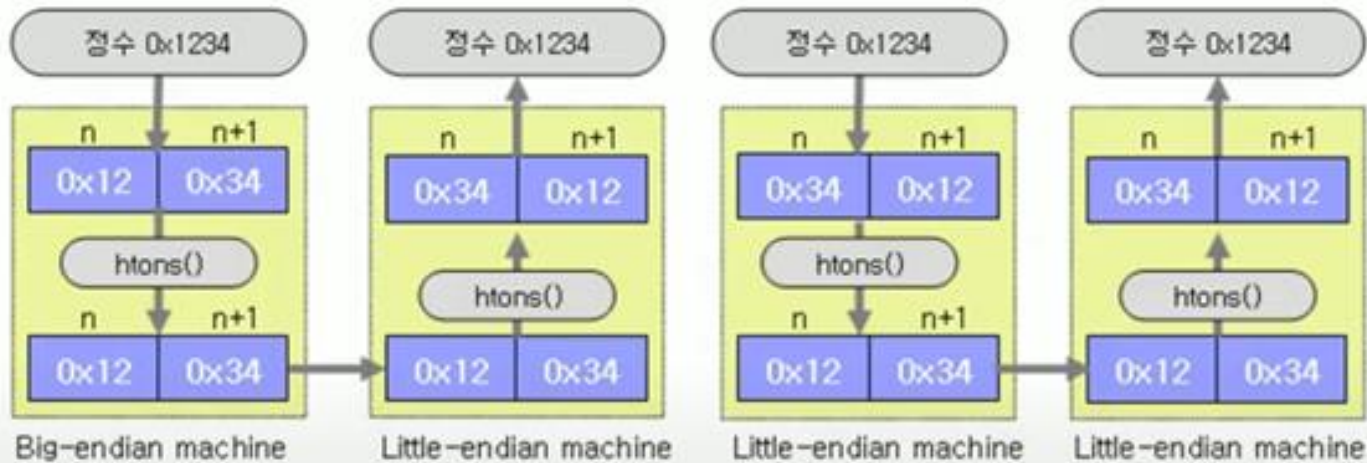
IPV4 기반의 주소표현을 위한 구조체

네트워크/호스트 바이트 순서 간 자료 변환

포트별 서비스 내역 열람

```
#include <netinet/in.h>

unsigned long htonl(unsigned long hostlong);
unsigned long htons(unsigned short hostlshort);
```



그림출처 : 열혈강의 정석용 TCP/IP 소켓 프로그래밍

IPV4 기반의 주소표현을 위한 구조체

```
struct sockaddr_in
{
    sa_family_t    sin_family; 주소체계
    uint16_t       sin_port; PORT번호
    struct in_addr sin_addr; 32비트 IP주소
    char           sin_zero[8]; 사용되지 않음
};
```

IP주소와 PORT번호는 구조체 `sockaddr_in`의 변수에 담아서 표현한다.

```
struct in_addr
{
    in_addr_t      s_addr;
};
```

32비트 IPv4 인터넷 주소

자료형 이름	자료형에 담길 정보
int8_t	signed 8-bit int
uint8_t	unsigned 8-bit int (unsigned char)
int16_t	signed 16-bit int
uint16_t	unsigned 16-bit int (unsigned short)
int32_t	signed 32-bit int
uint32_t	unsigned 32-bit int (unsigned long)
sa_family_t	주소체계(address family)
socklen_t	길이정보(length of struct)
in_addr_t	IP주소정보, uint32_t로 정의되어 있음
in_port_t	PORT번호정보, uint16_t로 정의되어 있음

문자열 정보를 네트워크 바이트 순서의 정수로 변환

```
#include <arpa/inet.h>
```

```
in_addr_t inet_addr(const char * string);
```

➔ 성공 시 빅 엔디안으로 변환된 32비트 정수 값, 실패 시 INADDR_NONE 반환

“211.214.107.99” 와 같이 점이찍힌 10진수로 표현된 문자열을 전달하면, 해당 문자열 정보를 참조해서 IP주소정보를 32비트 정수형으로 반환!

```
#include <stdio.h>
#include <arpa/inet.h>

int main(int argc, char *argv[])
{
    char *addr1="127.212.124.78";
    char *addr2="127.212.124.256";

    unsigned long conv_addr=inet_addr(addr1);
    if(conv_addr==INADDR_NONE)
        printf("Error occured! \n");
    else
        printf("Network ordered integer addr: %#1x \n", conv_addr);

    conv_addr=inet_addr(addr2);
    if(conv_addr==INADDR_NONE)
        printf("Error occured \n");
    else
        printf("Network ordered integer addr: %#1x \n\n", conv_addr);
    return 0;
}
```

inet_aton

```
#include <arpa/inet.h>
```

```
int inet_aton(const char * string, struct in_addr * addr);
```

→ 성공 시 1(true), 실패 시 0(false) 반환

- string 변환할 IP주소 정보를 담고 있는 문자열의 주소 값 전달.
- addr 변환된 정보를 저장할 in_addr 구조체 변수의 주소 값 전달.

기능상으로 inet_addr 함수와 동일하다. 다만 in_addr형 구조체 변수에 변환의 결과가 저장된다는 점에서 차이를 보인다.

```
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>
void error_handling(char *message);

int main(int argc, char *argv[])
{
    char *addr="127.232.124.79";
    struct sockaddr_in addr_inet;

    if(!inet_aton(addr, &addr_inet.sin_addr))
        error_handling("Conversion error");
    else
        printf("Network ordered integer addr: %#x \n", addr_inet.sin_addr.s_addr);
    return 0;
}

void error_handling(char *message)
{
    fputs(message, stderr);
    fputc('\n', stderr);
    exit(1);
}
```

inet_ntoa

```
#include <arpa/inet.h>
```

```
char * inet_ntoa(struct in_addr adr);
```

➔ 성공 시 변환된 문자열의 주소 값, 실패 시 -1 반환

inet_aton 함수의 반대기능 제공! 네트워크 바이트 순서로 정렬된 정수형 IP주소정보를 우리가 눈으로 쉽게 인식할 수 있는 문자열의 형태로 변환.

```
~/socket_programming
1 #include <stdio.h>
2 #include <string.h>
3 #include <arpa/inet.h>
4
5 int main(int argc, char *argv[])
6 {
7     struct sockaddr_in addr1, addr2;
8     char *str_ptr;
9     char str_arr[20];
10
11     addr1.sin_addr.s_addr=htonl(0x1020304);
12     addr2.sin_addr.s_addr=htonl(0x1010101);
13
14     str_ptr=inet_ntoa(addr1.sin_addr);
15     strcpy(str_arr, str_ptr);
16     printf("Dotted-Decimal notation1: %s \n", str_ptr);
17
18     inet_ntoa(addr2.sin_addr);
19     printf("Dotted-Decimal notation2: %s \n", str_ptr);
20     printf("Dotted-Decimal notation3: %s \n", str_arr);
21     return 0;
22 }
```

1,1

Top

인터넷주소의 초기화

일반적인 인터넷 주소의 초기화 과정

```
struct sockaddr_in addr;  
char *serv_ip="211.217.168.13";      // IP주소 문자열 선언  
char *serv_port="9190";              // PORT번호 문자열 선언  
memset(&addr, 0, sizeof(addr));      // 구조체 변수 addr의 모든 멤버 0으로 초기화  
addr.sin_family=AF_INET;              // 주소체계 지정  
addr.sin_addr.s_addr=inet_addr(serv_ip); // 문자열 기반의 IP주소 초기화  
addr.sin_port=htons(atoi(serv_port)); // 문자열 기반의 PORT번호 초기화
```

서버에서 주소정보를 설정하는 이유!

"IP 211.217.168.13, PORT 9190으로 들어오는 데이터는 내게로 다 보내라!"

클라이언트에서 주소정보를 설정하는 이유!

"IP 211.217.168.13, PORT 9190으로 연결을 해라!"

INADDR_ANY

```
struct sockaddr_in addr;  
char *serv_port="9190";  
memset(&addr, 0, sizeof(addr));  
addr.sin_family=AF_INET;  
addr.sin_addr.s_addr=htonl(INADDR_ANY);  
addr.sin_port=htons(atoi(serv_port));
```

현재 실행중인 컴퓨터의 IP를 소켓에 부여할때 사용되는 것이 INADDR_ANY이다. 이는 서버 프로그램의 구현에 주로 사용된다.

`./hserver 9190`

서버의 실행방식, 서버의 리스닝 소켓 주소는 `INADDR_ANY`로 지정을 하니, 소켓의 `PORT`번호만 인자를 통해 전달하면 된다.

`./hclient 127.0.0.1 9190`

클라이언트의 실행방식, 연결할 서버의 `IP`와 `PORT`번호를 인자로 전달한다.

`127.0.0.1`은 루프백 주소라 하며, 이는 클라이언트를 실행하는 컴퓨터의 `IP`주소를 의미한다.

루프백 주소를 전달한 이유는, 서버와 클라이언트를 한 대의 컴퓨터에서 실행시켰기 때문이다.

소켓에 인터넷 주소 할당하기

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

```
int bind(int sockfd, struct sockaddr *myaddr, socklen_t addrlen);
```

➔ 성공 시 0, 실패 시 -1 반환

- sockfd 주소정보(IP와 PORT를) 할당할 소켓의 파일 디스크립터.
- myaddr 할당하고자 하는 주소정보를 지니는 구조체 변수의 주소 값.
- addrlen 두 번째 인자로 전달된 구조체 변수의 길이정보.

```
int serv_sock;
```

```
struct sockaddr_in serv_addr;
```

```
char *serv_port="9190";
```

```
/* 서버 소켓(리스닝 소켓) 생성 */
```

```
serv_sock=socket(PF_INET, SOCK_STREAM, 0);
```

```
/* 주소정보 초기화 */
```

```
memset(&serv_addr, 0, sizeof(serv_addr));
```

```
serv_addr.sin_family=AF_INET;
```

```
serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
```

```
serv_addr.sin_port=htons(atoi(serv_port));
```

```
/* 주소정보 할당 */
```

```
bind(serv_sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
```

```
.....
```

서버프로그램에서의 일반적인

주소할당의 과정!

Iterative 서버의 구현

socket()

bind()

listen()

accept()

read()/write()

close(client)

close(server)

- 서버는 한 순간에 하나의 클라이언트와 연결되어 에코 서비스를 제공한다.
- 서버는 총 다섯 개의 클라이언트에게 순차적으로 서비스를 제공하고 종료한다.
- 클라이언트는 프로그램 사용자로부터 문자열 데이터를 입력 받아서 서버에 전송한다.
- 서버는 전송 받은 문자열 데이터를 클라이언트에게 재전송한다. 즉 에코 시킨다.
- 서버와 클라이언트간의 문자열 에코는 클라이언트가 Q를 입력할 때 까지 계속한다.

왼쪽의 그림과 같이 반복적으로 `accept` 함수를 호출하면, 계속해서 클라이언트의 연결요청을 수락할 수 있다. 그러나, 동시에 둘 이상의 클라이언트에게 서비스를 제공할 수 있는 모델은 아니다.

Iterative 서버의 구현

제대로 동작은 하나 문제의 발생 소지가 있는 TCP 에코 클라이언트의 코드

```
write(sock, message, strlen(message));  
str_len=read(sock, message, BUF_SIZE-1);  
message[str_len]=0;  
printf("Message from server: %s", message);
```

TCP의 데이터 송수신에는 경계가 존재하지 않는다! 그런데 위의 코드는 다음 사항을 가정하고 있다.

“한 번의 read 함수호출로 앞서 전송된 문자열 전체를 읽어 들일 수 있다.”

그러나 이는 잘못된 가정이다. TCP에는 데이터의 경계가 존재하지 않기 때문에 서버가 전송한 문자열의 일부만 읽혀질 수도 있다.

에코 클라이언트의 문제점 확인

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. TCP가 본디 데이터의 경계가 없는 프로토콜이므로, 두 번의 write 함수 호출을 통해서 데이터를 전송하건, 세 번의 write 함수호출을 통해서 데이터를 전송하건, 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 & write 함수호출은 데이터의 경계를 구분하지 않기 때문이다.

에코 클라이언트의 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 함수호출을 통해서 전송한 데이터의 길이만큼 읽어 들이기 위한 반복문의 삽입이 필요하다. 이것이 TCP를 기반으로 데이터를 구분지어 읽어 들이는데 부가적으로 필요한 구분이다.

Iterative echo 서버 클라이언트 구현

echo_server.c

```
~/socket_programming
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <arpa/inet.h>
6 #include <sys/socket.h>
7
8 #define BUF_SIZE 1024
9 void error_handling(char *message);
10
11 int main(int argc, char *argv[])
12 {
13     int serv_sock, clnt_sock;
14     char message[BUF_SIZE];
15     int str_len, i;
16
17     struct sockaddr_in serv_adr;
18     struct sockaddr_in clnt_adr;
19     socklen_t clnt_adr_sz;
20
21     if(argc!=2) {
22         printf("Usage : %s <port>\n", argv[0]);
23         exit(1);
24     }
25
26     serv_sock=socket(PF_INET, SOCK_STREAM, 0);
27     if(serv_sock==-1)
28         error_handling("socket() error");
29
30     memset(&serv_adr, 0, sizeof(serv_adr));
31     serv_adr.sin_family=AF_INET;
32     serv_adr.sin_addr.s_addr=htonl(INADDR_ANY);
33     serv_adr.sin_port=htons(atoi(argv[1]));
34
35     if(bind(serv_sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr))==-1)
36         error_handling("bind() error");
37
38     "echo_server.c" [noel] [dos] 66L, 1440C
```

1,1

Top

Iterative echo 서버 클라이언트 구현

echo_server.c

```
~/socket_programming
31 serv_addr.sin_family=AF_INET;
32 serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
33 serv_addr.sin_port=htons(atoi(argv[1]));
34
35 if(bind(serv_sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr))==-1)
36     error_handling("bind() error");
37
38 if(listen(serv_sock, 5)==-1)
39     error_handling("listen() error");
40
41 clnt_adr_sz=sizeof(clnt_adr);
42
43 for(i=0; i<5; i++)
44 {
45     clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_adr, &clnt_adr_sz);
46     if(clnt_sock==-1)
47         error_handling("accept() error");
48     else
49         printf("Connected client %d \n", i+1);
50
51     while((str_len=read(clnt_sock, message, BUF_SIZE))!=0)
52         write(clnt_sock, message, str_len);
53
54     close(clnt_sock);
55 }
56
57 close(serv_sock);
58 return 0;
59 }
60
61 void error_handling(char *message)
62 {
63     fputs(message, stderr);
64     fputc('\n', stderr);
65     exit(1);
66 }
```

Iterative echo 서버 클라이언트 구현

echo_client.c

```
~/socket_programming
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <arpa/inet.h>
6 #include <sys/socket.h>
7
8 #define BUF_SIZE 1024
9 void error_handling(char *message);
10
11 int main(int argc, char *argv[])
12 {
13     int sock;
14     char message[BUF_SIZE];
15     int str_len, recv_len, recv_cnt;
16     struct sockaddr_in serv_addr;
17
18     if(argc!=3) {
19         printf("Usage : %s <IP> <port>\n", argv[0]);
20         exit(1);
21     }
22
23     sock=socket(PF_INET, SOCK_STREAM, 0);
24     if(sock==-1)
25         error_handling("socket() error");
26
27     memset(&serv_addr, 0, sizeof(serv_addr));
28     serv_addr.sin_family=AF_INET;
29     serv_addr.sin_addr.s_addr=inet_addr(argv[1]);
30     serv_addr.sin_port=htons(atoi(argv[2]));
31
32     if(connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr))==-1)
33         error_handling("connect() error!");
34     else
35         puts("Connected.....");
36
37     "echo_client2.c" [dos] 69L, 1473C
```

Iterative echo 서버 클라이언트 구현

echo_client.c

```
~/socket_programming
34     else
35         puts("Connected.....");
36
37     while(1)
38     {
39         fputs("Input message(Q to quit): ", stdout);
40         fgets(message, BUF_SIZE, stdin);
41
42         if(!strcmp(message, "q\n") || !strcmp(message, "Q\n"))
43             break;
44
45         str_len=write(sock, message, strlen(message));
46
47         recv_len=0;
48         while(recv_len<str_len)
49         {
50             recv_cnt=read(sock, &message[recv_len], BUF_SIZE-1);
51             if(recv_cnt==-1)
52                 error_handling("read() error!");
53             recv_len+=recv_cnt;
54         }
55
56         message[recv_len]=0;
57         printf("Message from server: %s", message);
58     }
59
60     close(sock);
61     return 0;
62 }
63
64 void error_handling(char *message)
65 {
66     fputs(message, stderr);
67     fputc('\n', stderr);
68     exit(1);
69 }
```

69,1

Bot

Echo – Server , Client

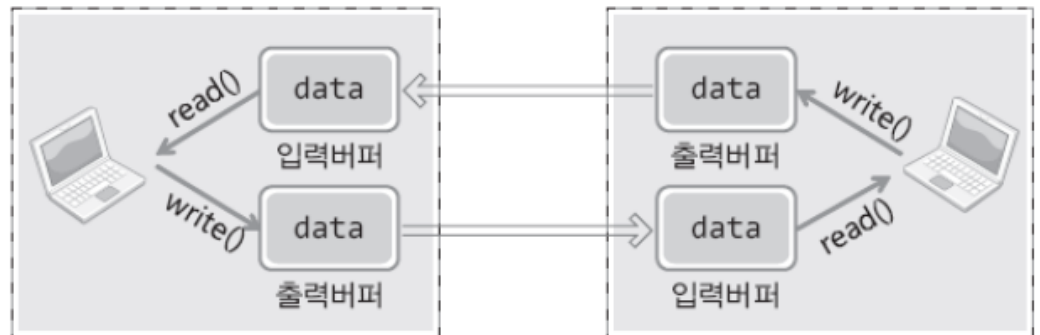
```
str_len=write(sock, message, strlen(message));  
recv_len=0;  
while(recv_len<str_len)  
{  
    recv_cnt=read(sock, &message[recv_len], BUF_SIZE-1);  
    if(recv_cnt==-1)  
        error_handling("read() error!");  
    recv_len+=recv_cnt;  
}  
message[recv_len]=0;  
printf("Message from server: %s", message);
```

write 함수호출을 통해서 전송한 데이터의 길이만큼 읽어 들이기 위한 반복문의 삽입이 필요하다. 이것이 TCP를 기반으로 데이터를 구분지어 읽어 들이는데 부가적으로 필요한 구분이다.

TCP 소켓에 존재하는 입출력 버퍼

- 입출력 버퍼는 TCP 소켓 각각에 대해 별도로 존재한다.
- 입출력 버퍼는 소켓생성시 자동으로 생성된다.
- 소켓을 닫아도 출력버퍼에 남아있는 데이터는 계속해서 전송이 이뤄진다.
- 소켓을 닫으면 입력버퍼에 남아있는 데이터는 소멸되어버린다.

이와 같은 버퍼가 존재하기 때문에 데이터의 슬라이딩 윈도우 프로토콜의 적용이 가능하고, 이로 인해서 버퍼가 차고 넘치는 상황은 발생하지 않는다.



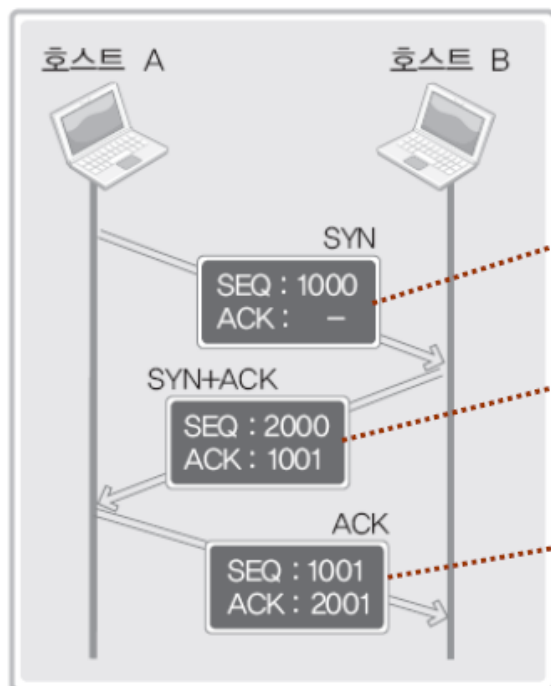
소켓 A 야 50바이트까지는 보내도 괜찮아!
소켓 B OK!

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

슬라이딩 윈도우 프로토콜의
데이터 송수신 유형

상대소켓과의 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.

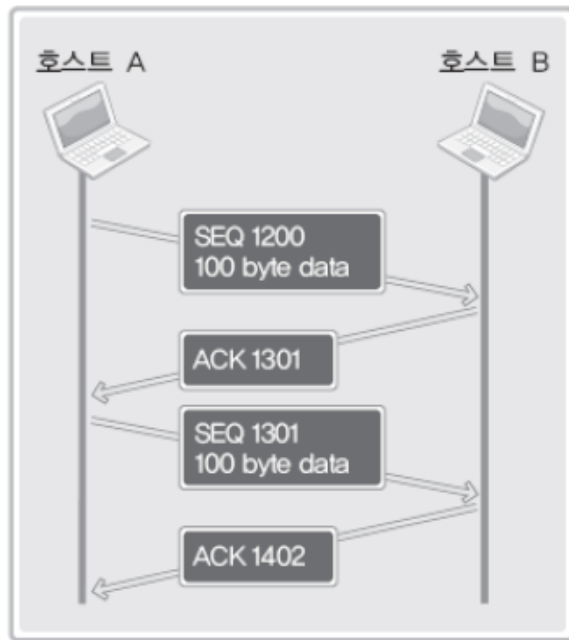


“내가 지금 보내는 이 패킷에 1000이라는 번호를 부여하니, 잘 받았다면 다음에는 1001번 패킷을 전달하라고 내게 말해달라!”

“내가 지금 보내는 이 패킷에 2000이라는 번호를 부여하니, 잘 받았다면 다음에는 2001번 패킷을 전달하라고 내게 말해달라!”

“좀 전에 전송한 SEQ가 1000인 패킷은 잘 받았으니, 다음 번에는 SEQ가 1001인 패킷을 전송하기 바란다!”

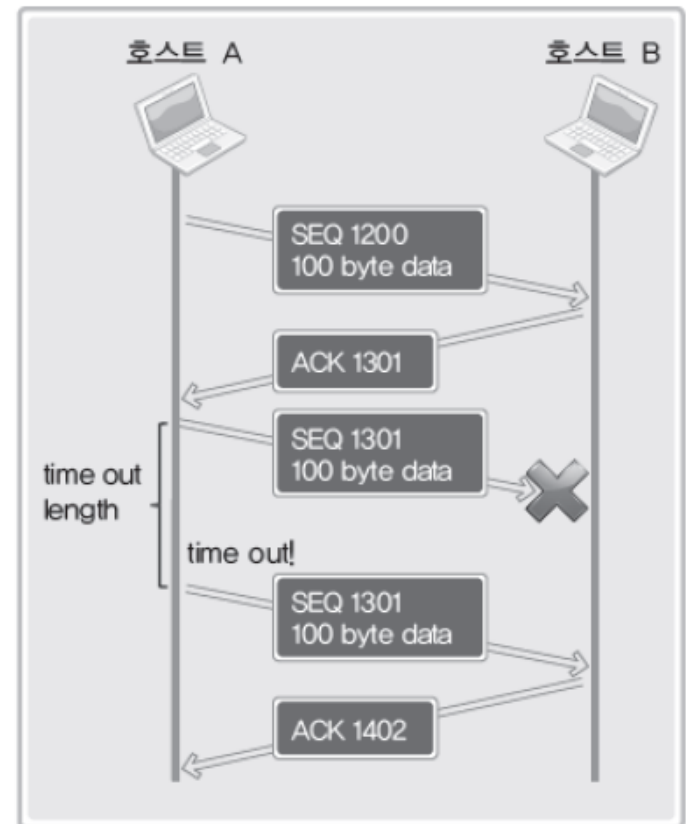
상대소켓과의 연결



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

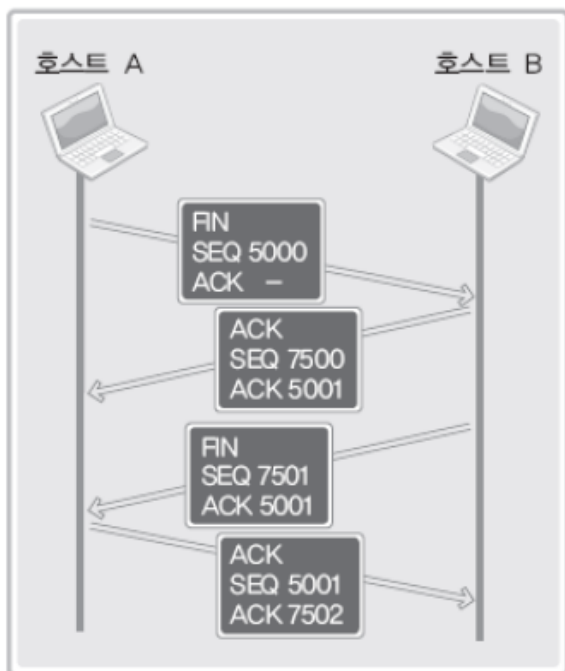
ACK 번호 \rightarrow SEQ 번호 + 전송된 바이트 크기 + 1

SEQ 전송 시 타이머 작동, 그리고 SEQ에 대한 ACK가 전송되지 않을 경우 데이터 재전송



상대소켓과의 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.

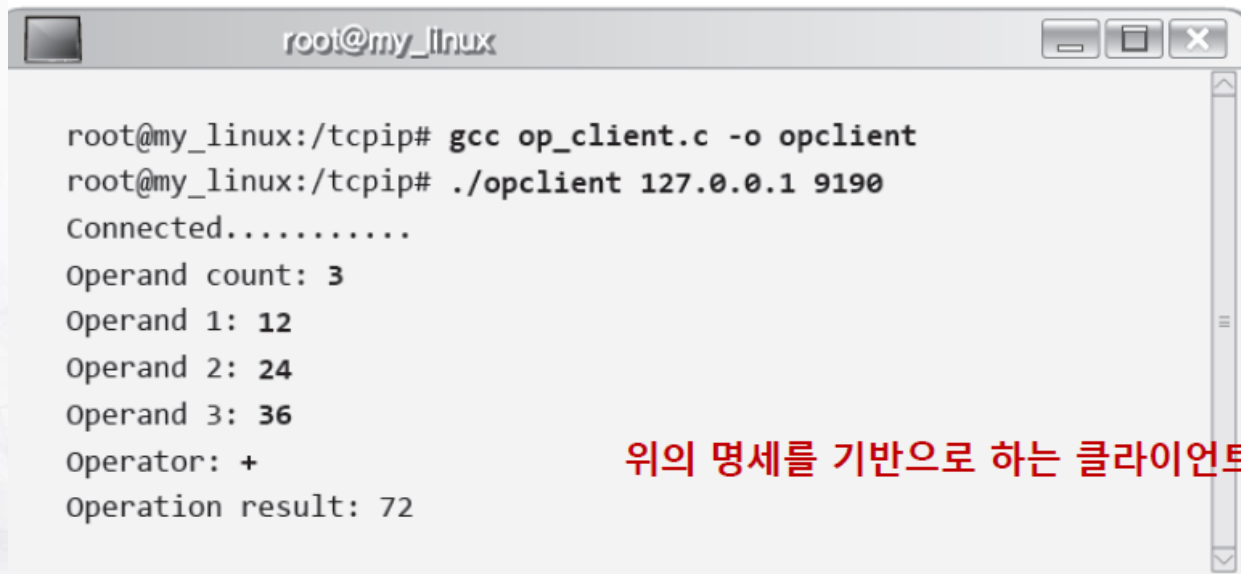


four-way handshaking 과정을 거쳐서 연결을 종료하는 이유는
일방적 종료로 인한 데이터의 손실을 막기 위함이다.

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 $3+5+9$ 의 연산결과가 전달되어야 하고, 곱셈연산이 요청된다면 클라이언트에는 $3\times 5\times 9$ 의 연산결과가 전달되어야 한다. 단, 서버로 4, 3, 2가 전달되고 뺄셈연산이 요청된다면 클라이언트에는 $4-3-2$ 의 연산결과가 전달되어야 한다. 즉, 뺄셈의 경우에는 첫 번째 정수를 대상으로 뺄셈이 진행되어야 한다.

이와 같은 서버 클라이언트 사이에서의 데이터 송수신 명세가 바로 프로토콜이다!



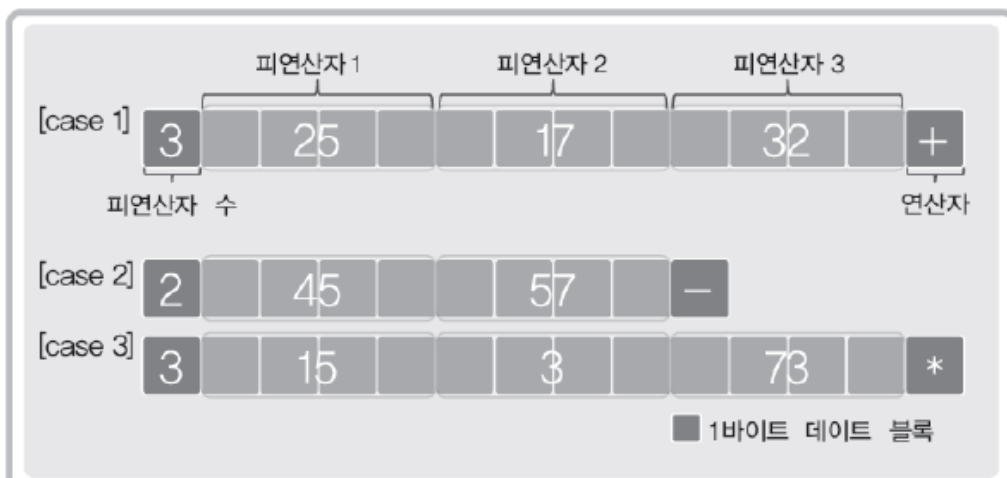
```
root@my_linux

root@my_linux:/tcpip# gcc op_client.c -o opclient
root@my_linux:/tcpip# ./opclient 127.0.0.1 9190
Connected.....
Operand count: 3
Operand 1: 12
Operand 2: 24
Operand 3: 36
Operator: +
Operation result: 72
```

위의 명세를 기반으로 하는 클라이언트 프로그램의 실행의 예

어플리케이션 프로토콜 구체화

- 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.



OP_Server.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <arpa/inet.h>
6 #include <sys/socket.h>
7 |
8 #define BUF_SIZE 1024
9 #define OPSZ 4
10 void error_handling(char *message);
11 int calculate(int opnum, int opnds[], char oprator);
12
13 int main(int argc, char *argv[])
14 {
15     int serv_sock, clnt_sock;
16     char opinfo[BUF_SIZE];
17     int result, opnd_cnt, i;
18     int recv_cnt, recv_len;
19     struct sockaddr_in serv_adr, clnt_adr;
20     socklen_t clnt_adr_sz;
21     if(argc!=2) {
22         printf("Usage : %s <port>\n", argv[0]);
23         exit(1);
24     }
25
26     serv_sock=socket(PF_INET, SOCK_STREAM, 0);
27     if(serv_sock==-1)
28         error_handling("socket() error");
29
30     memset(&serv_adr, 0, sizeof(serv_adr));
31     serv_adr.sin_family=AF_INET;
```

OP_Server.c -1

```
32 serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
33 serv_addr.sin_port=htons(atoi(argv[1]));
34
35 if(bind(serv_sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr))== -1)
36     error_handling("bind() error");
37 if(listen(serv_sock, 5)== -1)
38     error_handling("listen() error");
39 clnt_adr_sz=sizeof(clnt_adr);
40
41 for(i=0; i<5; i++)
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 }
57 close(serv_sock);
58 return 0;
59 }
```

```
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 }
79
80 void error_handling(char *message)
81 {
82     fputs(message, stderr);
83     fputc('\n', stderr);
84     | exit(1);
85 }
```

OP_Client.c

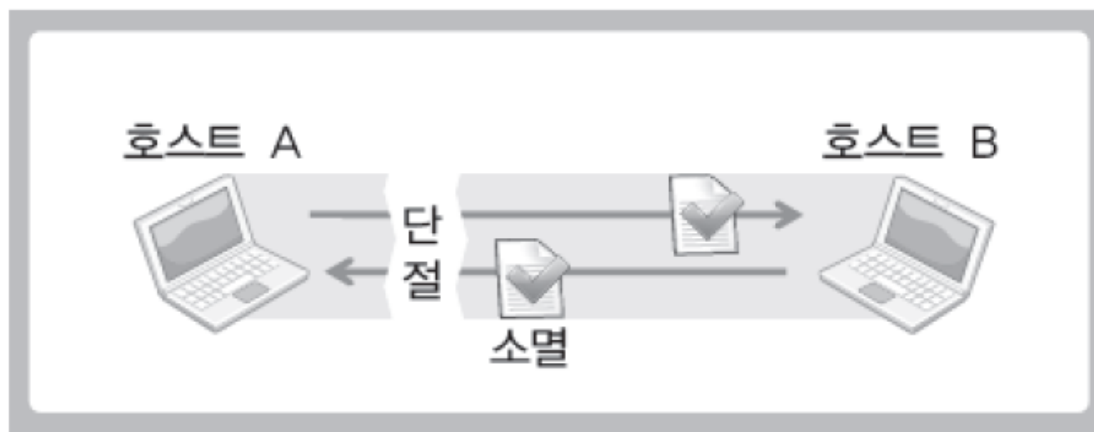
```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <arpa/inet.h>
6 #include <sys/socket.h>
7
8 #define BUF_SIZE 1024
9 #define RLT_SIZE 4
10 #define OPSZ 4
11 void error_handling(char *message);
12
13 int main(int argc, char *argv[])
14 {
15     int sock;
16     char opmsg[BUF_SIZE];
17     int result, opnd_cnt, i;
18     struct sockaddr_in serv_addr;
19     if(argc!=3) {
20         printf("Usage : %s <IP> <port>\n", argv[0]);
21         exit(1);
22     }
23
24     sock=socket(PF_INET, SOCK_STREAM, 0);
25     if(sock==-1)
26         error_handling("socket() error");
27
28     memset(&serv_addr, 0, sizeof(serv_addr));
29     serv_addr.sin_family=AF_INET;
30     serv_addr.sin_addr.s_addr=inet_addr(argv[1]);
31     serv_addr.sin_port=htons(atoi(argv[2]));
```

OP_Client.c -1

```
33     if(connect(sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr))== -1)
34         error_handling("connect() error!");
35     else
36         puts("Connected.....");
37
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 Wn", result);
54     close(sock);
55     return 0;
56 }
57
58 void error_handling(char *message)
59 {
60     fputs(message, stderr);
61     fputc('Wn', stderr);
62     | exit(1);
63 }
```


일반적인 연결종료의 문제점

- ▶ close 및 closesocket 함수의 기능
 - ▶ 소켓의 완전 소멸을 의미한다.
 - ▶ 소켓이 소멸되므로 더 이상의 입출력은 불가능하다.
 - ▶ 상대방의 상태에 상관 없이 일방적인 종료를 형태를 띤다.
 - ▶ 때문에 상대 호스트의 데이터 송수신이 아직 완료되지 않은 상황이라면, 문제가 발생할 수 있다.
 - ▶ 이러한 문제의 대안으로 Half-close 기법이 존재한다.



▶ Half-close

- ▶ 종료를 원한다는 것은, 더 이상 전송할 데이터가 존재하지 않는 상황이다.
- ▶ 따라서 출력 스트림은 종료를 시켜도 된다.
- ▶ 다만 상대방도 종료를 원하는지 확인되지 않은 상황이므로, 출력 스트림은 종료시키지 않을 필요가 있다.
- ▶ 때문에 일반적으로 Half-close라 하면, 입력 스트림만 종료하는 것을 의미한다.
- ▶ Half-close를 가리켜 '우아한 종료'라고도 한다.



입력 또는 출력 스트림 중
하나만 종료하는 것을 가리
켜 Half-close라 한다.

우아한 종료를 위한 Shut-down 함수와 그 필요성

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

```
int shutdown(int sock, int howto);
```

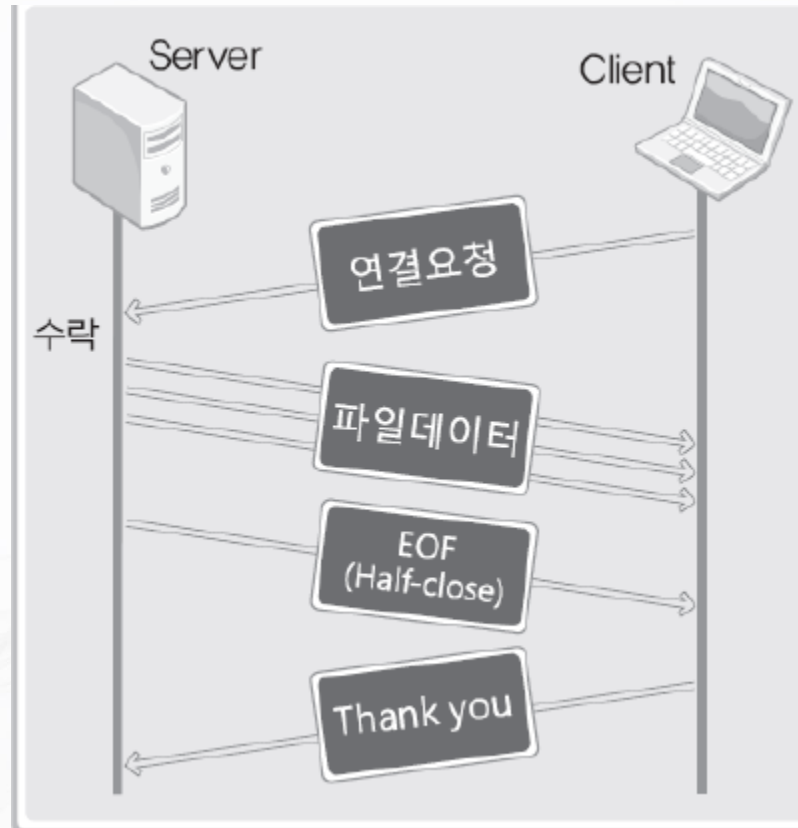
➔ 성공 시 0, 실패 시 -1 반환

- sock 종료할 소켓의 파일 디스크립터 전달.
- howto 종료방법에 대한 정보 전달.

- SHUT_RD 입력 스트림 종료
- SHUT_WR 출력 스트림 종료
- SHUT_RDWR 입출력 스트림 종료

close 함수가 호출되면 **상대 호스트(소켓)으로 EOF가 전달된다**. 그리고 이는 **모든 데이터의 전송이 끝났다는 신호의 의미**를 갖는다. 이것이 종료 이외의 close 함수를 호출하는 목적이다. 그런데 **출력 스트림만 종료를 해도 EOF가 전달이 되니**, close 함수의 호출을 대체하고도, **상대 호스트의 종료를 기다릴 수 있다**.

Half-close 기반 파일 전송 프로그램



file_server.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <arpa/inet.h>
6 #include <sys/socket.h>
7
8 #define BUF_SIZE 30
9 void error_handling(char *message);
10
11 int main(int argc, char *argv[])
12 {
13     int serv_sd, clnt_sd;
14     FILE * fp;
15     char buf[BUF_SIZE];
16     int read_cnt;
17
18     struct sockaddr_in serv_adr, clnt_adr;
19     socklen_t clnt_adr_sz;
20
21     if(argc!=2) {
22         printf("Usage: %s <port>\n", argv[0]);
23         exit(1);
24     }
25
26     fp=fopen("file_server.c", "rb");
27     serv_sd=socket(PF_INET, SOCK_STREAM, 0);
28
```

file_server.c - 1

```
28
29     memset(&serv_adr, 0, sizeof(serv_adr));
30     serv_adr.sin_family=AF_INET;
31     serv_adr.sin_addr.s_addr=htonl(INADDR_ANY);
32     serv_adr.sin_port=htons(atoi(argv[1]));
33
34     bind(serv_sd, (struct sockaddr*)&serv_adr, sizeof(serv_adr));
35     listen(serv_sd, 5);
36
37     clnt_adr_sz=sizeof(clnt_adr);
38     clnt_sd=accept(serv_sd, (struct sockaddr*)&clnt_adr, &clnt_adr_sz);
39
40     while(1)
41     {
42         read_cnt=fread((void*)buf, 1, BUF_SIZE, fp);
43         if(read_cnt<BUF_SIZE)
44         {
45             write(clnt_sd, buf, read_cnt);
46             break;
47         }
48         | write(clnt_sd, buf, BUF_SIZE);
49     }
50
51     shutdown(clnt_sd, SHUT_WR);
52     read(clnt_sd, buf, BUF_SIZE);
53     printf("Message from client: %s\n", buf);
54
55     fclose(fp);
```

```
56     close(clnt_sd); close(serv_sd);
57     return 0;
58 }
59
60 void error_handling(char *message)
61 {
62     fputs(message, stderr);
63     fputc('\n', stderr);
64     exit(1);
65 }
```

file_client.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <arpa/inet.h>
6 #include <sys/socket.h>
7
8 #define BUF_SIZE 30
9 void error_handling(char *message);
10
11 int main(int argc, char *argv[])
12 {
13     int sd;
14     FILE *fp;
15
16     char buf[BUF_SIZE];
17     int read_cnt;
18     struct sockaddr_in serv_addr;
19     if(argc!=3) {
20         printf("Usage: %s <IP> <port>\n", argv[0]);
21         exit(1);
22     }
23
24     fp=fopen("receive.dat", "wb");
25     sd=socket(PF_INET, SOCK_STREAM, 0);
26
27     memset(&serv_addr, 0, sizeof(serv_addr));
28     serv_addr.sin_family=AF_INET;
```



```
29     serv_adr.sin_addr.s_addr=inet_addr(argv[1]);
30     serv_adr.sin_port=htons(atoi(argv[2]));
31
32     connect(sd, (struct sockaddr*)&serv_adr, sizeof(serv_adr));
33
34     while((read_cnt=read(sd, buf, BUF_SIZE ))!=0)
35         fwrite((void*)buf, 1, read_cnt, fp);
36
37     puts("Received file data");
38     write(sd, "Thank you", 10);
39     fclose(fp);
40     close(sd);
41     return 0;
42 }
43
44 void error_handling(char *message)
45 {
46     fputs(message, stderr);
47     fputc('\n', stderr);
48     exit(1);
49 }
```

9주차 수업이 끝났습니다

고생하셨습니다.

