컴퓨터 네트워크

Socket 프로그래밍

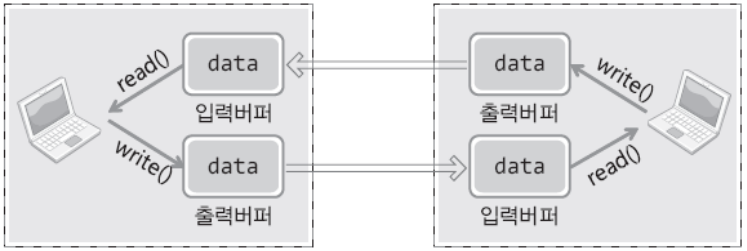
Ducsun Lim
Mobile&Network Intelligence Lab.
Hanyang University
2019/11/25

TCP의 이론적인 부분

TCP 소켓에 존재하는 입출력 버퍼

- 입출력 버퍼는 TCP 소켓 각각에 대해 별도로 존재함
- 입출력 버퍼는 소켓 생성 시 자동으로 생성됨
- 소켓을 닫아도 출력 버퍼에 남아있는 데이터는 계속해서 전송이 이뤄짐
- 소켓을 닫으면 입력 버퍼에 남아있는 데이터는 소멸되어 버림

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



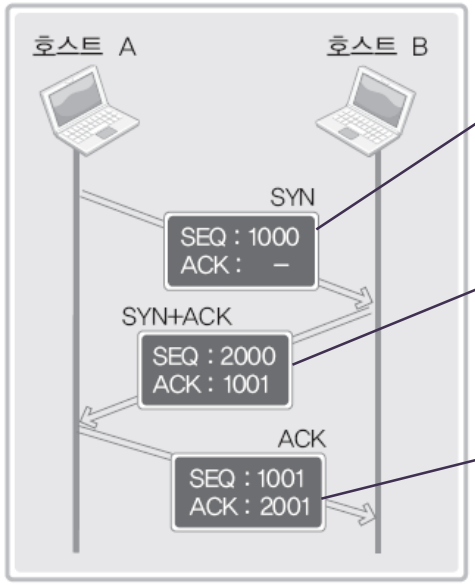
- **슬라이딩 윈도우 프로토콜의 데이터 송수신 유형**

소켓A 50 byte까지는 보내도 괜찮음
소켓B OK!

소켓A 현재 20바이트 비워 있으니 70byte까지 보내도 괜찮음
소켓B OK!

TCP의 내부 동작 원리 1: 상대 소켓과의 연결

[Shake 1] socket A : socket B에게 전달할 데이터가 있으니 연결을 요청함
[Shake 2] socket B : 준비가 되면 시작해도 좋다는 전달을 함
[Shake 2] socket A : 요청에 대해 회신을 함

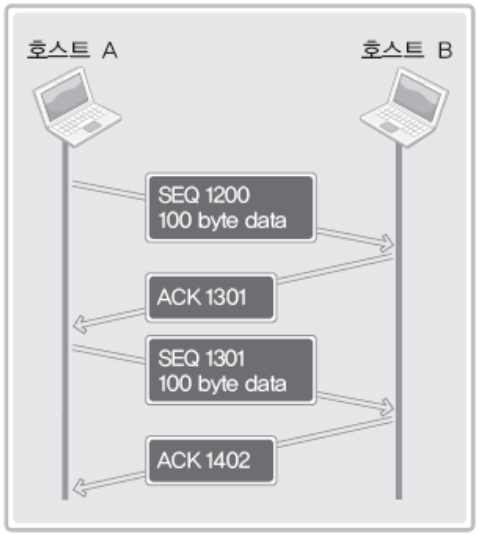


호스트 A는 지금 보내는 이 패킷에 1000이라는 번호를 부여하니, 잘 받았다면 다음에는 1001번 패킷을 전달하라고 요청함

호스트 B는 지금 보내는 이 패킷에 2000이라는 번호를 부여하니, 잘 받았다면 다음에는 2001번 패킷을 전달하라고 요청함

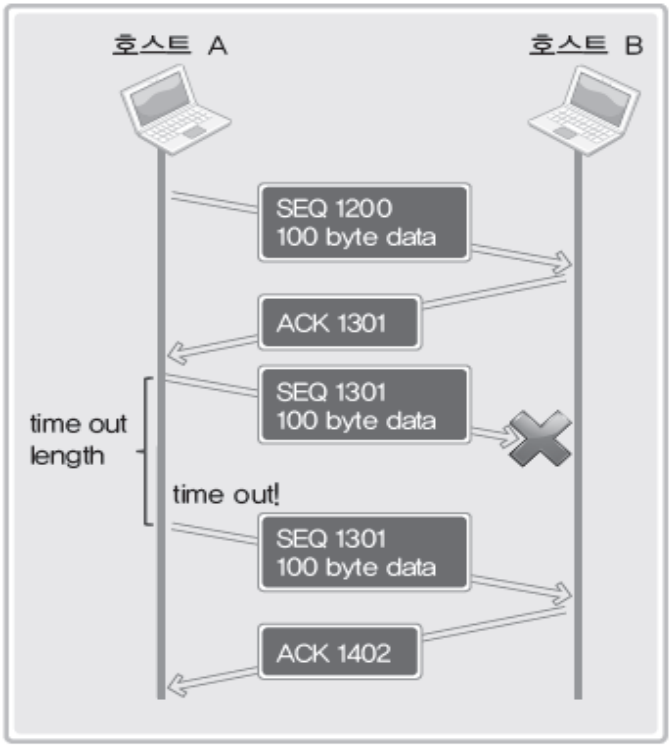
전송한 SEQ가 2000인 패킷은 잘 받았으니, SEQ가 1001인 패킷을 전달함

TCP의 내부 동작 원리 2: 상대 소켓과의 연결



ACK의 값을 전송된 바이트 크기만큼 증가시키는 이유는 패킷의 전송 유무 뿐만 아니라, 데이터의 손실 유무까지 확인하기 위함

ACK 번호 → SEQ 번호 + 전송된 바이트 크기 + 1

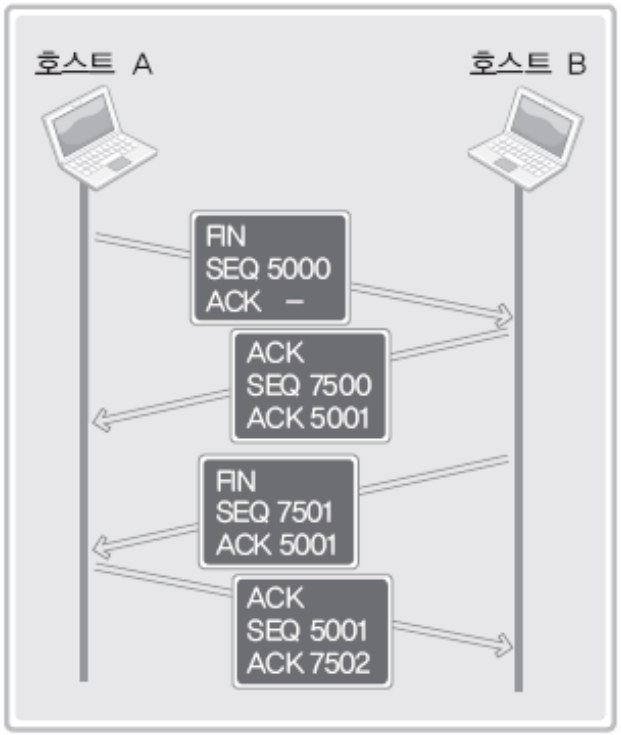


SEQ 전송 시 타이머 작동, 그리고 SEQ에 대한 ACK가 전송되지 않을 경우 데이터 재전송

TCP의 내부 동작 원리 3: 상대 소켓과의 연결

- Socket A: 이제 연결을 끊고자 함
- Socket B: 잠시만 대기

- Socket B: 준비가 끝났음 그럼 연결을 끊어도 됨
- Socket A: ok



four-way handshaking 과정을 거쳐 연결을 종료하는 이유는 일방적 종료로 인한 데이터의 손실을 막기위함

IP 주소와 도메인 이름 사이의 변환

도메인 이름을 이용해서 IP 주소 얻어오기

```
#include <netdb.h>

struct hostent * gethostbyname(const char * hostname);
```

➜ 성공 시 hostent 구조체 변수의 주소 값, 실패 시 NULL 포인터 반환

gethostbyname 함수의 인자로 도메인의 이름 정보를 전달하면, 해당 도메인의 서버 정보가 hostent 구조체 변수에 채워지고, 그 변수의 주소 값이 반환됨

```
struct hostent
{
    char * h_name;        // official name
    char ** h_aliases;    // alias list
    int h_addrtype;       // host address type
    int h_length;         // address length
    char ** h_addr_list;  // address list
}
```

- IP는 도메인이름에 비해 상대적으로 변동이 심함
- 때문에 프로그램 코드상에서 서버의 IP를 직접코드로 입력한다면, 서버의 IP가 변경될 때마다 컴파일을 다시 해야 하는 번거로운 상황이 발생함
- 그러나 상대적으로 변동이 덜한 도메인 이름을 이용해서 서버가 실행될 때마다 IP를 얻어오게 구현한다면, 서버의 코드를 재 컴파일 할 필요가 없음

구조체 hostent에 채워지는 정보의 형태

h_name:

공식도메인이름

h_aliases:

별칭의도메인이름

h_addrtype:

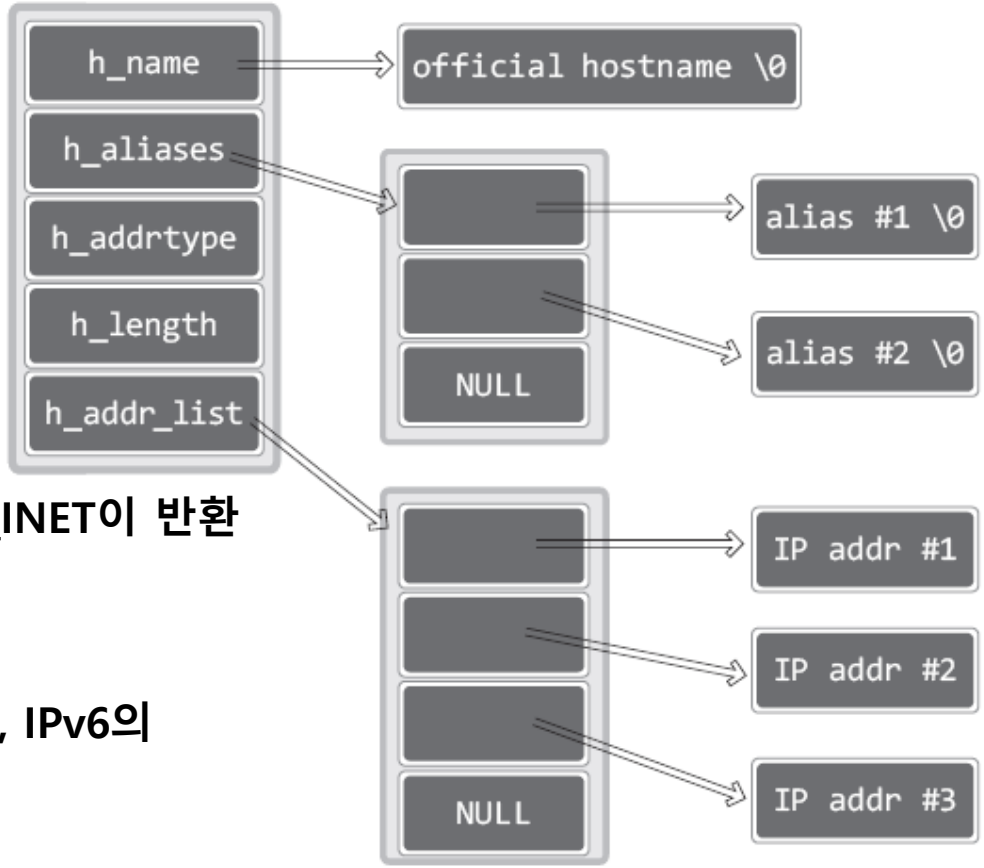
반환된IP의 정보가 IPv4인 경우, AF_INET이 반환

h_length:

반환된IP 정보의 크기, IPv4의 경우4, IPv6의 경우 16이 저장

h_addr_list

IP의 주소정보, 둘 이상의 경우 모두 반환

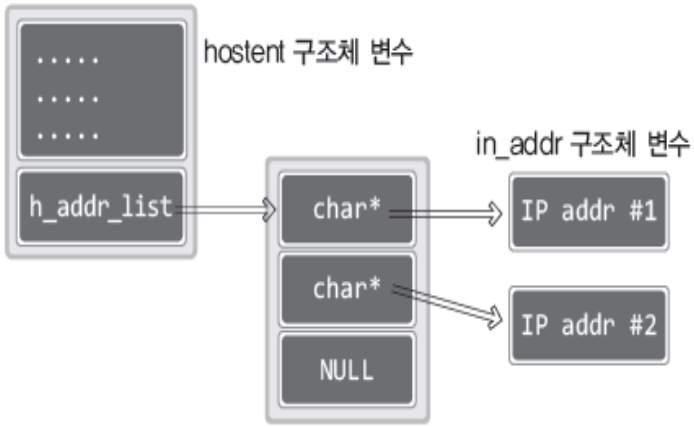


gethostbyname 함수 호출 예

❑ gethostbyname.c의 일부

```
host=gethostbyname(argv[1]);
if(!host)
    error_handling("gethost... error");

printf("Official name: %s \n", host->h_name);
for(i=0; host->h_aliases[i]; i++)
    printf("Aliases %d: %s \n", i+1, host->h_aliases[i]);
printf("Address type: %s \n",
    (host->h_addrtype==AF_INET)?"AF_INET":"AF_INET6");
for(i=0; host->h_addr_list[i]; i++)
    printf("IP addr %d: %s \n", i+1,
        inet_ntoa(*(struct in_addr*)host->h_addr_list[i]));
return 0;
```



- 반복문의 구성을 통해서 반환된 모든 정보를 출력하고 있음
- 정보의 끝은 NULL로 표시가 된다는 사실을 기억

```
root@my_linux:/tcpip# gcc gethostbyname.c -o hostname
root@my_linux:/tcpip# ./hostname www.naver.com
Official name: www.g.naver.com
Aliases 1: www.naver.com
Address type: AF_INET
IP addr 1: 202.131.29.70
IP addr 2: 222.122.195.6
```

IP 주소를 이용해서 도메인 정보 얻어 오기

```
#include <netdb.h>

struct hostent * gethostbyaddr(const char * addr, socklen_t len, int family);
```

➜ 성공 시 hostent 구조체 변수의 주소 값, 실패 시 NULL 포인터 반환

- addr — IP주소를 지니는 in_addr 구조체 변수의 포인터 전달, IPv4 이외의 다양한 정보를 전달받을 수 있도록 일반화하기 위해서 매개변수를 char형 포인터로 선언.
- len — 첫 번째 인자로 전달된 주소정보의 길이, IPv4의 경우 4, IPv6의 경우 16 전달.
- family — 주소체계 정보 전달. IPv4의 경우 AF_INET, IPv6의 경우 AF_INET6 전달.

gethostbyaname 함수의 반대 기능 제공

gethostbyaddr

❑ 예제: gethostbyaddr.c의 일부

```
memset(&addr, 0, sizeof(addr));
addr.sin_addr.s_addr=inet_addr(argv[1]);
host=gethostbyaddr((char*)&addr.sin_addr, 4, AF_INET);
if(!host)
    error_handling("gethost... error");

printf("Official name: %s \n", host->h_name);
for(i=0; host->h_aliases[i]; i++)
    printf("Aliases %d: %s \n", i+1, host->h_aliases[i]);
printf("Address type: %s \n",
    (host->h_addrtype==AF_INET)?"AF_INET":"AF_INET6");
for(i=0; host->h_addr_list[i]; i++)
    printf("IP addr %d: %s \n", i+1,
        inet_ntoa(*(struct in_addr*)host->h_addr_list[i]));
return 0;
```

실행 결과

```
root@my_linux:/tcpip# gcc gethostbyaddr.c -o hostaddr
root@my_linux:/tcpip# ./hostaddr 74.125.19.106
Official name: nuq04s01-in-f106.google.com
Address type: AF_INET
IP addr 1: 74.125.19.106
```

윈도우 기반 도메인 관련 함수

```
#include <winsock2.h>

struct hostent * gethostbyname(const char * name);
```

➡ 성공 시 hostent 구조체 변수의 주소 값, 실패 시 NULL 포인터 반환

```
#include <winsock2.h>

struct hostent * gethostbyaddr(const char * addr, int len, int type);
```

➡ 성공 시 hostent 구조체 변수의 주소 값, 실패 시 NULL 포인터 반환

전달인자의 이름에서만 차이가 있을 뿐 그 이외의 모든 것은 Linux의 함수와 동일함

소켓의 옵션과 입출력 버퍼의 크기

다양한 소켓의 옵션

Protocol Level	Option Name	Get	Set
SOL_SOCKET	SO_SNDBUF	O	O
	SO_RCVBUF	O	O
	SO_REUSEADDR	O	O
	SO_KEEPALIVE	O	O
	SO_BROADCAST	O	O
	SO_DONTROUTE	O	O
	SO_OOBINLINE	O	O
	SO_ERROR	O	X
	SO_TYPE	O	X
IPPROTO_IP	IP_TOS	O	O
	IP_TTL	O	O
	IP_MULTICAST_TTL	O	O
	IP_MULTICAST_LOOP	O	O
	IP_MULTICAST_IF	O	O
IPPROTO_TCP	TCP_KEEPALIVE	O	O
	TCP_NODELAY	O	O
	TCP_MAXSEG	O	O

- 소켓의 특성을 변경시킬 때 사용하는 옵션 정보들
- 이러한 소켓의 옵션은 계층별로 분류됨
- IPPROTO_IP 레벨의 옵션들은 IP 프로토콜에 관련된 사항들이며,
- IPPROTO_TCP 레벨의 옵션들은 TCP 프로토콜에 관련된 사항들
- SOL_SOCKET 레벨의 옵션들은 소켓에 대한 가장 일반적인 옵션들로 생각하면 됨

옵션 정보의 참조에 사용되는 함수

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

int getsockopt(int sock, int level, int optname, void *optval, socklen_t *optlen);
```

➜ 성공 시 0, 실패 시 -1 반환

- sock — 옵션확인을 위한 소켓의 파일 디스크립터 전달.
- level — 확인할 옵션의 프로토콜 레벨 전달.
- optname — 확인할 옵션의 이름 전달.
- optval — 확인결과의 저장을 위한 버퍼의 주소 값 전달.
- optlen — 네 번째 매개변수 optval로 전달된 주소 값의 버퍼크기를 담고 있는 변수의 주소 값 전달, 함수호출이 완료되면 이 변수에는 네 번째 인자를 통해 반환된 옵션정보의 크기가 바이트 단위로 계산되어 저장된다.

- **앞서 표에서 제시한 protocol level과 option name이 두 번째, 세 번째 인자로 전달되어, 해당 옵션의 등록 정보를 얻어 옴**

옵션 정보의 설정에 사용되는 함수

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

int setsockopt(int sock, int level, int optname, const void *optval, socklen_t optlen);
```

➜ 성공 시 0, 실패 시 -1 반환

- sock — 옵션변경을 위한 소켓의 파일 디스크립터 전달.
- level — 변경할 옵션의 프로토콜 레벨 전달.
- optname — 변경할 옵션의 이름 전달.
- optval — 변경할 옵션정보를 저장한 버퍼의 주소 값 전달.
- optlen — 네 번째 매개변수 optval로 전달된 옵션정보의 바이트 단위 크기 전달.

- **앞서 표에서 제시한 protocol level과 option name이 두 번째, 세 번째 인자로 전달하고, 해당 옵션의 등록 정보를 변경 함**

소켓의 타입정보(TCP or UDP)의 확인

예제: sock_type.c의 일부

```
tcp_sock=socket(PF_INET, SOCK_STREAM, 0);
udp_sock=socket(PF_INET, SOCK_DGRAM, 0);
printf("SOCK_STREAM: %d \n", SOCK_STREAM);
printf("SOCK_DGRAM: %d \n", SOCK_DGRAM);

state=getsockopt(tcp_sock, SOL_SOCKET, SO_TYPE, (void*)&sock_type, &optlen);
if(state)
    error_handling("getsockopt() error!");
printf("Socket type one: %d \n", sock_type);

state=getsockopt(udp_sock, SOL_SOCKET, SO_TYPE, (void*)&sock_type, &optlen);
if(state)
    error_handling("getsockopt() error!");
printf("Socket type two: %d \n", sock_type);
```

소켓의 타입정보는 변경이 불가능하기 때문에, 옵션 SO_TYPE은 확인만 가능하고 변경이 불가능한 옵션

실행결과

```
root@my_linux:/tcpip# gcc sock_type.c -o socktype
root@my_linux:/tcpip# ./socktype
SOCK_STREAM: 1
SOCK_DGRAM: 2
Socket type one: 1
Socket type two: 2
```

소켓의 입출력 버퍼 크기 확인

예제: get_buf.c의 일부

```
sock=socket(PF_INET, SOCK_STREAM, 0);
len=sizeof(snd_buf);
state=getsockopt(sock, SOL_SOCKET, SO_SNDBUF, (void*)&snd_buf, &len);
if(state)
    error_handling("getsockopt() error");

len=sizeof(rcv_buf);
state=getsockopt(sock, SOL_SOCKET, SO_RCVBUF, (void*)&rcv_buf, &len);
if(state)
    error_handling("getsockopt() error");

printf("Input buffer size: %d \n", rcv_buf);
printf("Output buffer size: %d \n", snd_buf);
```

입력버퍼의 크기를 참조 및 변경할 때에는SO_SNDBUF, 출력버퍼의 크기를 참조 및 변경할 때에는 SO_RCVBUF를 사용함

실행결과

```
root@my_linux:/tcpip# gcc get_buf.c -o getbuf
root@my_linux:/tcpip# ./getbuf
Input buffer size: 87380
Output buffer size: 16384
```

소켓의 입출력 버퍼 크기 변경

예제: get_buf.c의 일부(에러 처리 코드는 생략)

```
sock=socket(PF_INET, SOCK_STREAM, 0);
state=setsockopt(sock, SOL_SOCKET, SO_RCVBUF, (void*)&rcv_buf, sizeof(rcv_buf));
state=setsockopt(sock, SOL_SOCKET, SO_SNDBUF, (void*)&snd_buf, sizeof(snd_buf));
len=sizeof(snd_buf);
state=getsockopt(sock, SOL_SOCKET, SO_SNDBUF, (void*)&snd_buf, &len);
len=sizeof(rcv_buf);
state=getsockopt(sock, SOL_SOCKET, SO_RCVBUF, (void*)&rcv_buf, &len);
printf("Input buffer size: %d \n", rcv_buf);
printf("Output buffer size: %d \n", snd_buf);
```

실행결과

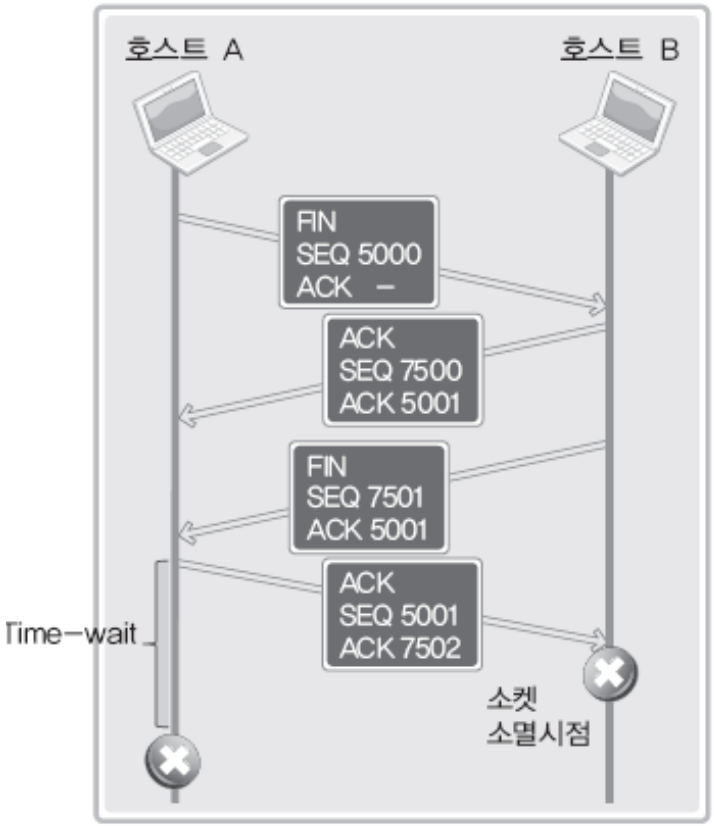
```
root@my_linux:/tcpip# gcc set_buf.c -o setbuf
root@my_linux:/tcpip# ./setbuf
Input buffer size: 6144
Output buffer size: 6144
```

입출력 버퍼는 상당히 주의 깊게 다뤄져야 하는 영역이기 때문에, 실행결과에서 보이듯이 코드에서 요구하는 바가 완벽히 반영되지는 않음

SO-REUSEADDR

주소 할당 에러의 원인 time-wait

SO_REUSEADDR 옵션을 설정하면 커널이 소켓을 사용하는 중에도 계속해서 사용할 수 있음 이 옵션은 서버프로그램이 종료된 후에도 커널이 소켓의 포트를 아직 점유 중인 경우에 서버 프로그램을 다시 구동해야 할 때 매우 유용함



Time-wait의 이해

서버, 클라이언트에 상관없이 TCP 소켓에서 연결의종료를 목적으로 Four-way handshaking의 첫 번째 메시지를 전달하는 호스트의 소켓은 Time-wait 상태를 거침

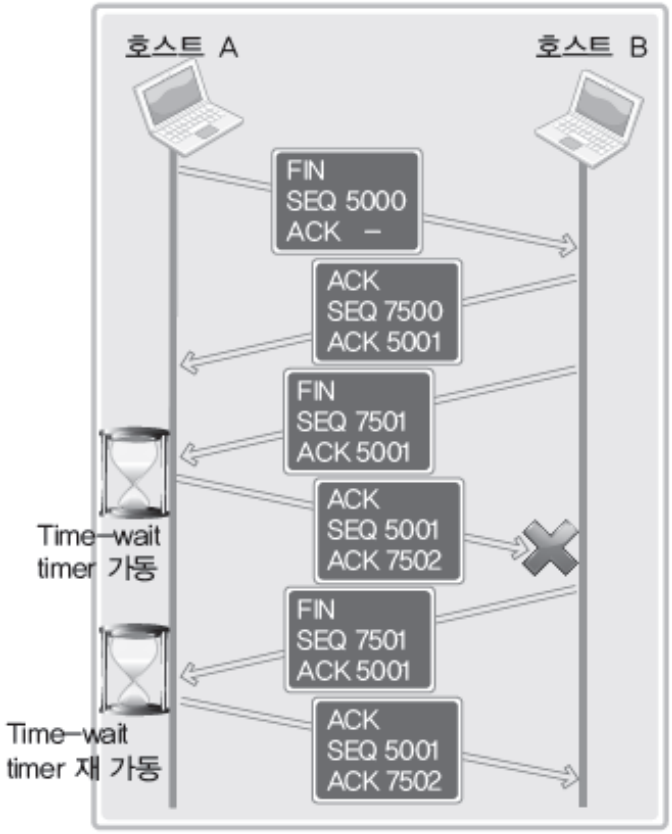
Time-wait 상태 동안에는 해당 소켓이 소멸되지 않아서 할당 받은 Port를 다른 소켓이 할당할 수 없음

Time-wait의 존재이유

왼쪽그림에서 호스트 A의 마지막 ACK가 소멸되는 상황을 대비해서 Time-wait 상태가필요함

호스트A의 마지막 ACK가 소멸되면, 호스트 B는 계속해서 FIN 메시지를 호스트 A에 전달하게 됨

주소의 재할당



Time-wait은 길어질 수 있음

Time-wait은 필요하나 실 서비스중인 서버의 경우Time-wait이 문제가 될 수 있음

그러한 경우에는 Time-wait 상태에 있는 Port의 할당이 가능하도록 코드를 수정해야 함

port 할당이 가능하도록 옵션을 변경

```
optlen=sizeof(option);
option=TRUE;
setsockopt(serv_sock, SOL_SOCKET, SO_REUSEADDR, (void*)&option, optlen);
```

윈도우 기반으로 구현하기

소켓의 옵션정보를 확인하는 getsockopt 함수

```
#include <winsock2.h>

int getsockopt(SOCKET sock, int level, int optname, char * optval, int * optlen);
```

➡ 성공 시 0, 실패 시 SOCKET_ERROR 반환

- sock — 옵션확인을 위한 소켓의 핸들 전달.
- level — 확인할 옵션의 프로토콜 레벨 전달.
- optname — 확인할 옵션의 이름 전달.
- optval — 확인결과의 저장을 위한 버퍼의 주소 값 전달.
- optlen — 네 번째 매개변수 optval로 전달된 주소 값의 버퍼 크기를 담고 있는 변수의 주소 값 전달, 함수호출이 완료되면 이 변수에는 네 번째 인자를 통해 반환된 옵션정보의 크기가 바이트 단위로 계산되어 저장된다.

Linux의 getsockopt 함수와 다르지 않음

소켓의 옵션정보를 확인하는 getsockopt 함수

```
#include <winsock2.h>

int setsockopt(SOCKET sock, int level, int optname, const char* optval, int optlen);
```

➜ 성공 시 0, 실패 시 SOCKET_ERROR 반환

- sock — 옵션변경을 위한 소켓의 핸들 전달.
- level — 변경할 옵션의 프로토콜 레벨 전달.
- optname — 변경할 옵션의 이름 전달.
- optval — 변경할 옵션정보를 저장한 버퍼의 주소 값 전달.
- optlen — 네 번째 매개변수 optval로 전달된 옵션정보의 바이트 단위 크기 전달.

Linux의 getsockopt 함수와 다르지 않음

다중 접속 서버의 구현 방법들

다중 접속 서버의 구현 방법들

• 멀티프로세스 기반 서버	다수의 프로세스를 생성하는 방식으로 서비스 제공
• 멀티플렉싱 기반 서버	입출력 대상을 묶어서 관리하는 방식으로 서비스 제공
• 멀티쓰레딩 기반 서버	클라이언트의 수만큼 쓰레드를 생성하는 방식으로 서비스 제공

다중 접속 서버란 둘 이상의 클라이언트에게 동시에 접속을 허용하여, 동시에 둘 이상의 클라이언트에게 서비스를 제공하는 서버를 의미함

프로세스와 프로세스의 ID

❑ 프로세스란?

- 간단한 정의로 실행 중인 프로그램을 뜻함
- 실행중인 프로그램에 관련된 메모리, 리소스 등을 총칭하는 의미
- 멀티프로세스 운영체제는 둘 이상의 프로세스를 동시에 생성 가능

❑ 프로세스 ID

- 운영체제는 생성되는 모든 프로세스에 ID를 할당함

```
wmlab@ubuntu:~$ ps au
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root       1012  0.0  0.0  15936  1820 tty1     Ss+  07:51   0:00 /sbin/agetty -
root       1019  3.0  2.7 301252 55924 tty7     Ss+  07:51   0:01 /usr/lib/xorg/
wmlab      2278  0.2  0.2  22376  4860 pts/17   Ss   07:51   0:00 bash
wmlab      2318  0.0  0.1  37364  3396 pts/17   R+   07:51   0:00 ps au
```

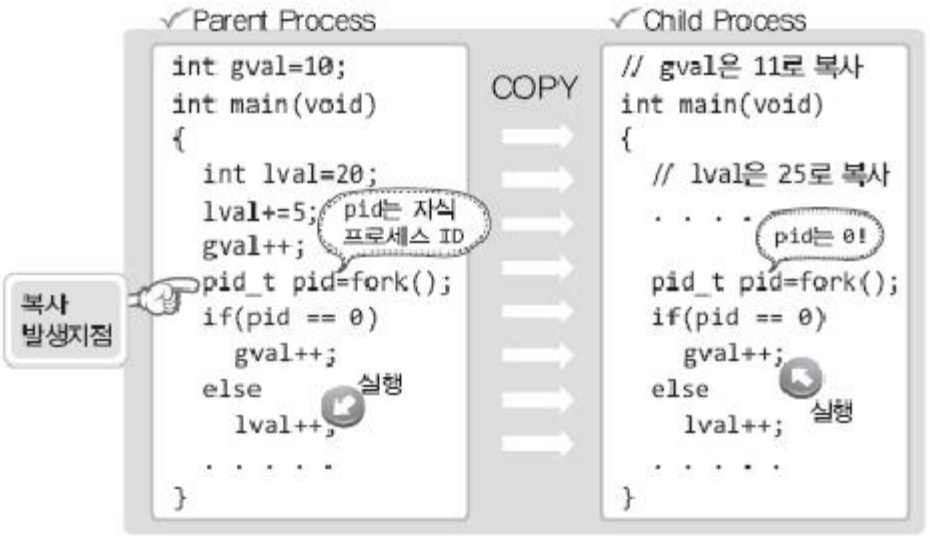
fork 함수의 호출을 통한 프로세스의 생성

```
#include <unistd.h>

pid_t fork(void);
```

➡ 성공 시 프로세스 ID, 실패 시 -1 반환

fork 함수가 호출되면, 호출한 프로세스가 복사되어 fork 함수 호출 이후를 각각의 프로세스가 독립적으로 실행하게 됨



fork 함수 호출 이후의 반환 값은 다음과 같음

- 부모 프로세스 fork 함수의 반환 값은 자식 프로세스의 ID
- 자식 프로세스 fork 함수의 반환 값은 0

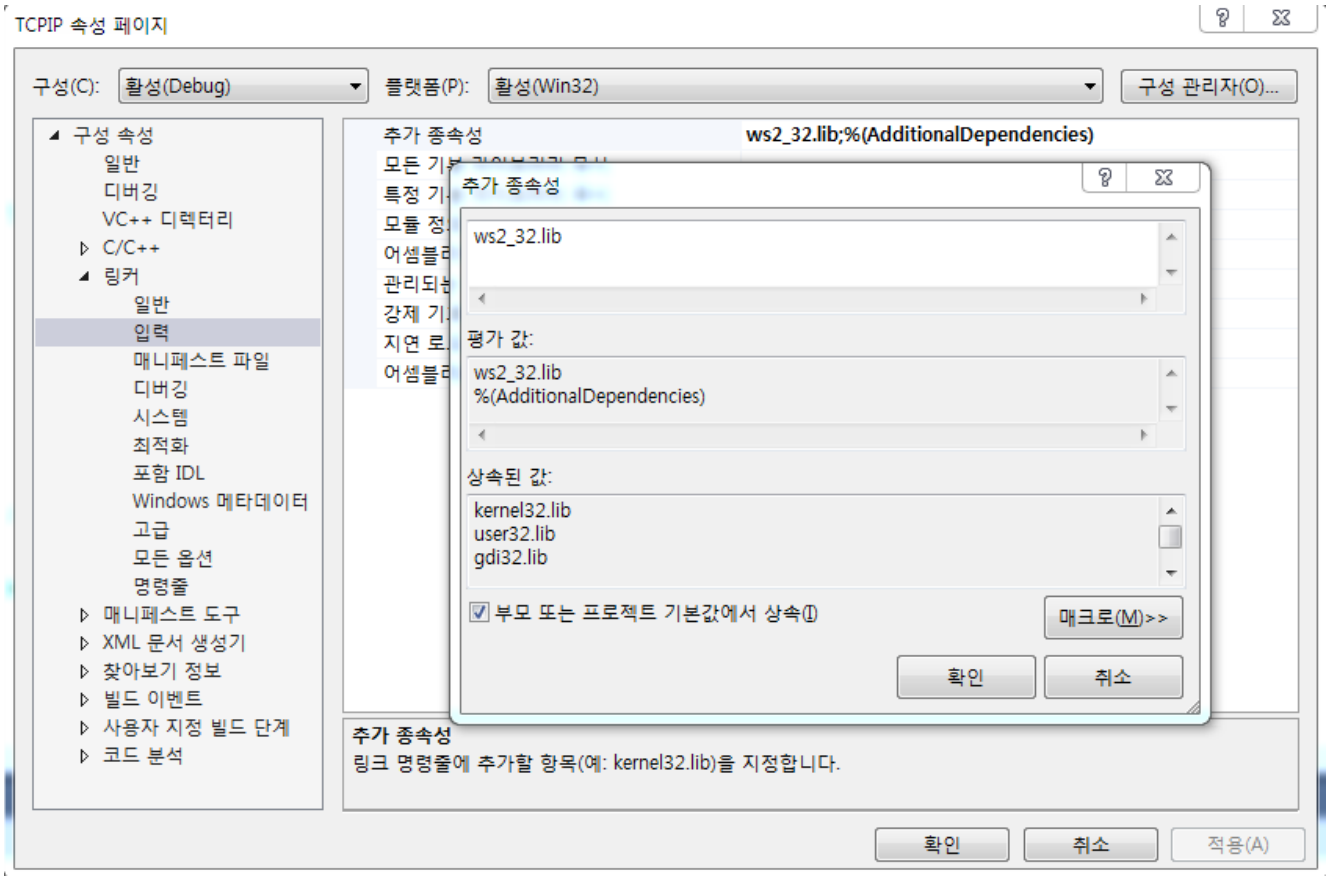
따라서 반환 값의 차를 통해 부모 프로세스와 자식 프로세스의 프로그램 흐름을 구분하게 됨

fork 함수를 호출한 프로세스는 부모 프로세스

fork 함수의 호출을 통해서 생성된 프로세스는 자식 프로세스

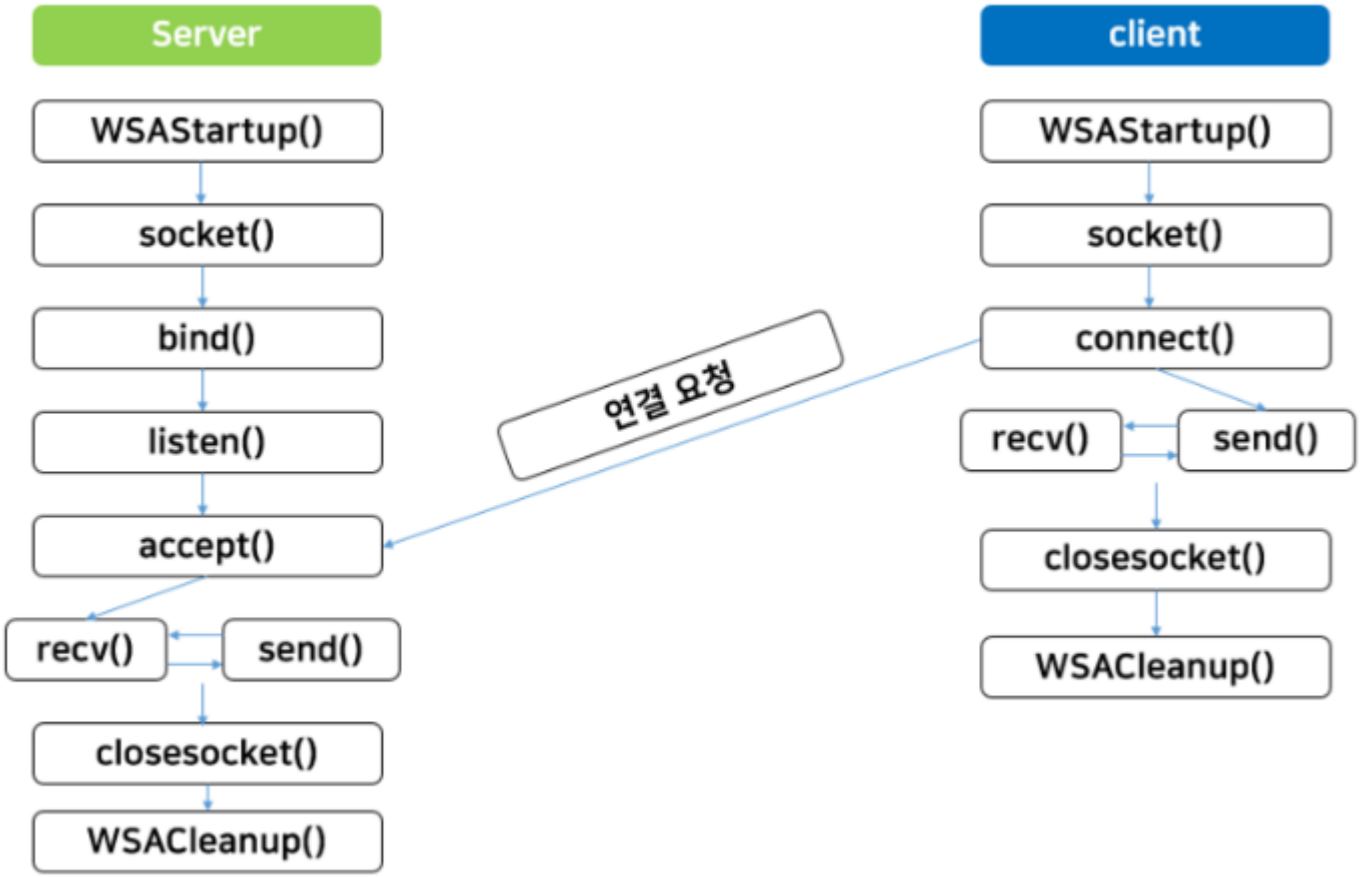
윈도우 소켓 프로그래밍

winsock2.h



(프로젝트 '속성'->'구성 속성'->'입력'->'추가 종속성'->ws2_32.lib 추가)

소켓 프로그래밍 함수의 사용



❑ server

- socket()함수를 통해서 서버의 소켓을 생성
- socket() 함수는 생성된 소켓이 TCP, UDP 등 기본적인 소켓에 대한 설정을 구성함

```
int     server_socket;
server_socket = socket( PF_INET, SOCK_STREAM, 0);
```

❑ server

- bind()함수를 통해서 서버가 웹 서버면(IP, 80) 이러한 방법으로 socket 함수를 사용하여 구성함
- 클라이언트가 서버의 위치를 알기 위한 IP와 Port의 등록작업이라 보면 됨
- socket 주소와 port를 할당하기 위해 sockaddr_in 구조체를 이용함

```
struct sockaddr_in server_addr;
memset( &server_addr, 0, sizeof( server_addr);
server_addr.sin_family      = AF_INET;              // IPv4 인터넷 프로토콜
server_addr.sin_port        = htons( 4000);         // 사용할 port 번호는 4000
server_addr.sin_addr.s_addr = htonl( INADDR_ANY);   // 32bit IPV4 주소

if( -1 == bind( server_socket, (struct sockaddr*)&server_addr, sizeof( server_a
ddr) ) )
{
   printf( "bind() 실행 에러\n");
   exit( 1);
}
```

❑ server

- listen() 함수를 통해 client의 connect() 함수의 요청을 확인하도록 요청함
- 확인된 요청에 대한 처리는 accept() 함수에게 넘김

```
listen( server_socket, 5)
```

- accept() 함수는 client의 요청에 대한 처리를 하며 접속이 허락될 시 통신을 하기 위한 소켓을 생성함 (접속된 client와 대화하기 위한 소켓을 생성)

```
int client_addr_size;
client_addr_size = sizeof( client_addr);

client_socket = accept( server_socket, (struct sockaddr*)&client_addr,&client_a
ddr_size);
```

❑ server

- recv() 함수를 통해 서버는 client가 send() 함수를 통해 데이터를 전송한 것을 받음
- 그리고 send() 함수를 통해 클라이언트에게 데이터를 전송함

```
int recv(int client_socket, void *buf, size_t len, int flags);
```

- closesocket() 함수를 통해 서버와 클라이언트의 소켓 연결을 종료함
- 종료된 소켓은 차후 다른 클라이언트의 접속을 받게 됨

Client

❑ client

- socket() 함수를 통해 client의 소켓을 생성하며 TCP, UDP인지 결정함

```
int     client_socket;
client_socket = socket( PF_INET, SOCK_STREAM, 0);
```

- connect() 함수를 통해서 listen() 상태의 서버에게 접속 요청을 시도함
- connect() 함수는 서버의 주소와 포트번호를 지정하기 때문에 지정된 서버의 주소와 포트번호로 연결을 시도함

```
struct sockaddr_in    server_addr;
memset( &server_addr, 0, sizeof( server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons( 4000);
server_addr.sin_addr.s_addr= inet_addr( "127.0.0.1");  // 서버의 주소
if( -1 == connect( client_socket, (struct sockaddr*)&server_addr, sizeof( serve
r_addr) ) )
{
   printf( "접속 실패\n");
   exit( 1);
}
```

❑ client

- recv() 함수를 통해서 서버에서 전송하는 데이터를 받거나 send() 함수를 통해서 데이터를 전송할 수 있음

```
int recv(int client_socket, void *buf, size_t len, int flags);
```

- closesocket() 함수를 통해 소켓 연결을 종료함

```
close( client_socket);
```

헤더파일

헤더파일

#include<stdio.h>

- include는 파이썬의 모듈과 같은 개념
- stdio는 standard input output에 대한 표준 입출력
  - printf, scanf, fopen등이 stdio.h에 포함됨

#include<stdlib.h>

- 문자열 변환, 동적 메모리 관리 등의 함수를 포함함(예로, malloc, free 함수가 있음)

#include<string.h>

- 메모리 블록, 문자열을 다를 수 있는 함수를 포함함
  - 예로 strcpy, memcpy, strlen, strcmp등 함수가 있음

#include<winsock2.h>

- **윈도우용 소켓 프로그래밍을 뜻하며 사용하기 전에 ws2_32.dll 파일을 초기화해야 함**
  - **WSAStartup() 함수**

헤더파일

```
void ErrorHandling(char *message);
```

ErrorHandling 함수를 사전에 정의한 뒤 후에 재정의함
- message라는 문자열을 인자로 받음
- WSACleanup() 함수를 실행

[재정의]

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

WinWock은 소켓 프로그래밍을 뜻하며, 이것을
사용하기 위해 ws2_32.dll 파일을 초기화 시켜줘야 함

```
int WSAStartup(
    WORD    wVersionRequested,    //WinSock 버전
    LPWSADATA    lpWSAData    //WSADATA 구조체의 포인터
);
//wVersionRequested 상위 바이트에는 마이너버전을 하위 바이트에는 메이저버전을 지정하며 통
상 MAKEWORD 매크로를 이용한다.
//SUCCESS Return 0
```

헤더파일

#include<conio.h>

- 주로 콘솔 입력/출력을 제공하기 위한 헤더 파일로 이 헤더는 프로그램에서 "콘솔 입력 및 출력"을 수행하기 위해 유용한 라이브러리 함수를 선언함
  - 대표적인 함수는 cgets, cprintf, cputs, getch 등이 있음

함수 부분

함수 부분

```
int WSAStartup(
    WORD    wVersionRequested,    //WinSock 버전
    LPWSADATA    lpWSAData    //WSADATA 구조체의 포인터
);
//wVersionRequested 상위 바이트에는 마이너버전을 하위 바이트에는 메이저버전을 지정하며 통
상 MAKEWORD 매크로를 이용한다.
//SUCCESS Return 0
```

WSASTartup() 함수는 두개의 인자를 받음

wVersionRequedsted: 사용할 윈속의 버전 값을 넣음

- 이러한 버전 값을 정의하기 위해 MAKEWORD 같은 매크로를 사용함

IPWSAData: WSADATA 구조체의 포인터 주소를 넣어주는데, 이 함수 (WSAStartup)가 리털이면

WSADATA 구조체의 인자에 세부 정보가 채워짐

함수 부분

```
typedef struct WSAData {
    WORD    wVersion;    //이 구조체가 사용될 것으로 예상되는 WinSock 버전
    WORD    wHighVersion;    //이 DLL이 지원하는 WinSock의 최신 버전
    char    szDescription[WSADESCRIPTION_LEN + 1];    //이 DLL의 설명문
    char    szSystemStatus[WSASYS_STATUS_LEN + 1];    //이 DLL의 현재 상태
    unsigned short    iMaxSockets;    //이용가능한 소켓의 최대 개수
    unsigned short    iMaxUdpDg;    //UDP 데이터그램이 송수신할 수 있는 최대 바이트수
    char FAR *    lpVendorInfo;    //공급자의 고유 정보
} WSADATA, *LPWSADATA
```

WSASTartup() 함수가 정상적으로 수행되면 리턴 값은 0이 반환되고 비정상적일 경우 에러코드 값을 반환

closesocket 함수가 소켓을 정상적으로 닫히게 해주는데, 이때 소켓은 전송 Queue에 걸려 있는 데이터가 존재하게 됨

즉, 이 함수를 사용하지 않을 경우 Queue에 잔존하는 데이터가 차후의 소켓을 재 연결했을 때 전송되는 것을 방지해줌

함수 부분

```
void main()
{
	WSADATA		wsaData;
	int		iResult;

	iResult = WSAStartup( MAKEWORD(2, 2), &wsaData );
	// 에러 여부
	if ( iResult == INVALID_SOCKET ) {
		printf( "WSAStartup failed with error %d\n", iResult );
		WSACleanup();
		return;
	}

	// 정상적으로 WSAStartup이 수행되었다면 실행할 루틴들을 적어준다..

	WSACleanup();		// 마지막엔 반드시 WSACleanup()을 호출해준다
}
```

WSAStartup 함수를 정의해야지만 윈소켓을 사용할 준비를 마칠 수 있음

함수 부분

- fputs 함수는 정의된 스트림에 문자열을 씀

```
int fputs ( const char * str, FILE * stream );
```

- str이 가리키는 문자열을 stream이 위치한 곳에 씀
- fputs의 특징은 str이 가리키는 문자열 '₩0에 도달 할 때까지 복사함
  - (보안상으로 오버플로우를 방지해주는 함수)
  - str: 스트림에 쓰여질 널 문자로 끝나는 문자 배열
  - stream: 문자열을 쓸 스트림의 객체를 가리키는 포인터

이 함수가 성공할 시 리턴 값은 음이 아닌 수가 리턴 되며 오류가 발생한다면 EOF(에러코드)를 리턴 함

함수 부분

fputc 함수는 한 문자를 쓰는 함수

```
int fputc ( int character, FILE * stream );
```

charater : 쓰여질 문자로 문자는 int로 형변환 되어 전달됨

stream : 문자가 쓰여질 스트림의 포인터

말그대로 사용자가 쓸 문자 charater의 한 문자를 stream 형식(stdin,stdout,stderr)에 의해 출력시켜줌

함수 부분

– _getch 함수

- 일반적인 함수는 입력을 받기 전까지는 잠시 프로그램이 정지하지만 _getch 및 kbhit 함수는 그에 대한 기능을 실행하고 입력이 없으면 그대로 프로그램이 진행되는 함수
- 차이점
  - _getch와 kbhit 함수의 차이는 getch가 버퍼를 사용하지 않고 바로 실행이 가능한 점
- 이 두 함수를 사용하기 위해서는 #conio.h 헤더가 필요함

```c
#include <stdio.h>
#include <conio.h>

int main()
{
    int ch;
        printf("문자 입력\n");
        printf("q키 종료\n");

        do
        {
                ch=_getch();
                printf("문자: %c, 아스키 코드: %d\n",ch,ch);
        }while(ch!='q');

}
```

함수 부분

```
WSADATA wsaData;
    SOCKET servSock, clntSock;   //SOCKET은 사실 UINT_PTR 형이다.
    SOCKADDR_IN servAddr, clntAddr;
```

WSADATA wsaData;는 WSADATA에 대한 구조체를 정의함
위에 언급했다시피 WSAStartup()함수를 사용하기 위해 사용됨

SOCKET 구조체

```
struct socket {
  socket_state state; //소켓 상태
  unsigned long flags; // 소켓 플래그
  const struct proto_ops * ops; // 프로토콜 소켓 작업
  struct fasync_struct * fasync_list; // 비동기, 동기
  struct file * file;
  struct sock * sk;
  wait_queue_head_t wait;
  short type; // 소켓 유형 SOCK_STREAM, SOCK_DGRAM (tcp,udp)
};
```

- 소켓 구조체를 통해서 bind, listen, accept, closesocket.. 등 모든 것을 이룸
- 소켓 구조체는 소켓 자체에 대한 정보를 지닌 구조체

함수

– SOCKADDR_IN servAddr, clntAddr, 의 SOCKADDR_IN 구조체

```
struct sockaddr_in {
    short   sin_family;   //주소 패밀리
    unsigned   short sin_port;   //포트 번호
    struct   in_addr sin_addr;   //IP주소
    char   sin_zero[8];   //패딩
};
```

- SOCKADDR_IN 구조체는 주소 체계를 정의함
- IP와 TCP/UDP 포트에 대한 포트 주소 정보를 지닌 구조체
  - sin_family: IPv4, IPv6에 대한 정보를 나타냄 이 인자로 들어가는 값의 예는 AF_INET이 있음
  - sin_addr, in_addr: 32비트 IP주소를 저장함
  - sin_port: 16비트 포트 번호를 지정함

❑ WSAStartup 함수

```
if(WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
    ErrorHandling("Load WinSock 2.2 DLL Error");
```

- MAKEWORD를 통해 2.2 버전을 지정하였으며, WSADATA 구조체를 가리키는 포인터 변수 및 wsaData를 지정함
- 이 함수의 반환 값이 0이면 정상적인 것
- 0이 아닐 시 ErrorHandling 함수를 호출

❑ WSAStartup 함수

```
if(WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
    ErrorHandling("Load WinSock 2.2 DLL Error");
```

위에서 언급한 WSAStartup 함수이다. MAKEWORD를 통해 2.2버전을 지정하였으며 WSADATA 구조체를 가리키는 포인터 변수 &wsaData를 지정하였다. 이 함수의 반환 값이 0이면 정상적인 것이며 0이 아닐 시 ErrorHandling 함수를 호출한다. ErrorHandling 함수에 대해선 위에서 언급하였다.

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

socket 함수에 대한 정의는 아래와 같다.

```
SOCKET socket(
    int   af,   //주소 패밀리
    int   type,   //소켓의 종류
    int   protocol   //프로토콜(Auto = 0)
);
```

❑ AF_INET

- Address Family의 인자는 주소 패밀리로서 일방적으로 인터넷을 사용하면 이 값을 대부분 AF_INET dla
- AF_INET은 인자 도메인 중 하나이며, 인자 도메인은 PF_INET과 AF_INET으로 나뉨
  - AF_INET – 주소체계, PF_INET – 프로토콜 체계
- 소켓 타입을 사용할지에 대한 인자로 값은 크게 두 가지로 나뉨
  - SOCK_STREAM = TCP 방식
    SOCK_DGRAM = UDP 방식

함수 부분

❑ AF_INET

- Address Family의 인자는 주소 패밀리로서 일방적으로 인터넷을 사용하면 이 값을 대부분 AF_INET dla
- AF_INET은 인자 도메인 중 하나이며, 인자 도메인은 PF_INET과 AF_INET으로 나뉨
  - AF_INET – 주소체계, PF_INET – 프로토콜 체계
- 소켓 타입을 사용할지에 대한 인자로 값은 크게 두 가지로 나뉨
  - SOCK_STREAM = TCP 방식
    SOCK_DGRAM = UDP 방식

```
SOCKET socket(
    int   af,    //주소 패밀리
    int   type,    //소켓의 종류
    int   protocol    //프로토콜(Auto = 0)
);
```

protocol
이 값을 0으로 설정할 시 통신에 알맞는 프로토콜을 자동으로 선정하여 사용

함수 부분

```
memset(&servAddr, 0, sizeof(SOCKADDR_IN));
    servAddr.sin_family = AF_INET;
    servAddr.sin_port = htons(atoi(argv[1]));
    servAddr.sin_addr.s_addr = htonl(INADDR_AN
Y);
```

memset을 사용하여 메모리에 값을 채우며 그 값은 소켓의 인자로 사용할 값을 정의

이 중 htons와 htonl 함수가 호출되는데 이 함수는 네트워크 바이트 순서(Network Byte Order)

- htons와 htonl 함수가 호출되는데 이 함수는 네트워크 바이트 순서(Network Byte Order)
- sockaddr_in 구조체 멤버에 값을 대입하기 위해선 대입할 값을 '네트워크 바이트' 순서로 변경한 다음 대입 시켜주어야 함

함수 부분

네트워크 바이트 순서란?

- 호스트 PC간 통신을 하고자 할 경우 CPU가 다르면 처리하는 데이터 방식이 다름
- (리틀엔디언 빅엔디언) 때문에, 이러한 처리 방식에 오류를 없애 주고자 네트워크 통신을 하고자 할 경우 네트워크 바이트 순서로 데이터를 전송함

네트워크 바이트 순서에 대한 대표적인 함수들은 아래와 같음

htonl()함수 : long intger(일반적으로 4byte)데이터를 네트워크 byte order로 변경

htons()함수 : short intger(일반적으로 2byte)데이터를 네트워크 byte order로 변경

ntohl()함수 : long intger 데이터를 호스트 byte order로 변경

ntohs()함수 : short intger 데이터를 호스트 byte order로 변경

반환 값은 네트워크 byte order 2바이트 값을 넘김

소켓통신을 위한 자료구조 (1/2)

- 윈도우 소켓 초기화 정보를 가지고 있는 구조체(정보 얻어 옴)

```
struct WSAData {
    WORD wVersion;    // 윈도우 소켓 버전
    WORD wHighVersion; // 윈도우 소켓의 가장 높은 버전. 통상적으로 wVersion과 같음
    char szDescription[WSADESCRIPTION_LEN+1]; //  NULL로 끝나는 문자열.
                                              //적재된 WS2_32.dll에서 소켓에 관련된 설명 문자열을 카피
    char szSystemStatus[WSASYSSTATUS_LEN+1]; // 시스템의 각종 상태를 알수 있게 해줌
    unsigned short iMaxSockets; // 어플리케이션에서 사용할 소켓의 최대 수 리턴. Version 2부터 무시
    unsigned short iMaxUdpDg; // 전송할 수 있는 데이터그램 최대 크기 리턴. Version 2부터 무시
    char FAR * lpVendorInfo; // Version 2부터 무시
};
```

- 소켓을 연결할 상대방 주소를 쓰는 구조체

```
struct sockaddr_in{
    short sin_family;
    unsigned short sin_port;
    struct in_addr sin_addr;
    char sin_zero[8];
};
```

소켓통신을 위한 자료구조 사용 예(2/2)

- Winsock 버전 2.2 정보를 wsadata에 얻어옴

```
WSADATA wsadata;      // 윈도우 소켓 정보를 저장할 wsadata 선언
WSAStartup(MAKEWORD(2,2), &wsadata);  // WS2_32.dll을 불러와 메모리 wsadata에 적재하는 함수
   // WSAStartup( ) 함수는 윈속 관련 라이브러리를 초기화하고, 그 데이터를 WSADATA에 할당
   // 1: 사용할 윈도우 소켓 버전, 2: 소켓 정보 저장 공간
```

- 소켓의 통신방법, 주소, 포트 설정

```
SOCKET s;                  // 소켓 사용할 s 선언
SOCKADDR_IN addr = { 0 };  // 소켓을 연결할 상대방 주소 정보에 사용될 구조체. 현재는 모두 0

s = socket(AF_INET, SOCK_STREAM, 0);    // 소켓 생성하고 사용은 s를 이용

addr.sin_family = AF_INET;
addr.sin_port = 20;
addr.sin_addr.s_addr = inet_addr("127.0.0.1");
```

127.0.0.1은 loopback 네트웍 접속을 위한 표준 IP어드레스 이다.

→ 127.0.0.1에 접속하고자 할때 바로 자신의 컴퓨터에 loopback 하게 된다.

→ 자신에게 데이터를 송신하는 것

19

소켓통신을 위한 함수(1/3)

- **Winsock 사용시작**

```
WSAStartup(MAKEWORD(2,2), &wsadata);
```

```
int WSAStartup(
    WORD wVersionRequested,  //사용할 윈도우즈 소켓버전(윈속 버전 정보 전달받음)
    LPWSADATA lpWSAData  //소켓 정보 저장공간 (WSADATA 구조체 변수의 주소 값 전달
);
```

- **소켓 생성**

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

```
SOCKET WSAAPI socket(
    int af,        //address family 를 명시함, AF_INET 이용
    int type,      // 소켓의 유형으로 SOCK_STREAM 이용
    int protocol   // 0을 입력
);
```

- **주소와 소켓을 연결하는 함수**

```
bind(s, (LPSOCKADDR)&addr, sizeof(addr));
```

```
BOOL bind (
    SOCKET s,     // 연결할 소켓
    const SOCKADDR* lpSockAddr,     // 소켓에 지정될 주소와
                                    // 포트번호를 포함하는 SOCKASSR구조체 주소
   int nSockAddrLen      // 주소 구조체 SOCKASSR의 크기
```

소켓통신을 위한 함수(2/3)

- 연결요구를 기다리는 함수

```
if (listen(s, 5) == -1) return 0;
```

```
BOOL listen(
    SOCKET s,   // 기다릴 소켓
    int nConnectionBacklog  // 대기할 수 있는 요구 최대 개수로 1~5
);
```

- 연결을 요구하는 함수

```
connect(s, (LPSOCKADDR)&addr, sizeof(addr));
```

```
BOOL connect(
    SOCKET s,  // 연결에 사용할 소켓
    const SOCKADDR* lpSockAddr, // 상대방의 주소 및 포트번호
    int nSockAddrLen // 주소 구조체 SOCKASSR의 크기
);
```

소켓통신을 위한 함수(3/3)

- 연결요구를 받아들이는 함수

```
SOCKET accept(
    SOCKET s, // 소켓에 대한 요구를 받아들임
    SOCKADDR* lpSockAddr,
    // 접속하는 상대방의 주소를 저장할 SOCKASSR구조체 주소
    int* lpSockAddrLen // 주소 구조체 SOCKASSR의 크기
);
```

```
accept(s, (LPSOCKADDR)&c_addr, &size);
```

- 소켓과 관련된 리소스를 해제하는 함수

```
int closesocket(SOCKET s);
```

```
closesocket(s);
WSACleanup();
```

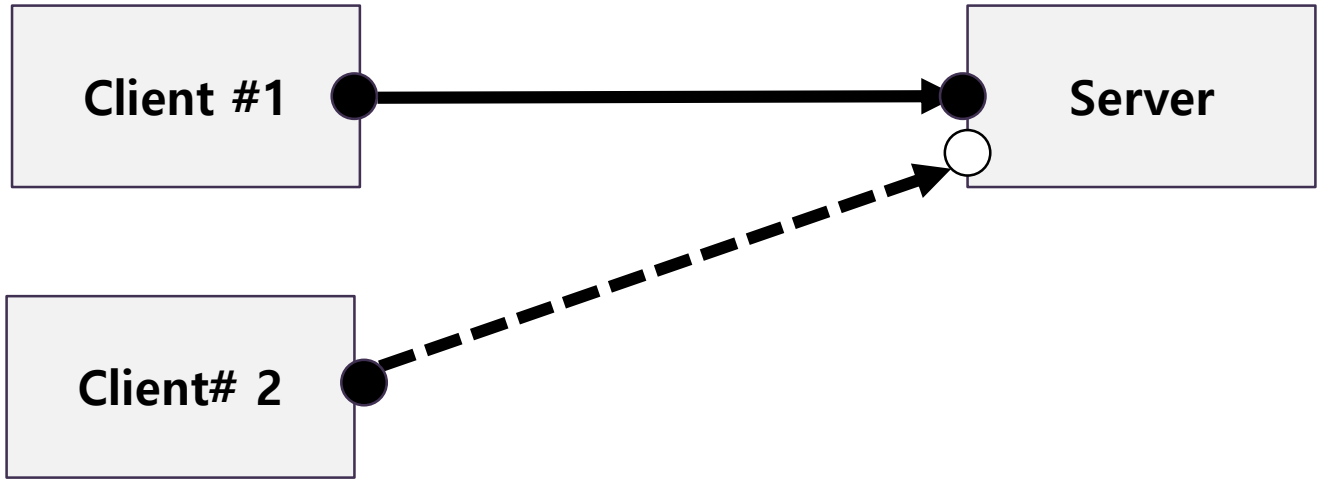
- Winsock 사용 끝

```
int WSACleanup( );
```

22

❑ 메시지 전공 전송 예제

– (로컬 호스트 환경에서의 테스트 구현)



컴퓨터 네트워크 실습 과제

- **소켓 프로그램을 코드로 작성하고, 소스 및 makefile을 만들어서 직접 컴파일 및 실행 후, 실행되는 것이 확인되면, zip 파일(레포트도 같이 넣어야 함)로 압축해서 연구실 홈페이지로 제출**
  - 제출 : imcoms@nate.com
  - 제출 파일 제목의 예) 컴네__학번_임덕선.zip
  - 평가 기준
    - 코드 작성시 코드 위에 주석을 꼭 달아서 제출하도록 하기

      (코드 시작부분에 이름 학번 넣기, 주석 없을 시 감점 -1)
    - 제출하는 코드는 컴파일 및 실행되는 소스여야 함 (실행되지 않을 시 -1점)
    - 제출기한 12월 11일 자정 (기한을 넘기면 -1점)