**CH01. 네트워크와 소켓 프로그래밍**

1. TCP/IP 프로토콜 개요

* TCP/IP 프로토콜 구조
  1. 호스트(host) : 최종 사용자 응용 프로그램을 수행하는 주체
     1. 인터넷에 연결된 PC, 노트북, 휴대전화, PDA 등
  2. 라우터(router) : 서로 다른 네트워크에 속한 호스트 간에 데이터를 교환할 수 있게하는 장비
  3. 통신 프로토콜(communication protocol) : 호스트와 라우터, 라우터와 라우터, 호스트와 호스트가 통신하기위한 정해진 절차와 방법
     + - Ex) TCP/IP
  4. TCP/IP
     1. 인터넷의 핵심 프로토콜
     2. ★운영체제의 일부로 구현
        + 애플리케이션은 운영체제의 일부로 구현한 TCP/IP 서비스 사용
        + 운영체제의 일부 기능을 불러 다가 설계하고 구현

|  |  |  |  |
| --- | --- | --- | --- |
| 응용 계층 | * + - * Telnet, FTP, HTTP, SMTP, 등 | user와 가까움 |  |
| 전송 계층 | * + - * TCP, UDP |  |
| 인터넷 계층 | * + - * IP |  |
| 네트워크 접근 계층 | * + - * 장치 드라이버 네트워크 하드웨어 | 네트워크 망에 가까움 |

* + 1. 계층적 구조
       - 1. 네트워크 접근 계층

역할 : 물리적 네트워크(망)를 통한 데이터 송수신

구성 요소 : 네트워크 하드웨어 + 장치 드라이버

주소 지정 방식 : 물리 주소 → ex) 이더넷

* + - * 1. 인터넷 계층

역할 : 네트워크 접근 계층의 도움을 받아 데이터를 목적지 호스트까지 전달

구성 요소 : IP 주소 + 라우팅(or 라우터)

주소 지정 방식 : IP 주소 → 유일 무이 한 주소

라우팅 : 데이터를 목적지까지 전달하는 일련의 작업

라우팅에 필요한 정보를 수집하는 작업과 라우팅 정보를 기초로 실제 데이터를 전달하는 작업도 포함

라우터(라우팅을 담당하는 전용 장비)를 통해 서로 다른 네트워크에 연결된 호스트가 통신할 수 있다

* + - * 1. 전송 계층

Handshake(응답 확인 방식)

(1) 상호 접수하는 신호를 개별 확인해 가면서 제어를 진행해가는 것. 시간에 대한 의존성이 없는 것이나 [시스템](https://terms.naver.com/entry.nhn?docId=853283&ref=y)에 적용하기 쉬운 것, 다양한 [입출력](https://terms.naver.com/entry.nhn?docId=851003&ref=y) 기기의 제어가 쉬운 것에서부터 [입출력 인터페이스](https://terms.naver.com/entry.nhn?docId=844790&ref=y)의 신호 형식에 적절히 널리 쓰이고 있다.  
(2) [중앙 처리 장치](https://terms.naver.com/entry.nhn?docId=858666&ref=y)와 [입출력 장치](https://terms.naver.com/entry.nhn?docId=844791&ref=y) 사이의 상태 조건을 결정하여 이에 따라 작동하는 것. 즉, 시스템 내의 기능 간의 상호 통신을 위해 필요한 [연속적인](https://terms.naver.com/entry.nhn?docId=818801&ref=y) 신호.

역할 : 최종 통신 목적지(응용 프로그램)를 지정하고, 오류 없이 데이터를 전송

* + - * 데이터 손실 또는 손상을 검출해 잘못된 데이터가 목적지에 전달되는 일을 방지

주소 지정 방식 : 포트 번호

대표 프로토콜

TCP

|  |  |
| --- | --- |
| TCP | UDP |
| 연결형 프로토콜   * 연결 설정 후 통신 가능 * Handshake * 수신자가 받을 준비가 되어있는지 확인 후 전송 | 비연결형 프로토콜   * 연결 설정 없이 통신 가능 * 수신자가 받을 준비가 되었는지 확인 없이 전송 (카톡) |
| 신뢰성 있는 데이터 전송   * 데이터를 재전송 함 * 데이터가 제대로 전송될 때까지 전송 | 신뢰성 없는 데이터 전송   * 데이터를 재전송하지 않음 * 깨진 데이터를 재전송 X |
| 1 : 1 통신 (unicast) | 1 : 1 통신, (unicast)  1 : 多 통신  (broadcast, multicast) |
| 데이터 경계 구분 안 함   * 바이트 스트림 | 데이터 경계 구분함   * 데이터그램 서비스 |

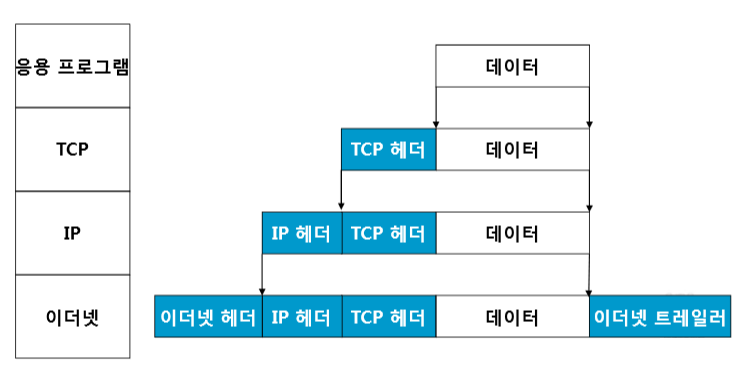
UDP

* + - * 1. 응용계층

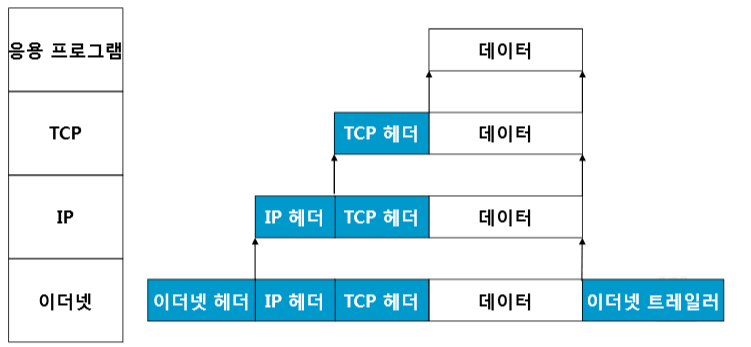
역할 : 전송 계층을 기반으로 한 다수의 프로토콜과 이 프로토콜을 사용하는 응용 프로그램

대표 프로토콜 : Telnet, FTP, HTTP, SMTP, 등

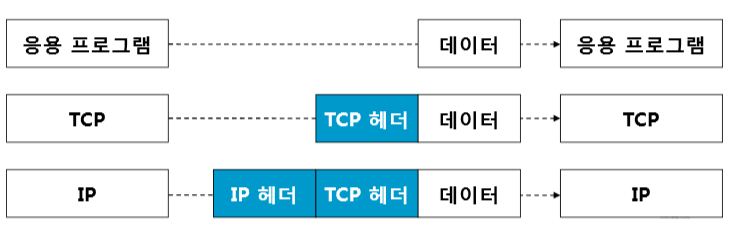
* 데이터 전송 원리
  1. 패킷(Packet)
     1. 각 프로토콜에서 정의한 제어 정보
     2. 제어 정보(IP주소, 포트 번호, 오류 체크 코드 등) + 데이터
     3. 제어 정보의 위치에 따라 앞쪽에 붙는 헤더(header)와 뒤쪽에 붙는 트레일러(trailer)로 구분
  2. 패킷 전송 형태
     1. 송신측



* + 1. 수신 측



* + 1. 계층별
       - 각 계층은 동일 위치의 상대 계층과 통신하는 것으로 간주



* + 1. 인터넷
       - 응용 계층, 전송 계층 : 하부 계층이 제공하는 가상적인 연결을 사용해 동작
       - 인터넷 계층 : IP주소와 라우팅 기능을 이용해 패킷 전송 경로 결정
       - 네트워크 접근 계층 : 물리 주소를 사용해 실제 패킷 전송
* IP주소, 포트번호
  1. IP 주소
     1. 인터넷에 있는 호스트와 라우터의 식별자
        + 폐쇄된 네트워크거나 IP를 공유하는 경우가 아니면 전 세계적으로 값이 유일
     2. IPv4 : 32bit, 8bit단위로 .(dot)로 구분하여 10진수 4개로 표기
        + Dotted-decimal notation
        + 147.46.114.70
     3. IPv6 : 128bit, 16bit 단위로 :(colon)으로 구분하여 16진수 8개로 표기
        + Colon-hexadecimal notation
        + 2001:0230:abcd:ffab:0023:eb00:ffff:1111
* 포트 번호
  1. 인터넷 통신의 종착점(하나 혹은 여러 프로세스)을 나타내는 식별자
  2. TCP와 UDP는 포트 번호로 부호 없는 16비트 정수를 사용하므로 0~65535 범위가 가능
  3. 영역별 포트 번호

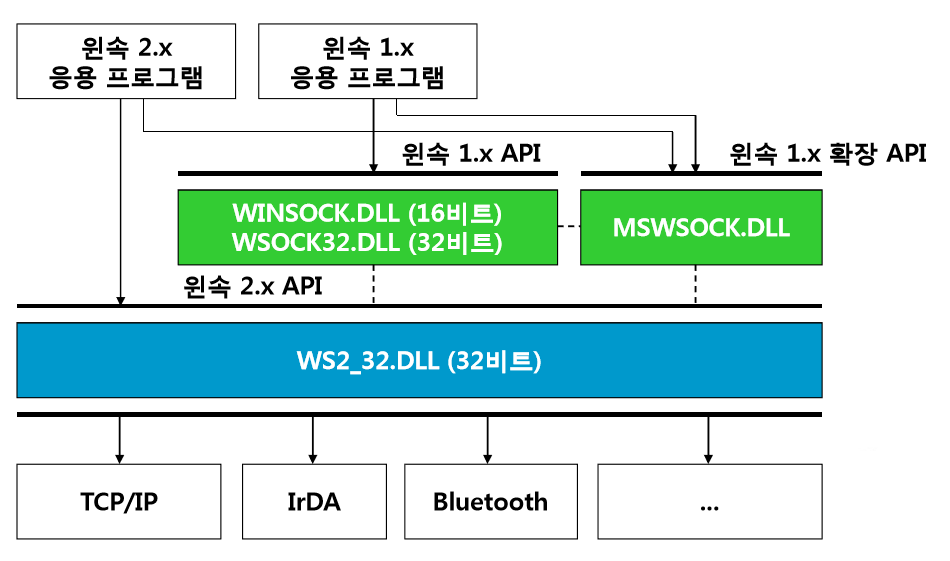


* 도메인 이름
  1. IP 주소에 대한 (기억하기 쉬운)별명
  2. 실제 통신할 때는 IP주소로 변환해야 함

IP 주소 -> 집주소

포트 번호 -> 받는 사람 이름

* 소켓의 개념
* TCP소켓과 UDP소켓은 다르게 생김
  1. 데이터 타입 (프로그래밍 관점)
     1. 파일 디스크립터 혹은 핸들과 유사한 개념
     2. 생성과 설정 과정이 끝나면 운영체제의 통신 관련 정보를 참조해 다양한 작업을 편리하게 할 수 있는 데이터 타입
  2. 통신 종단점 (응용 프로그램 관점)
     1. 응용 프로그램은 자신의 소켓이 상대편의 소켓과 연결된 것으로 생각하고 데이터를 주고받음
  3. 네트워크 프로그래밍 인터페이스 (TCP/IP 프로토콜 관점)
     1. 통신 양단이 모두 소켓을 사용할 필요는 없음
     2. TCP/IP 프로토콜에서 일반적으로 응용 계층과 전송 층 사이에 위치하는 것으로 간주
* 윈도우 소켓(윈속)
  1. 윈도우 소켓과 유닉스 소켓의 차이점
     1. 윈도우 소켓은 DLL을 통해 대부분의 기능이 제공되므로 DLL 초기화와 종료 작업을 위한 함수 필요
     2. 윈도우 프로그램을 대개 GUI를 갖추고 메시지 구동으로 동작하므로 이를 위한 확장 함수가 존재
     3. 윈도우는 운영체제 차원에서 멀티스레드를 지원하므로 멀티스레드 환경에서 안정적으로 동작하는 구조와 이를 위한 함수가 필요
  2. 윈속의 장점
     1. 유닉스 소켓과 소스 코드 수준에서 호환성이 높으므로 기존 코드를 이식하여 활용하기 쉬움
     2. 가장 널리 사용하는 네트워크 프로그래밍 인터페이스므로 한번 배우면 여러 운영체제(윈도우, 리눅스 등)에서 사용할 수 있음
     3. TCP/IP 외의 프로토콜도 지원하므로 최소 코드 수정으로 응용 프로그램이 사용할 프로토콜을 변경할 수 있음
     4. 비교적 저수준(low-level or mid-level)프로그래밍 인터페이스므로, 세부 제어가 가능하며 고성능 네트워크 프로그램을 개발할 수 있음
  3. 윈속의 단점
     1. 응용 프로그램 수준의 프로토콜을 프로그래머가 직접 설계해야 함
        + 주고받는 데이터 형식이나 전송 절차 등을 고려해 프로그래밍해야 하며, 설계 변경 시에는 코드 수정이 불가능
     2. 서로 다른 바이트 정렬 방식을 사용하거나 데이터 처리 단위가 서로 다른 호스트끼리 통신할 경우, 응용 프로그램 수준에서 데이터 반환을 처리해야 함
  4. 윈속 버전 : 1.1, 2.2
  5. 윈속 구조



* 루프백 주소
  1. 시스템 자신을 나타내는 의미로 내부적으로만 사용
  2. IPv4 : 127.0.0.1
  3. IPv6 : 0:0:0:0:0:0:0:1

**CH02. 윈도우 소켓 시작하기**

* 윈속 함수 오류 처리
  1. 오류 처리 유형
     1. 오류를 처리할 필요가 없는 경우 : 리턴 값이 없거나 호출 시 항상 성공하는 일부 소켓 함수
     2. 리턴 값만으로 오류를 처리하는 경우 : WSAStartup() 함수
     3. 리턴 값으로 오류 발생을 확인하고, 구체적인 내용은 오류 코드로 확인하는 경우 : Int WSAGetLastError(void)
  2. 오류 코드 얻기
     1. WSAGetLastError()함수
     2. WSAGetLastError()함수의 리턴 값을 바로 표시하면 오류코드 의미 해석이 불편하므로 해당 오류 코드를 적절한 문자열 형태로 출력
     3. FormatMessage()함수를 사용하면 오류 코드에 대응하는 오류 메시지를 얻을 수 있음
  3. FormatMessage()함수
     1. DWORD dwFlags : 옵션
     2. LPCVOID lpSource : NULL
     3. DWORD dwMessageId : 오류 코드, WSAGetLastError()함수의 리턴 값을 여기에 넣음
     4. DWORD dwLanguageld : 오류 메시지를 표시할 언어
        + MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT)를 사용하면 제어판에서 설정한 기본언어로 오류 메시지 출력
     5. LPTSTR lpBuffer : 오류 메시지의 시작 주소
     6. DWORD nSize : 0
     7. Vs\_list\* Arguments : NULL
* 윈속 초기화와 종료
  1. 윈속 응용 프로그램의 공통 구조



* 1. 윈속 초기화
     1. 모든 윈속 프로그램은 소켓 함수를 호출하기 전에 반드시 윈속 초기화 함수인 **WSAStartup ()**함수를 호출해야 함
     2. WSAStartup() 함수는 프로그램에서 사용할 윈속 버전을 요청함으로써 윈속 라이브러리인 WS2\_32.DLL을 초기화하는 역할 수행
        + Int WSAStartup(윈속 버전, WSAData) return 0 or 오류코드
  2. 윈속 종료
     1. 프로그램을 종료할 때는 윈속 종료 함수인 **WSACleanup()**을 호출
        + WSAStartup()함수를 두 번 이상 호출하는 것도 가능하지만 함수를 호출한 횟수만큼 WSACleanup()함수를 호출해야함
     2. WSACleanup()함수는 윈속 사용을 중지함을 운영체제에 알리고, 관련 리소스를 반환하는 역할 수행
     3. WSAGetLastError()함수를 호출함으로써 구체적인 오류 코드를 획득
* 소켓 생성과 닫기
  1. 소켓 생성하기
     + - TCP or UDP사용 여부를 결정하고 서로 약속
       - SOCKET socket(주소 체계 지정, 소켓 타입 지정, 사용할 프로토콜 지정)
       - return 성공 : 새로운 소켓, 실패 : INVALID\_SOCKET

사용자가 요청한 프로토콜을 사용해 통신할 수 있도록 내부적으로 리소스를 할당하고, 이에 접근할 수 있는 일종의 핸들 값을 리턴

* + - * 이 값을 소켓 디스크립터(socket descriptor)라 부르며, 각종 소켓 함수를 호출할 때 인자로 전달해 사용한다.
    1. 주소 체계
       - 통신 상대를 유일하게 지정할 수 있는 주소 지정 방법

|  |  |
| --- | --- |
| #define **AF\_INET** 2 | Internetwork : UDP, TCP, etc |
| #define AF\_INET 23 | Internetwork Version 6 |
| #define AF\_IRDA 26 | IrDA |
| #define AF\_BTH 32 | Bluetooth RFCOMM/L2CAP protocols |

* + 1. 소켓 타입
       - 사용할 프로토콜의 특성을 나타내는 값

|  |  |
| --- | --- |
| SOCK\_STREAM | 신뢰성 있는 데이터 전송 기능 제공, TCP |
| SOCK\_DGRAM | 신뢰성 없는 데이터 전송 기능 제공, UDP |

* 1. 소켓 닫기
     1. 소켓을 닫고 관련 리소스를 반환
     2. closesocket(소켓) : 해당 소켓을 닫고 관련 리소스를 반환
        + 성공 : 0, 실패 : SOCKET\_ERROR

**CH03. 소켓 주소 구조체 다루기**

* 소켓 주소 구조체
  1. 소켓 주소 구조체
     1. 네트워크 프로그램에서 필요로 하는 주소 정보를 담고 있는 구조체로, 다양한 소켓 함수의 인자로 사용
     2. 프로토콜 체계에 따라 다양한 형태가 존재
        + TCP/IP : SOCKADDR\_IN
        + IrDA : SOCKADDR\_IRDA
     3. 기본형은 SOCKADDR 구조체임
  2. SOCKADDR 구조체 – 기본형

Typedef struct sockaddr{

u\_short sa\_family // 주소 체계를 나타내는 16비트 정수 값

char sa\_data[14] // 해당 주소 체계에서 사용할 주소 정보

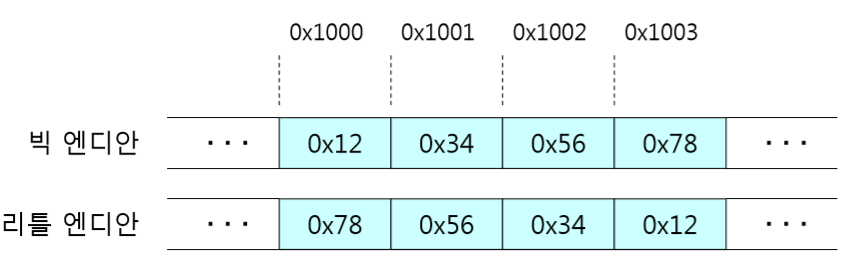
} SOCKADDR; 주소 체계에 따라 필요한 정보가 상이하므로

바이트 배열로 선언

* 바이트 정렬 함수

바이트 정렬 : 주소 받고 저장하는 방식이 pc마다 달라서 문제가 생겨 그 문제를 해결하기 위해 사용

* 1. 메모리에 데이터를 저장할 때 바이트 순서
     1. 빅 엔디안(big-endian) : 최상위 바이트부터 차례로 저장하는 방식
     2. 리틀 엔디안(little-endian) : 최하위 바이트부터 차례로 저장하는 방식
     3. 시스템에서 사용하는 바이트 정렬 방식은 CPU와 운영체제에 따라 상이함
        + 파일에 데이터를 저장하고 읽어오는 경우나 네트워크를 통해 데이터를 송신하고 수신하는 경우에는 바이트 정렬 방식에 유의해야 한다.



* 1. 바이트 정렬 방식을 고려해야 하는 경우
     1. IP주소에서 문제가 발생하는 상황 : IP주소 -> 빅 엔디안
     2. 포트 번호에서 문제가 발생하는 상황 : 포트번호 -> 빅 엔디안
        + (1+2) 프로토콜 구현을 위해 필요한 정보 :

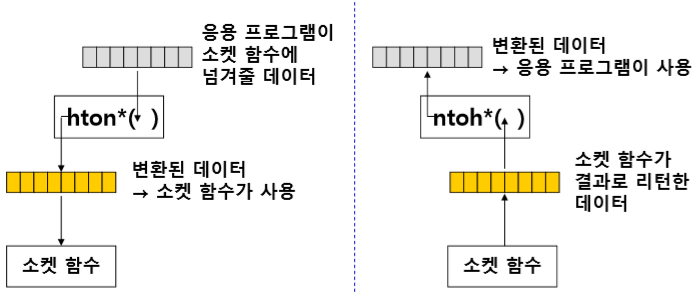
호스트와 라우터가 IP주소의 바이트 정렬 방식을 약속하지 않으면 IP주소 해석이 달라져 라우팅에 문제 발생

* + 1. 응용프로그램 데이터에서 문제가 발생하는 상황 : 빅 또는 리틀
       - 응용 프로그램이 주고 받는 데이터 : 서버와 클라이언트 동시에 개발시에는 바이트 정렬 방식을 통일하고 클라이언트만 개발시에는 서버의 바이트 정렬 방식을 따른다

네트워크 바이트 정렬 (network byte ordering) : 빅 엔디안 방식

호스트 바이트 정렬 (host byte ordering) : 시스템이 사용하는 고유한 바이트 정렬 방식

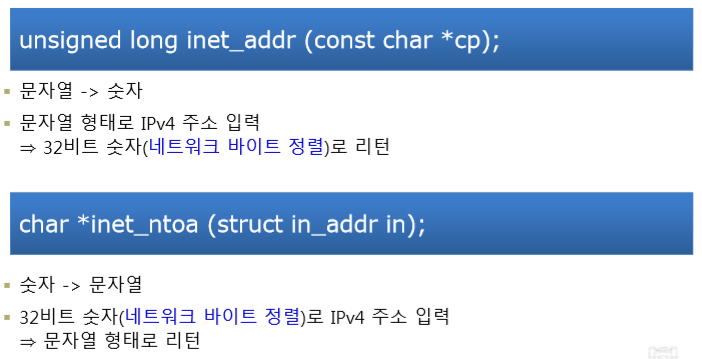
* 1. 바이트 정렬 함수 사용 상황
     1. hton\*()함수는 응용프로그램이 소켓 함수에 데이터를 넘겨주기 전에 호출
     2. ntoh\*()함수는 소켓 함수가 결과로 리턴한 데이터를 응용 프로그램이 출력 등의 목적으로 이용하기 전에 호출

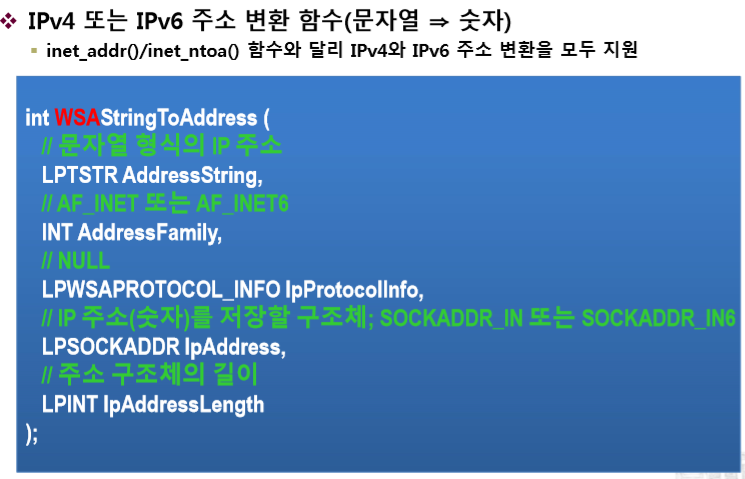


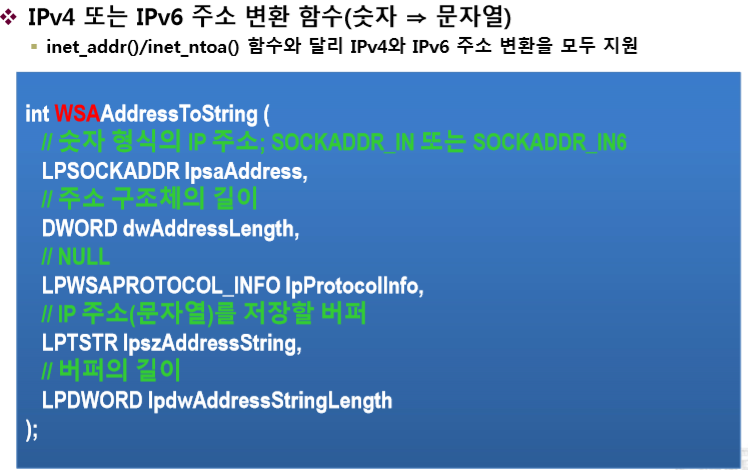
* 1. SOCKADDR\_IN 구조체의 바이트 정렬 방식



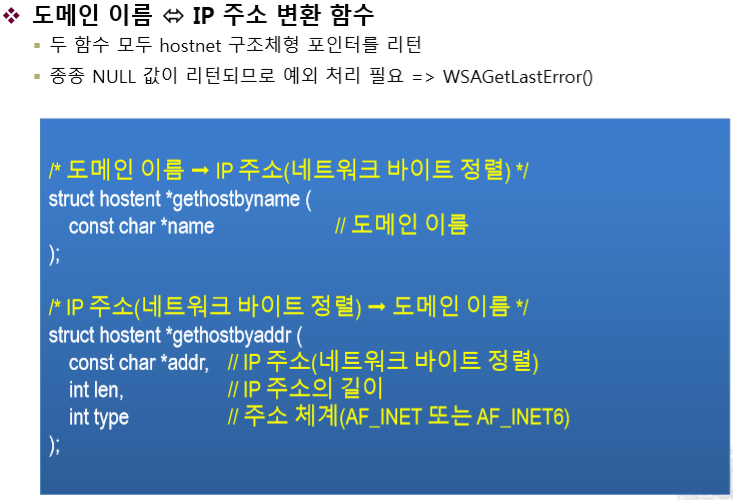
* IP 주소 변환 함수



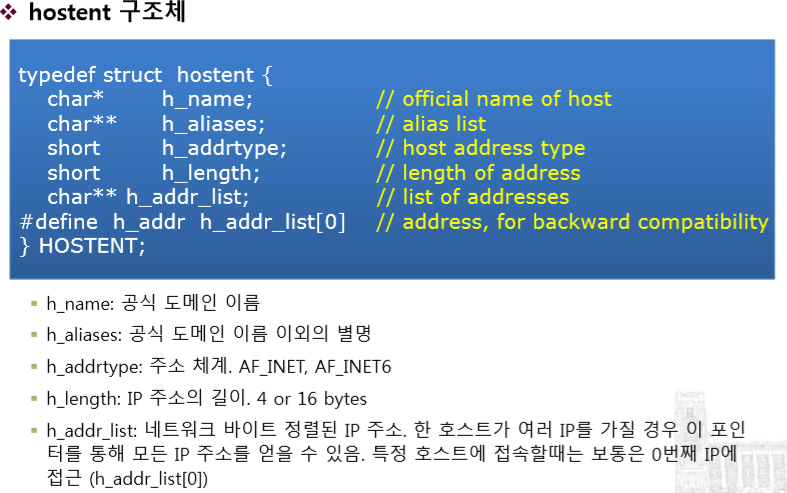


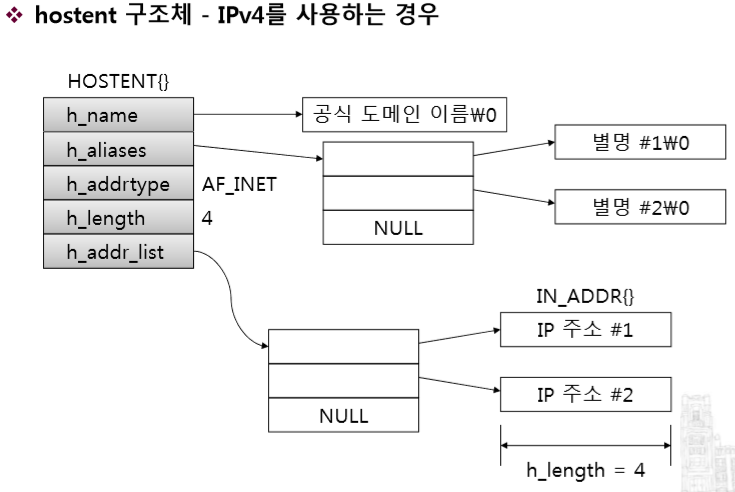


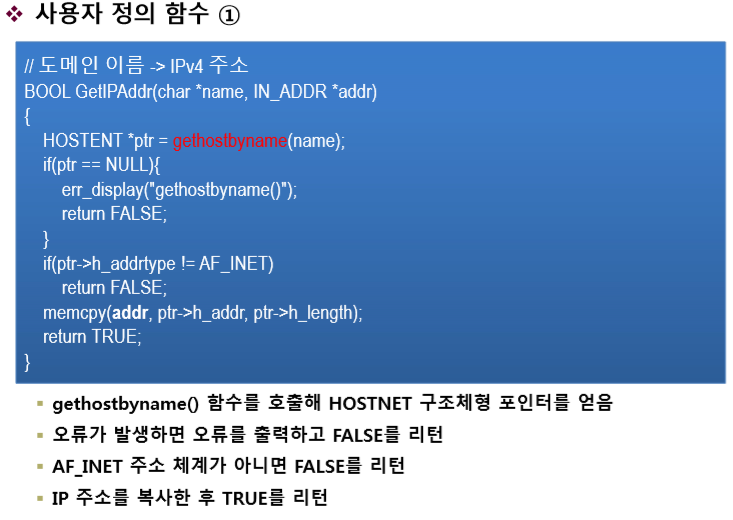
* 1. ping + 명령행 인자(IP 주소 or 도메인 주소)
     + - 호스트 대기 시간을 알려주는 명령어
  2. 127.0.0.1 (loop back 주소)
     1. 네트워크를 타고 망 밖으로 나가는 것이 없음
        + 인터프로세스 커뮤니케이션 테스팅
  3. 소켓 주소 구조체에 넣기 전에 2진수로 바꿔야함
     + - 자동으로 바꿔 주기 때문에 빅 엔디안으로 바꿀 필요 X
* 도메인 이름 시스템과 이름 변환 함수
  1. 도메인 이름
     1. IP주소보다 기억하고 사용하기 쉬움
     2. IP주소처럼 호스트나 라우터의 고유한 식별자로 사용
     3. 도메인 이름을 IP주소로 바꿔야함
     4. Ping + 도메인네임
        + 도메인 네임 서버가 해당하는 IP주소를 넘겨 줌
        + 그 IP주소를 받아서 2진수로 바꿔야 함

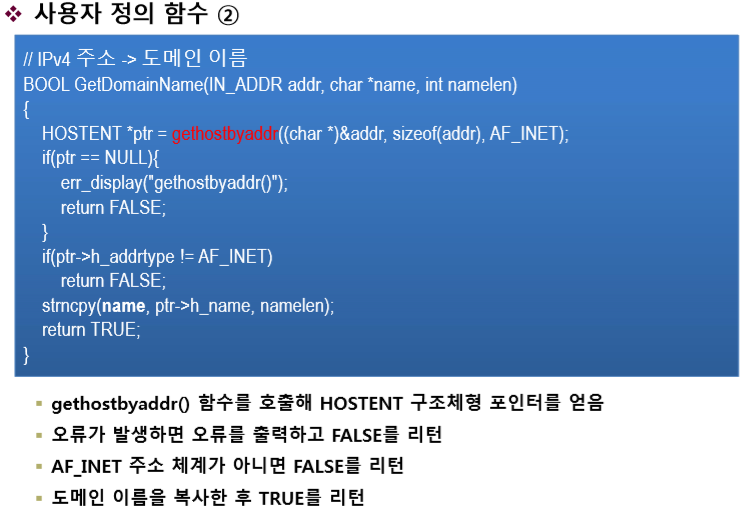


* 1. gethostbyaddr()
     1. 주소 체계를 넣어 주어야 함
        + V4랑 v6를 상관없이 사용 가능한 함수이기 때문에



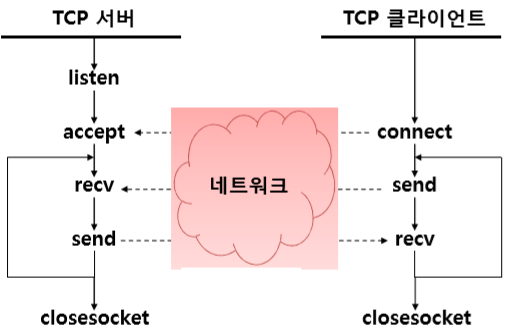


****

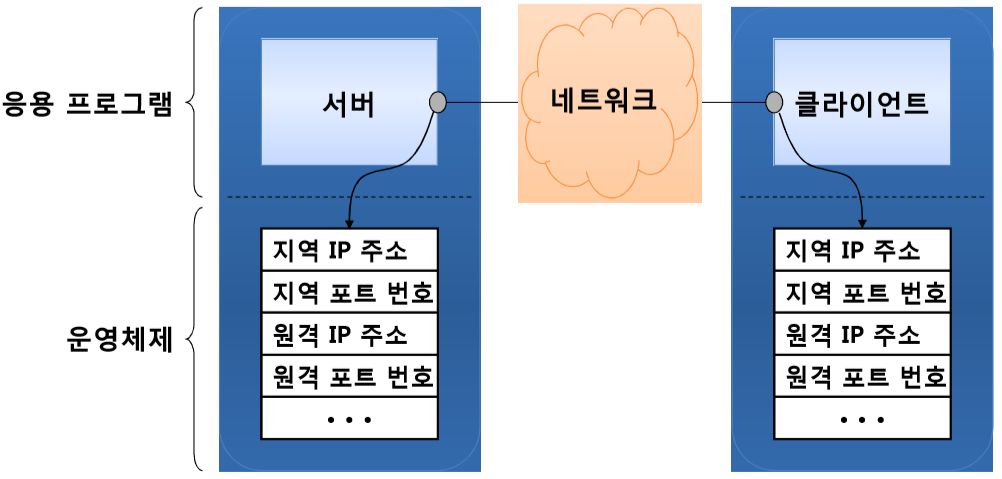
****

**CH04. TCP 서버-클라이언트**

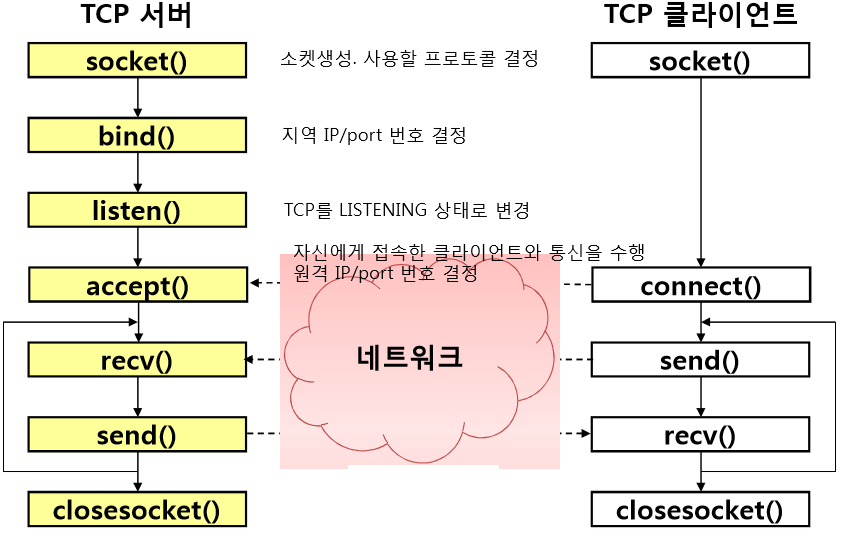
* TCP 서버-클라이언트 동작
  1. HTTP는 TCP에 기반한 프로토콜이므로 웹서버-클라이언트는 대표적인 TCP 서버-클라이언트 응용 프로그램이라고 할 수 있다.
  2. TCP 서버-클라이언트 핵심 동작



* + 1. 서버는 클라이언트가 접속하기를 대기 (listen)
    2. 클라이언트는 서버에 접속 (Connect)
    3. 클라이언트는 데이터를 서버로 전송 (send)
    4. 서버는 클라이언트 접속을 수용 (accept)
    5. 클라이언트가 보낸 데이터를 수신 (recv) 후 처리
    6. 서버는 처리한 데이터를 클라이언트에 전송 (send)
    7. 클라이언트는 서버가 보낸 데이터를 수신 (recv)
    8. 접속 종료 (closesocket)
* TCP 서버-클라이언트 분석
  1. 소켓 통신을 위해 결정해야 할 요소
     1. 프로토콜
        + 통신 규약, 소켓을 생성할 때 결정
     2. 지역 IP주소와 지역 포트 번호
        + 서버 또는 클라이언트 자신의 주소
     3. 원격 IP 주소와 원격 포트 번호
        + 서버 또는 클라이언트가 통신하는 상대의 주소
* TCP 응용 프로그램의 두 가지 종료 방식
  1. 정상 종료 : 사용하던 포트는 일정 시간(5분 이내) TIME\_WAIT 상태에 있다가 사라짐
  2. 비정상 종료(= 강제종료) : 사용하던 포트는 TIME\_WAIT 상태를 거치지 않고 사라짐
* 소켓 데이터 구조체

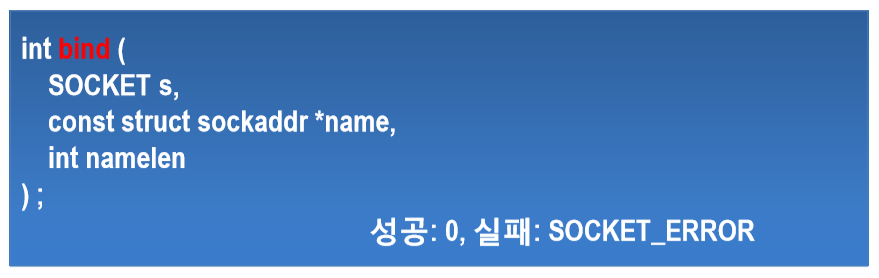


* 1. 소켓 함수는 지역 주소와 원격 주소를 결정하고 TCP 상태를 변경하기 위한 일련의 절차
  2. 데이터 전송 함수는 양쪽에서 사용
* TCP 서버 함수



**↕데이터 전송 함수**

* 1. bind() 함수
     1. 소켓의 지역 IP 주소와 지역 포트 번호를 결정



**클라이언트 접속을 수용할 목적으로 만든 소켓**

**소켓 주소 구조체를 지역 IP 주소와 지역 포트 번호로 초기화하여 전달**

**소켓주소 구조체의 길이**

* 1. listen() 함수
     1. 소켓의 TCP 포트 상태를 LISTENING으로 변경



**클라이언트 접속을 수용할 목적으로 만든 소켓**

**서버가 당장 처리하지 않더라도 접속 가능한 클라이언트 개수**

* 1. accept() 함수
     1. 접속한 클라이언트와 통신할 수 있도록 새로운 소켓을 생성해서 리턴
     2. 접속한 클라이언트의 IP주소와 포트 번호를 알려줌

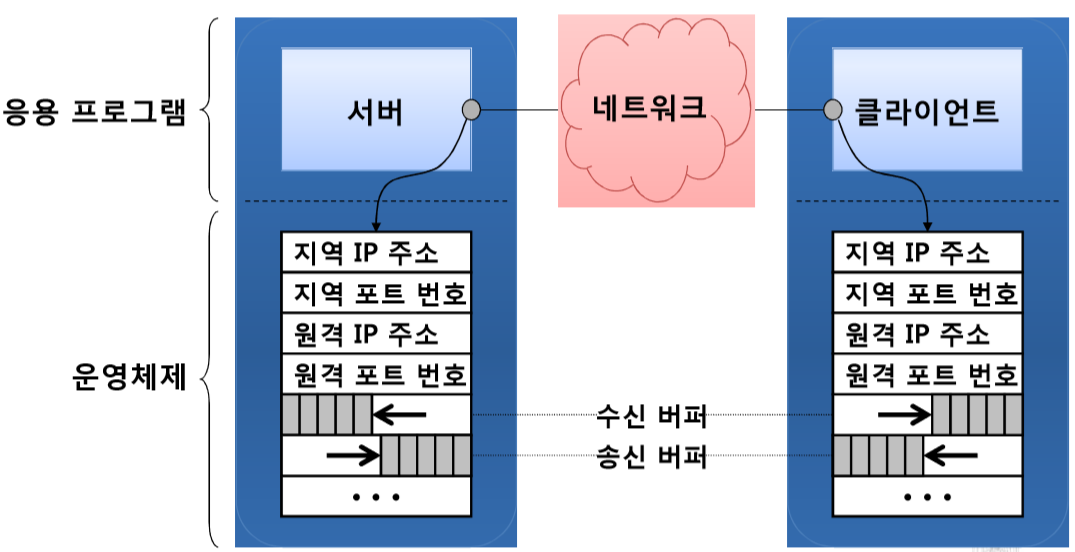


**클라이언트 접속을 수용할 목적으로 만든 소켓**

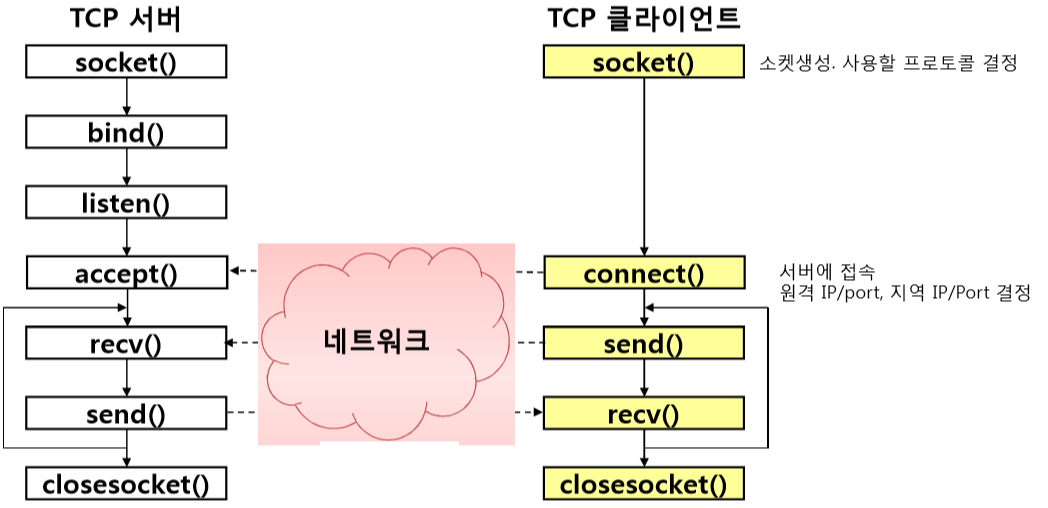
**접속한 클라이언트의 주소 정보**

**정수형 변수를 addr이 가리키는 소켓 주소 구조체의 크기**

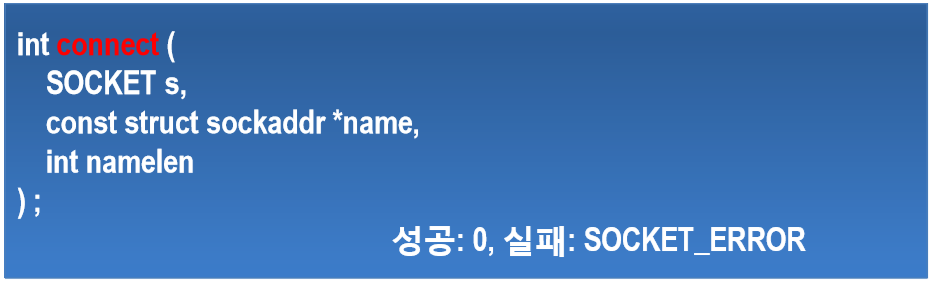
* 소켓 데이터 구조체



* TCP 클라이언트 함수



* 1. connect() 함수
     1. TCP 프로토콜 수준에서 서버와 논리적 연결을 설정
     2. 서버와 달리 bind()함수를 호출하지 않고 connect()함수를 호출하면, 운영체제가 자동으로 지역 IP주소와 지역 포트번호를 할당해줌
        + 자동으로 할당되는 포트 번호는 윈도우 버전에 따라 다름



**클라이언트 접속을 수용할 목적으로 만든 소켓**

**소켓 주소 구조체를 서버 주소로 초기화하여 전달**

**소켓주소 구조체의 길이**

* 1. send() 함수
     1. 응용 프로그램 데이터를 운영체제의 송신 버퍼에 복사함으로써 데이터를 전송
     2. send() 함수는 첫 번째 인자로 전달하는 소켓의 특성에 따라 다음과 같이 두 종류의 성공적인 리턴을 할 수 있다
        + 블로킹(blocking) 소켓 : 기본적인 소켓. 블로킹 소켓을 대상으로 send()함수를 호출하면, 송신 버퍼의 여유 공간이 send()함수의 세 번째 인자인 len보다 작을 경우 해당 프로세스는 대기 상태가 된다. 송신 버퍼에 충분한 공간이 생기면 프로세스는 깨어나고 len 크기만큼 데이터 복사가 일어난 후 send()함수가 리턴(값 = len)한다.
        + 넌블로킹(nonblocking) 소켓 : ioctlsocket()함수를 이용하면 사용 가능. 넌블로킹 소켓을 대상으로 send()함수를 호출하면, 송신 버퍼의 여유 공간만큼 데이터를 복사한 후 실제 복사한 바이트 수를 리턴(최소 1, 최대 len)한다.



**클라이언트 접속을 수용할 목적으로 만든 소켓**

**보낼 데이터를 담고 있는 응용 프로그램의 버퍼 주소**

**보낼 데이터 크기**

**send()함수의 동적을 바꾸는 옵션. 대부분 0 사용**

* 1. recv() 함수
     1. 운영체제의 수신 버퍼에 도착한 데이터를 응용 프로그램 버퍼에 복사
     2. recv()함수는 다음 두 종류의 성공적인 리턴을 할 수 있다.
        + 수신 버퍼에 데이터가 도달한 경우 : recv()함수의 세 번째 인자인 len보다 크지 않은 범위에서 가능하면 많은 데이터를 응용 프로그램 버퍼에 복사한 후 실제 복사한 바이트 수를 리턴(최소 1, 최대 len)한다.
        + 접속이 정상 종료한 경우 : 상대편 응용 프로그램이 closesocket()함수를 호출해 접속을 종료하면, TCP 프로토콜 수준에서 접속 종료를 위한 패킷 교환 절차가 일어난다. 0을 리턴하는데 이 경우를 정상 종료(normal close of graceful close)라 부른다.

TCP 종료 시 FIN, ACK, FIN, ACK 네 개의 패킷 교환이 일어난다. 그러나 때로는 FIN, FIN/ACK, ACK 세개의 패킷이 교환되기도 한다. 접속 종료 시 먼저 closesocket()함수를 호출한 소켓의 TCP 포트는 곧바로 파괴되지 않고 TIME\_WAIT라는 특별한 상태를 거친 후 사라진다.



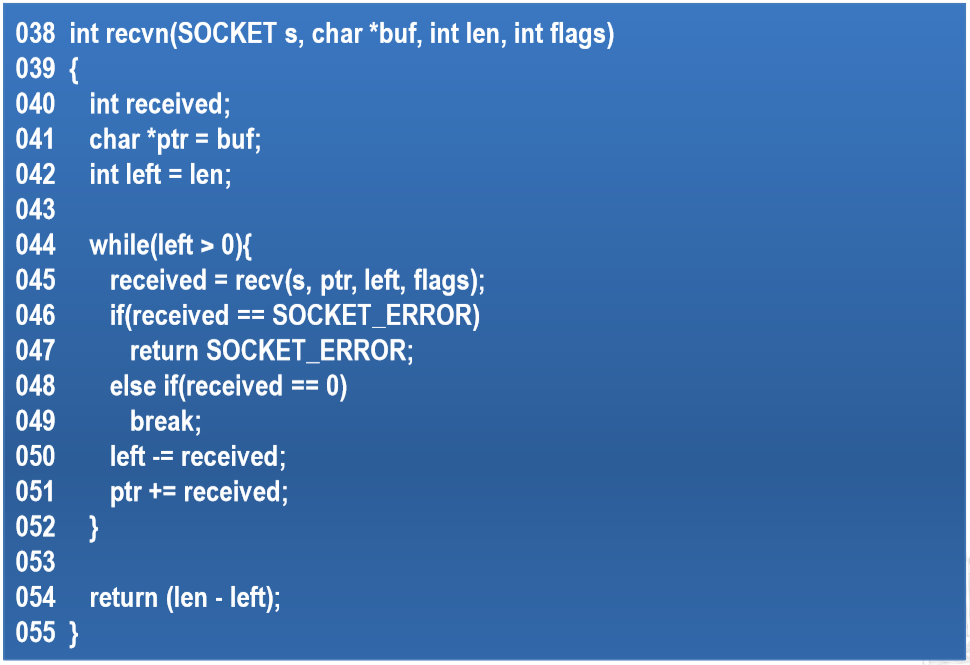
**통신할 대상과 연결된 소켓**

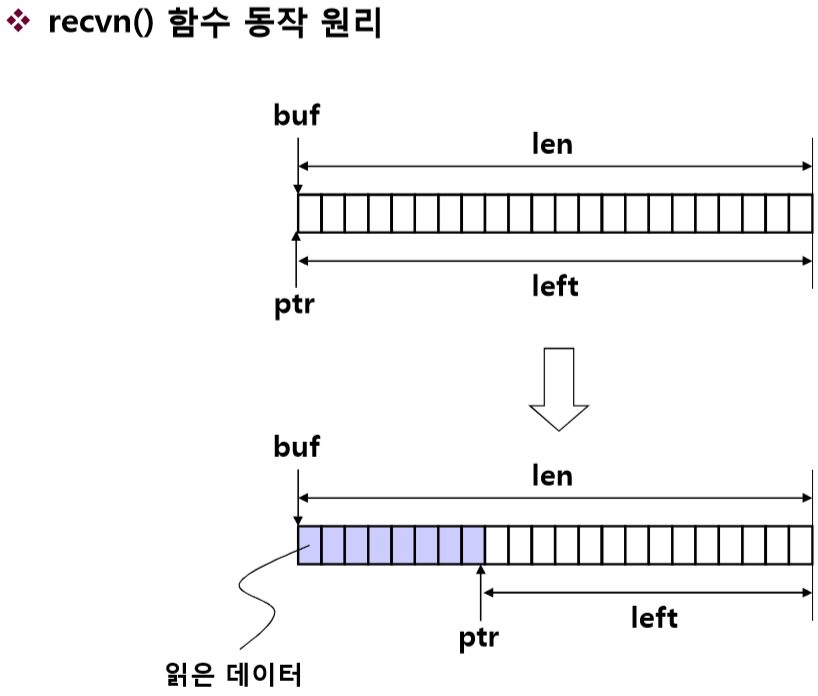
**벋은 데이터를 저장할 응용 프로그램 버퍼 주소**

**운영체제의 수신 버퍼로부터 복사할 최대 데이터 크기**

**recv()함수의 동적을 바꾸는 옵션. 대부분 0 사용**

* + 1. recv()함수 사용 시 특히 주의할 점은 세 번째 인자인 len으로 지정한 크기보다 적은 테이터가 응용 프로그램 버퍼에 복사될 수 있다는 사실이다. 이는 TCP가 데이터 경계를 구분하지 않는다는 특성에 기인한다. 따라서 자신이 받을 데이터의 크기를 미리 알고 있다면 그만큼 받을 때까지 recv()함수를 여러 번 호출해야 한다.
       - 사용자 정의 함수 recvn()을 사용



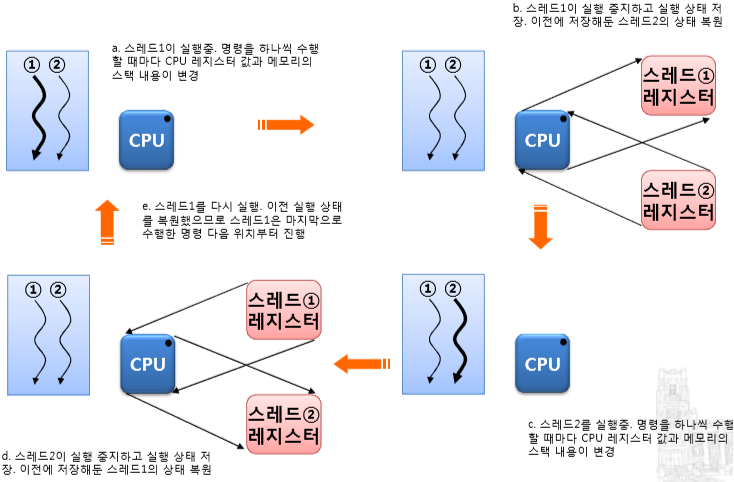


**CH05. 데이터 전송하기**

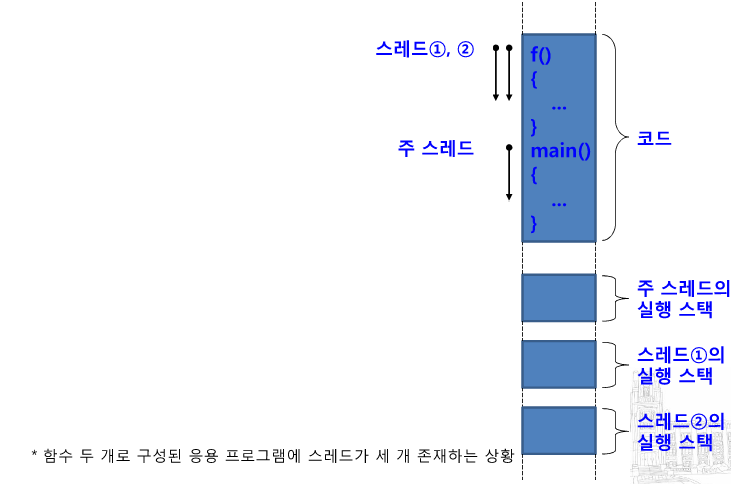
* 응용 프로그램 프로토콜
  1. 응용 프로그램 수준에서 주고받는 데이터의 형식과 의미 그리고 처리 방식을 정의한 프로토콜
* 데이터 전송 시 고려 사항
  1. 경계 구분을 위한 네 가지 방식
     1. 송신자는 항상 고정 길이 데이터를 보냄. 수신자는 항상 고정 길이 데이터를 읽음
     2. 송신자는 가변 길이 데이터를 보내고 끝 부분에 특별한 표시(EOR, End Of Record)를 붙임. 수신자는 EOR이 나올 때까지 데이터를 읽음
     3. 송신자는 보낼 데이터 크기를 고정 길이 데이터로 보내고, 이어서 가변 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 종료함. 수신자는 recv()함수의 리턴 값이 0이 될 때까지 데이터를 읽음
  2. 바이트 정렬 : 서로 다른 바이트 정렬 방식을 사용하는 시스템 사이에서 데이터를 교환할 때 바이트 정렬 방식을 통일하지 않으면 데이터 해석에 문제 발생
  3. 구조체 멤버 맞춤
     1. 구조체(C++의 클래스 포함) 멤버의 메모리 시작 주소를 결정하는 컴파일러의 규칙을 뜻함
     2. #pragma pack 지시자를 사용하거나 프로젝트 속성을 설정하면 기본 구조체 멤버 맞춤 방식 변경 가능

**CH06. 멀티스레드**

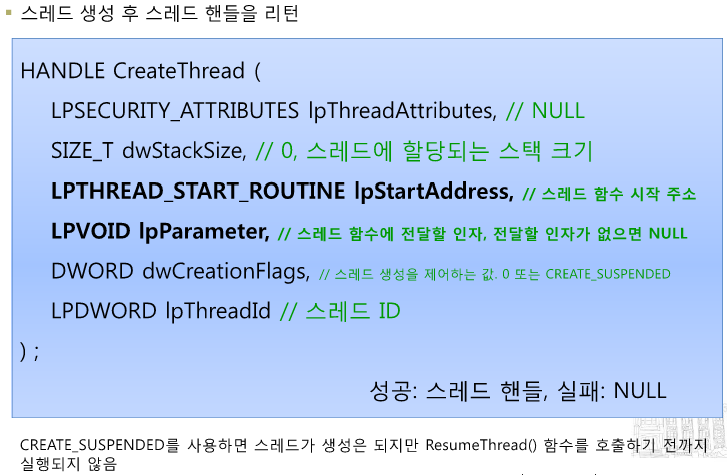
* TCP 서버-클라이언트의 문제점 및 해결책
  1. 1 : 동시에 둘 이상의 클라이언트 서비스 불가
     1. 서버가 각 클라이언트와 통신하는 시간을 짧게 줄임
        + 장점 : 구현이 쉬움, 가장 적은 시스템 자원 사용
        + 단점 : 각 클라이언트의 처리 지연 시간이 길어질 수 있음
     2. 각 클라이언트를 스레드를 이용해 독립적으로 처리
        + 장점 : 소켓 입출력 모델에 비해 구현이 쉬움
        + 단점 : 가장 많은 시스템 자원 사용
     3. 소켓 입출력 모델 사용
        + 장점 : 소수의 스레드를 이용해 다수의 클라이언트를 처리
        + 상대적으로 적은 시스템 자원 사용
        + 단점 : 구현이 어려움
  2. 2 : 교착 상태 발생 가능성
     + - 서버와 클라이언트의 send(), recv() 함수의 호출 순서가 서로 맞아야한다. 데이터를 보내지 않은 상태에서 양쪽에서 동시에 recv()함수를 호출하면 교착상태가 발생할 수 있다.
       - 교착상태 : 영원히 일어나지 않을 이벤트를 두개 이상의 프로세스가 기다리는 상황
     1. 데이터 송수신 부분 잘 설계하기
        + 장점 : 특별한 기법 없이 곧바로 구현 가능
        + 단점 : 모든 경우에 대한 해결책이 될 수 없음
     2. 소켓에 타임아웃 옵션 적용하기
        + 장점 : 구현이 쉬움
        + 단점 : 다른 방법에 비해 성능이 낮음
     3. 넌블로킹 소켓 사용하기
        + 장점 : 근본적으로 교착 상태 해결
        + 단점 : 구현이 복잡, 시스템 자원(CPU시간)낭비
     4. 소켓 입출력 모델 사용
        + 장점 : 넌블로킹 소켓의 단점을 보완 & 교착 상태 해결
        + 단점 : 구현이 어려움
* 용어
  1. 프로세스 : 코드, 데이터, 리소스를 파일에서 읽어 들여 윈도우 운영체제가 할당해 놓은 메모리 영역에 담고 있는 일종의 컨테이너로 정적인 개념
  2. 스레드 : CPU 시간을 할당 받아 프로세스 메모리 영역에 있는 코드를 수행하고 데이터를 사용하는 동적인 개념
  3. 주 스레드 or 메인 스레드
     1. 응용 프로그램 실행 시 최초로 생성되는 스레드
     2. WinMain() 또는 main() 함수에서 실행 시작
  4. 멀티 스레드 응용 프로그램 : 응용 프로그램에서 주 스레드와 별도로 동시에 수행하고자 스레드를 추가로 생성해 이 스레드가 해당 작업을 수행하도록 구현
  5. 컨텍스트 전환(Context switch) : 하드웨어(CPU)와 소프트웨어(OS)의 협력으로 이루어지는 스레드 실행 상태의 저장과 복원 작업
* 멀티 스레드 동작 원리



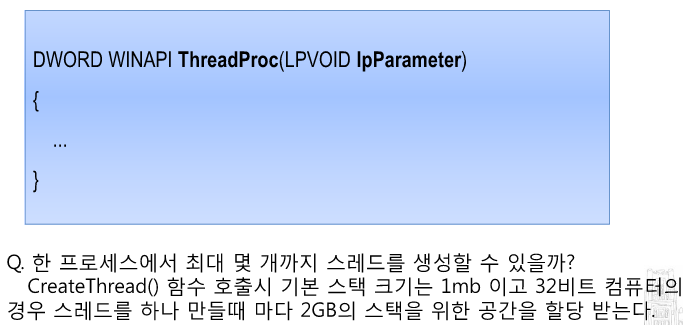
* 스레드 생성과 종료
  1. 스레드 함수의 시작 주소
     1. 운영체제는 f() 함수의 시작 주소를 알아야 한다
     2. f() 함수와 같이 스레드 실행 시작점이 되는 함수를 스레드 함수라 칭함
  2. 스레드 함수 실행 시 사용할 스택의 크기
     1. 모든 함수는 실행 중 인자 전달과 변수 할당을 위해 스택이 필요
     2. 스레드 실행에 필요한 스택 생성은 운영체제가 자동으로 설정. 스택 크기만 정의하면 됨



* 1. CreateThread() 함수



* 1. 스레드 함수 형태



* 1. 스레드 종료 방법
     1. 스레드 함수가 리턴

주로 사용

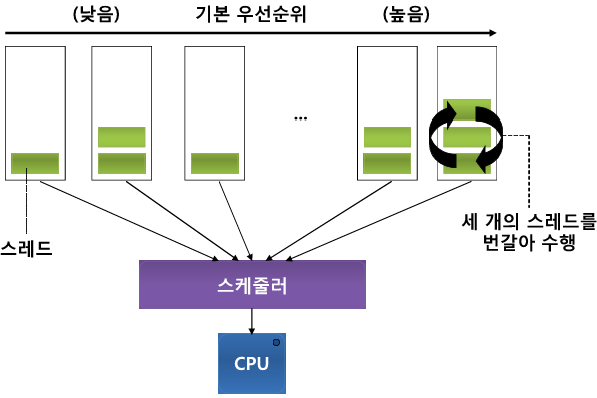
* + 1. 스레드 함수 내에서 ExitThread() 함수를 호출
    2. 다른 스레드가 TerminateThread() 함수를 호출
    3. 주 스레드가 종료하면 모든 스레드가 종료

필요한 경우만 사용

* 1. 스레드 종료 함수

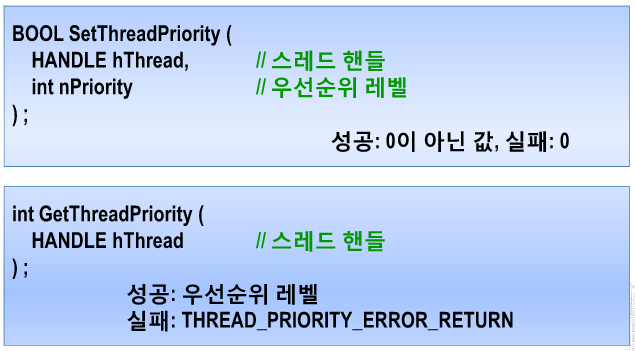


* 스레드 제어 – 우선순위 변경
  1. 스레드는 윈도우 운영체제의 실행 단위이므로, 우선순위를 변경하거나 실행을 중지하고 재 시작하는 등의 제어 기능을 윈도우 API수준에서 지원
  2. 용어
     1. 스레드 스케줄링 or CPU 스케줄링
        + 윈도우 운영체제가 각 스레드에 CPU 시간을 적절히 분배하기 위한 정책
     2. 우선순위 클래스
        + 프로세스 속성으로, 같은 프로세스가 생성한 스레드는 우선순위 클래스가 모두 같음
     3. 우선순위 레벨
        + 스레드 속성으로, 같은 프로세스에 속한 스레드 간 상대적인 우선순위를 결정할 때 사용
        + 프로그래머들이 건들일 수 있는 것
     4. 기본 우선순위
        + 우선순위 클래스와 우선순위 레벨을 결합한 값으로, 스레드 스케줄링에 사용
  3. 윈도우 운영체제에서 제공하는 우선순위 클래스
     1. REALTIME\_PRIORITY\_CLASS (실시간)
     2. HIGH\_PRIORITY\_CLASS (높음)
     3. ABOVE\_NORMAL\_PRIORITY\_CLASS (높은 우선 순위)
     4. NORMAL\_PRIORITY\_CLASS (보통)
     5. BELOW\_NORMAL\_PRIORITY\_CLASS (낮은 우선 순위)
     6. IDLE\_PRIORITY\_CLASS (낮음)
  4. 윈도우 운영 체제에서 제공하는 우선순위 레벨
     + - 우선순위 레벨은 스레드 속성으로, 같은 프로세스에 속한 스레드 간 상대적인 우선순위를 정할 때 사용
     1. THREAD\_PRIORITY\_TIME\_CRITICAL
     2. THREAD\_PRIORITY\_HIGHEST
     3. THREAD\_PRIORITY\_ABOVE\_NORMAL
     4. THREAD\_PRIORITY\_NORMAL
     5. THREAD\_PRIORITY\_BELOW\_NORMAL
     6. THREAD\_PRIORITY\_LOWEST
     7. THREAD\_PRIORITY\_IDLE
  5. 윈도우의 스레드 스케줄링 방식
     + - 우선순위가 가장 높은 스레드에 CPU 시간을 할당하되, 우선순위가 같은 스레드가 여러 개 있을 때는 CPU 시간을 번갈아 가며 할당



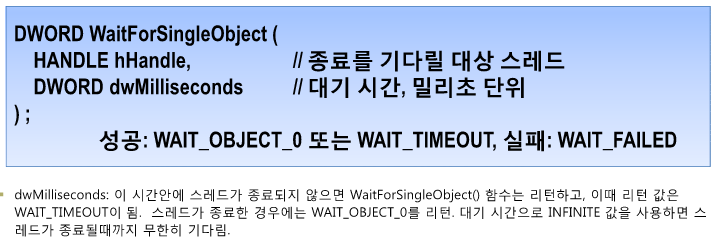
* 1. 우선순위 레벨 조작(변경) 함수

**우선순위 레벨을 변경**

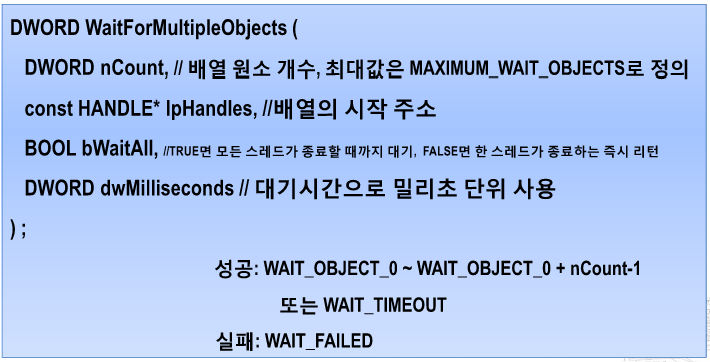


**우선순위 레벨을 얻을 때**

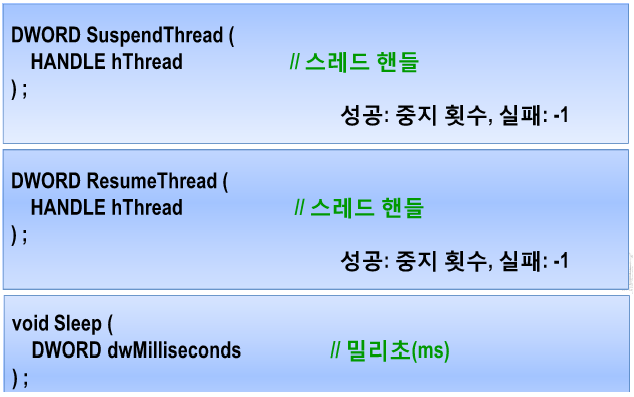
* 스레드 제어 – 스레드 종료 기다리기
  1. WaitForSingleObject() 함수
     + - 특정 스레드가 종료할 때까지 기다리기



* 1. WaitForMultipleObjects() 함수
     + - 여러 스레드가 종료하기를 기다리려면 WaitForSingleObject()함수를 스레드 개수만큼 호출해야 함. 하지만 이 함수를 사용하면 호출 한번으로 끝낼 수 있음
       - 둘 이상의 스레드가 종료할 때까지 기다리기



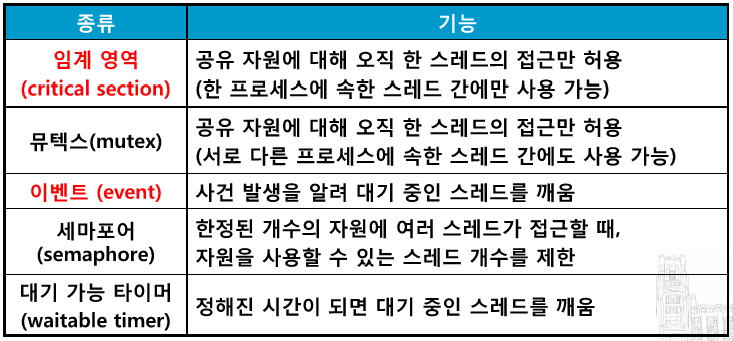
* 스레드 제어 – 실행 중지와 재실행
  1. 실행 중지 함수(SuspendTread())
  2. 재실행 함수(ResumeThread())
     + - 윈도우 운영체제는 스레드의 중지 횟수(suspend count)를 관리
  3. SuspendThread()함수를 호출한 경우에는 반드시 ResumeThread() 함수를 사용해야 스레드가 재 시작
  4. Sleep() 함수를 호출하면 dwMilisecond로 지정한 시간이 지나면 자동으로 재시작



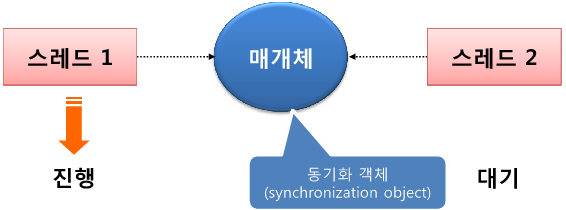
* 소켓과 연관된 주소 정보 얻기
  1. 스레드 함수에 소켓만 전달한 경우에는 별도의 주소 정보가 없으므로, 소켓을 통해 주소 정보를 얻는 기능 구현이 필요
  2. getpeername() : 원격 IP/Port
  3. getsocketname() : 지역 IP/Port



* 스레드 동기화
  1. 스레드 동기화 필요성
     1. 멀티스레드를 이용하는 프로그램에서 스레드 두 개 이상이 공유 데이터에 접근하면 다양한 문제가 발생
     2. 스레드 동기화 : 멀티스레드 환경에서 발생하는 문제를 해결하기 위한 일련의 작업
  2. 스레드 동기화 기법



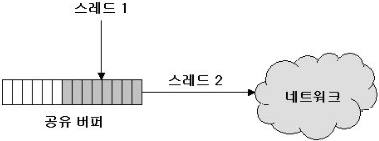
* 1. 스레드 동기화가 필요한 상황
     1. 둘 이상의 스레드가 공유 자원에 접근
     2. 한 스레드가 작업을 완료한 후, 기다리고 있는 다른 스레드에 알림
  2. 스레드 동기화 원리
     1. 스레드를 동기화 하려면 스레드가 상호 작용해야 하므로 중간에 매개체 역할을 하는 부분이 필요
     2. 두 스레드가 동시에 진행하면 안되는 상황에서 두 스레드는 매개체를 통해 진행 여부를 판단하고 이에 근거해 자신의 실행을 계속할지 결정



* 1. (스레드) 동기화 객체의 특징
     1. Create\*() 함수를 호출하면 커널 메모리 영역에 동기화 객체가 생성되고, 이에 접근할 수 있는 핸들이 리턴됨
     2. 평소에는 비신호 상태로 있다가 특정 조건이 만족되면 신호 상태가 됨. 비 신호 상태에서 신호 상태로 변화 여부는 Wait\*()함수를 사용해 감지
     3. 사용이 끝나면 CloseHandle() 함수를 호출
     4. Wait\*()함수는 스레드 동기화를 위한 필수 함수
     5. 동기화 객체 활용시 비신호->신호, 신호->비신호 상태 변화 조건 이해 필요하며, 상황에 맞게 Wait\*()함수를 사용해야 함



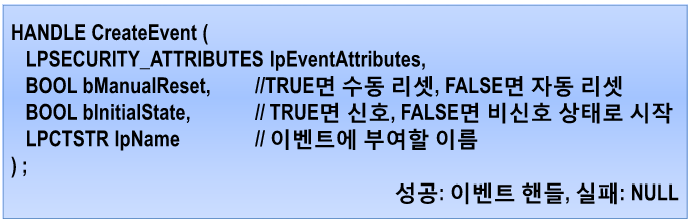
* 임계 영역
  1. 임계 영역
     1. 둘 이상의 스레드가 공유 자원에 접근할 때, 오직 한 스레드만 접근을 허용해야 하는 경우에 사용
     2. 대표적인 스레드 동기화 기법이지만, 생성과 사용법이 달라서 동기화 객체로 분류하지 않음
  2. 특징
     1. 프로세스의 유저 메모리 영역에 존재하는 단순한 구조체이므로 한 프로세스에 속한 스레드 간 동기화에만 사용
     2. 일반 동기화 객체보다 빠르고 효율적
     3. 임계 영역만으로는 어느 스레드가 먼저 리소스를 사용할지 결정할 수 없음
  3. 사용법
     1. CRITICAL\_SECTION 구조체 변수를 전역 변수로 선언
     2. 임계 영역을 사용하기 전에 InitializeCriticalSection() 함수 호출(초기화)
     3. 공유 자원에 접근하기 전에 EnterCriticalSection() 함수를 호출. 공유자원을 사용하고 있는 스레드가 없다면 EnterCriticalSection() 함수는 곧바로 리턴. 공유자원을 사용하고 있는 스레드가 있다면 EnterCriticalSection() 함수는 리턴하지 못하고 스레드는 대기상태가 됨.
     4. 공유 자원 사용을 마치면 LeaveCriticalSection() 함수를 호출. 이때 EnterCriticalSection() 함수에서 대기 중인 스레드가 있다면 하나만 선택되어 깨어남.
     5. 임계 영역을 사용하는 모든 스레드가 종료하면 DeleteCriticalSection() 함수를 호출하여 삭제



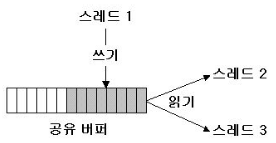
* 이벤트
  1. 이벤트
     1. 사건 발생을 다른 스레드에 알릴 때 사용
     2. 한 스레드가 작업을 완료한 후 기다리고 있는 다른 스레드에 알릴 때 사용
  2. 이벤트를 사용하는 전형적인 절차
     1. 이벤트를 비신호 상태로 생성
     2. 한 스레드가 작업을 진행하고, 나머지 스레드는 이벤트에 대해 Wait\*( ) 함수를 호출해 이벤트가 신호 상태가 될 때까지 대기 (sleep)
     3. 스레드가 작업을 완료하면 이벤트를 신호 상태로 바꿈
     4. 기다리고 있던 스레드 중 하나 혹은 전부가 깨어남 (wakeup)
  3. 이벤트 상태 변경



* 1. 이벤트의 종류
     1. 자동 리셋 이벤트
        + 이벤트를 신호 상태로 바꾸면, 기다리는 스레드 중 하나만 깨운 후 자동으로 비신호 상태가 됨
     2. 수도 리셋 이벤트
        + 이벤트를 신호 상태로 바꾸면, 기다리는 스레드를 모두 깨운 후 계속 신호 상태를 유지함
        + 비신호로 변경이 필요할 시 ResetEvent()함수를 호출
  2. 이벤트 생성

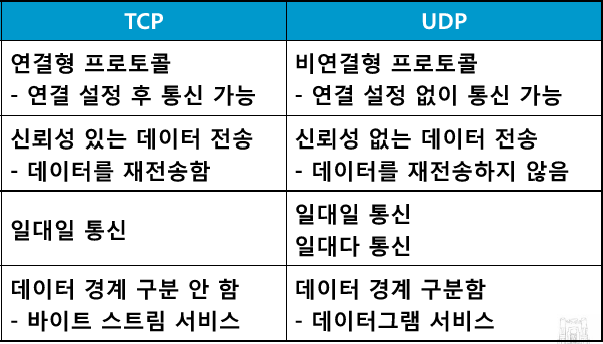


* 1. 전제 조건
     1. 스레드 1이 쓰기를 완료한 후 스레드 2나 스레드3이 읽을 수 있다. 이때 스레드 2와 스레드 3중 한 개만 버퍼 데이터를 읽을 수 있으며, 일단 한 스레드가 읽기 시작하면 다른 스레드는 읽을 수 없다.
     2. 스레드 2가 스레드 3이 읽기를 완료하면 스레드 1이 다시 쓰기를 할 수 있다.

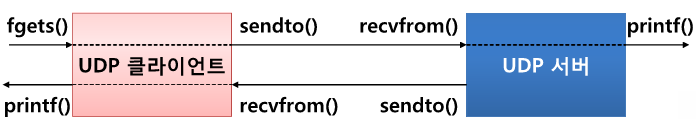


**CH07. UDP 서버-클라이언트**

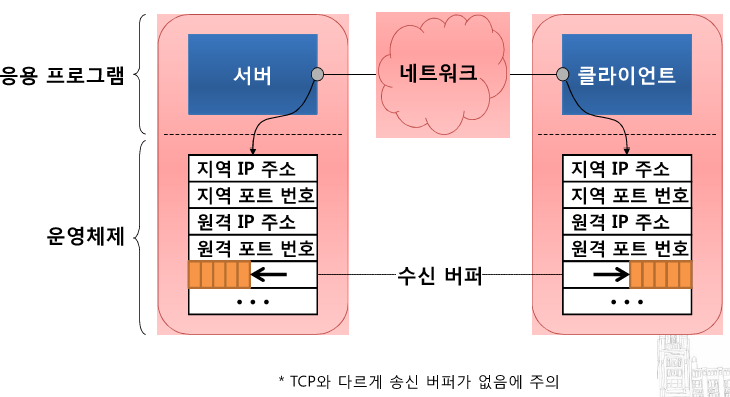
* TCP 서버-클라이언트의 문제점 및 해결책
  1. TCP와 UDP의 공통점
     1. 전송 계층 프로토콜
     2. 포트 번호를 이용해 주소를 지정
        + 포트 번호를 이용하여 종단(응용 프로그램)간 전송
        + 두 응용 프로그램이 TCP나 UDP를 이용해 통신하려면 반드시 포트 번호를 결정해야 함
     3. 데이터 오류를 체크
        + 데이터 위변조 확인
        + IP와 달리 TCP와 UDP는 헤더는 물론이고 데이터에 대한 오류도 체크함
  2. TCP와 UDP의 차이점



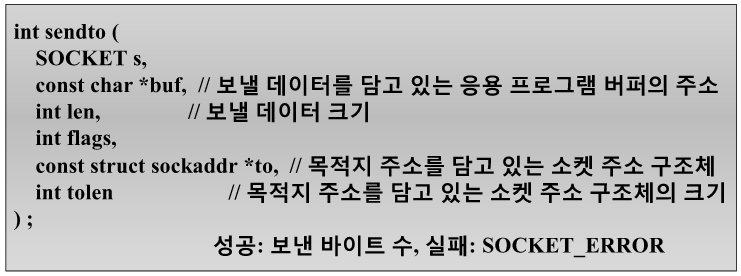
* 1. UDP의 특징
     1. 연결 설정을 하지 않으므로 connect( ) 함수 불필요
        + UDP는 연결 자체가 없음
     2. 프로토콜 수준에서 신뢰성 있는 데이터 전송을 보장하지 않으므로, 필요하다면 응용 프로그램 수준에서 신뢰성 있는 데이터 전송 기능을 구현해야 함
     3. 간단한 소켓 함수 호출 절차만 따르면 다자 간 통신을 쉽게 구현할 수 있음
        + 1:1 혹은 1:다 연결 TCP와 달리 응용 프로그램이 데이터 경계 구분을 위한 작업을 별도로 할 필요가 없음
     4. 데이터그램(메시지) 단위 전송이며 하나의 데이터그램은 65535바이트 크기로 제한됨
        + 65535 바이트 이상의 크기는 잘라서 보내야 함
     5. Not ordered
     6. 높은 성능
        + 신뢰성을 희생해서 성능 확보
  2. UDP를 사용하는 경우
     1. 성능을 높일 수 있다
     2. 실시간 멀티미디어 서비스 프로그램 개발이 용이
        + 연속성이 신뢰성보다 중요한 서비스
     3. 개발이 쉽다
        + 흐름이 관리하기 위한 노력이 필요 없다
     4. Agent & Manager 모델에 적합
        + Agent가 Manager로 데이터를 요청
        + 다수의 Manager를 효과적으로 관리
        + 오류에 민감하지 않음 (다시 요청하면 됨)
     5. 주로 파일, 스트리밍 콘텐츠 전송 등에 사용
     6. TCP는 전화로 비유되고 UDP는 우편으로 비유할 수 있음
* UDP 서버-클라이언트
  1. UDP서버
     1. 클라이언트가 보낸 데이터를 받고(recvfrom), 이를 문자열로 간주해 무조건 화면에 출력
     2. 받은 데이터를 변경 없이 다시 클라이언트에 보냄(sendto)
  2. UDP클라이언트
     1. 사용자가 키보드로 입력한(fgets) 문자열을 서버에 보냄(sendto)
     2. 서버가 데이터를 그대로 돌려보내면, 클라이언트는 이를 받아(recvfrom)화면에 출력



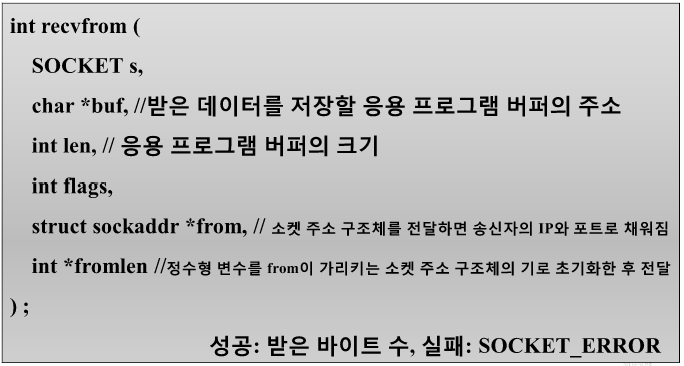
* UDP 서버-클라이언트 분석
  1. 소켓 통신을 위해 결정해야 할 요소
     1. 프로토콜
        + 통신 규약. 소켓을 생성할 때 결정
     2. 지역 IP주소와 지역 포트 번호
        + 서버 또는 클라이언트 자신의 주소
     3. 원격 IP주소와 원격 포트 번호
        + 서버 또는 클라이언트가 통신하는 상대의 주소
  2. 소켓 데이터 구조체



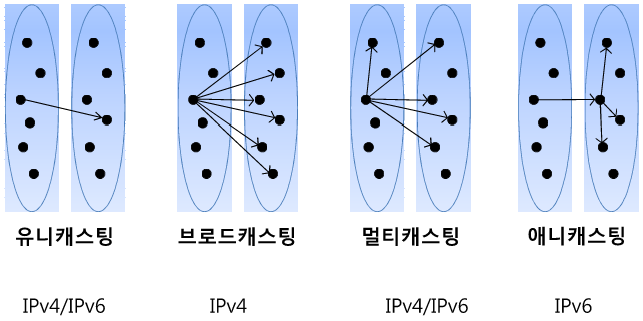
* 데이터 전송 함수
  1. sendto() 함수
     1. 응용 프로그램 데이터를 운영체제의 송신 버퍼에 복사함으로써 데이터를 전송
        + 소켓의 지역 IP주소와 지역 포트 번호가 아직 결정되지 않은 상태라면 운영체제가 자동으로 결정
     2. connect함수로, 서버를 명시하기 위해서 사용
     3. 실제 connect과정을 수행하는 건 아님
     4. 데이터를 전송할 때, 주소 복사 과정을 생략할 수 있음



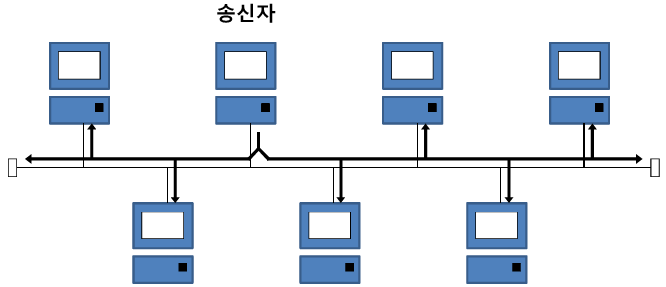
* 1. recvfrom() 함수
     1. 운영체제의 수신 버퍼에 도착한 데이터를 응용 프로그램 버퍼에 복사
        + UDP패킷 데이터를 한 번에 하나만 읽을 수 있음



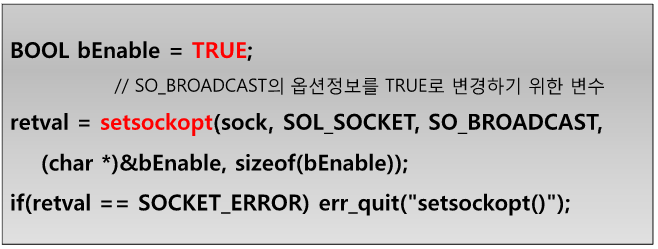
* 브로드캐스팅
  1. 통신에 참여하는 개체 간 상호 작용



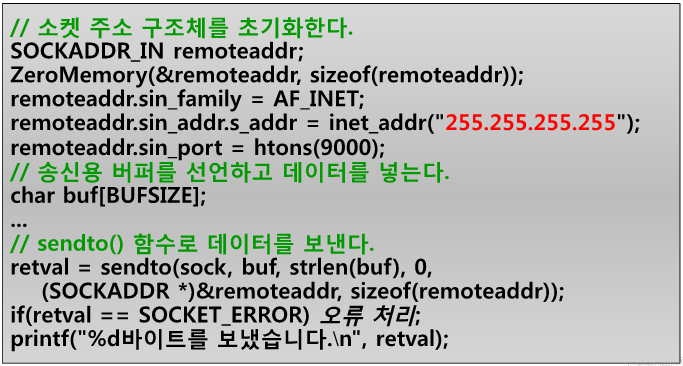
* 1. 개념
     1. 송신자가 보낸 데이터 하나를 다수의 수신자가 받는 방식
        + 데이터 복사본을 여러 개 만들어 보내는 것이 아니므로 송신자 관점에서 보면 상당히 효율적인 기술임



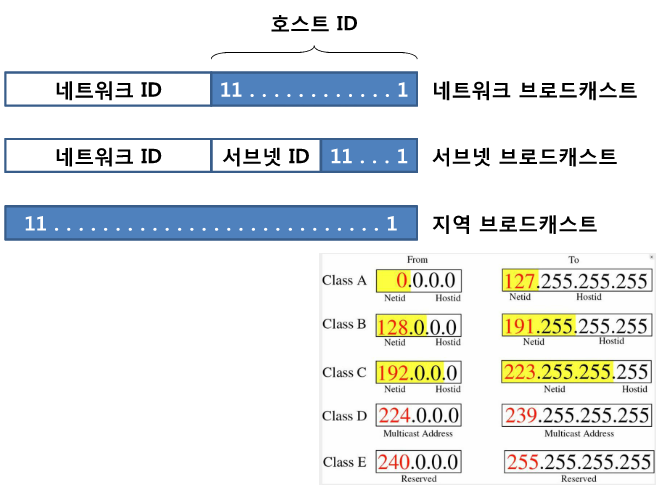
* 1. 브로드캐스트 데이터를 보내기 위한 절차
     1. 브로드캐스팅을 활성화

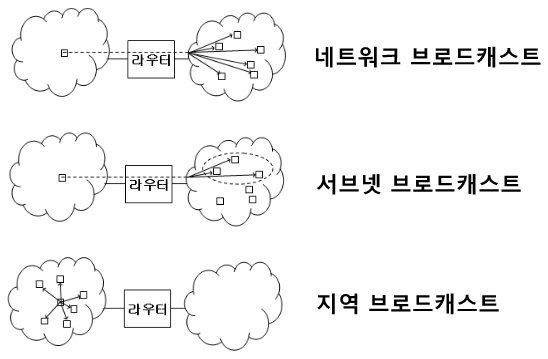


* + 1. 브로드캐스트 주소로 데이터를 보냄



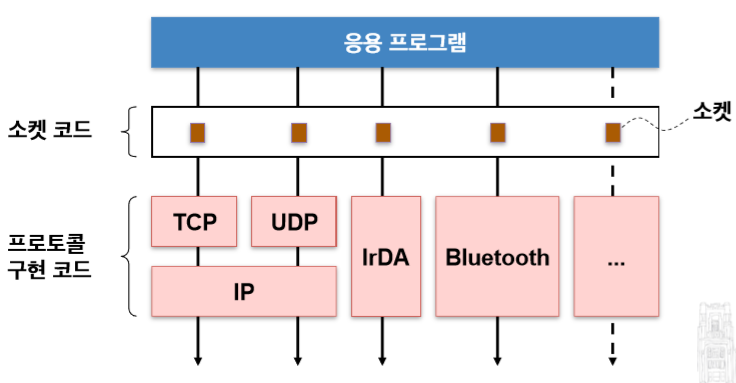
* 1. 브로드캐스트 주소의 종류



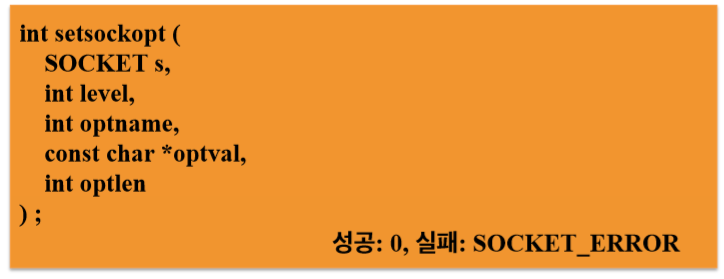


**CH07. UDP 서버-클라이언트**

* 개요
  1. 소켓 프로그래밍 모델
     1. 응용 프로그램은 소켓 코드가 제공하는 인터페이스인 소켓 함수를 사용하므로, 통신 프로토콜을 변경하더라도 기존 코드 수정을 최소화할 수 있다는 장점이 있다.



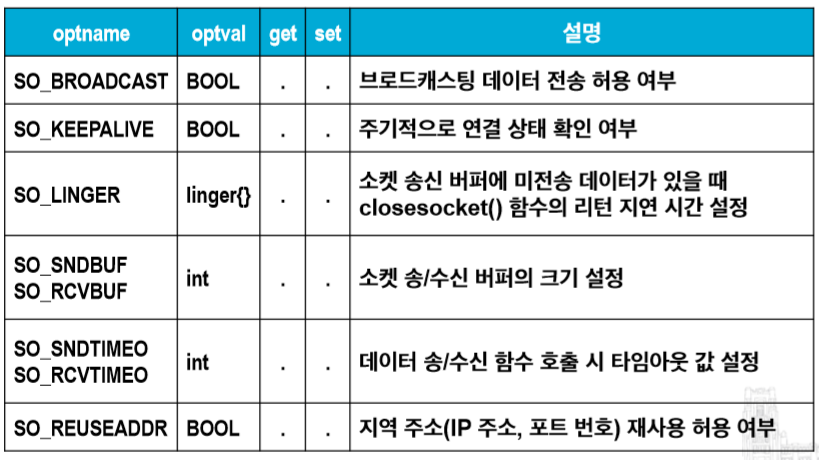
* 1. 소켓 옵션
     1. 소켓 함수의 기본 동작을 변경
        + 소켓 코드와 프로토콜 구현 코드를 세부적으로 제어
  2. 소켓 옵션의 종류
     1. 소켓 코드가 처리하는 옵션
        + 옵션을 설정하면 소켓 코드에서 해석하고 처리
     2. 프로토콜 구현 코드가 처리하는 옵션
        + 옵션을 설정하면 프로토콜 구현 코드에서 해석하고 처리
  3. 소켓 옵션 설정(=변경)하기



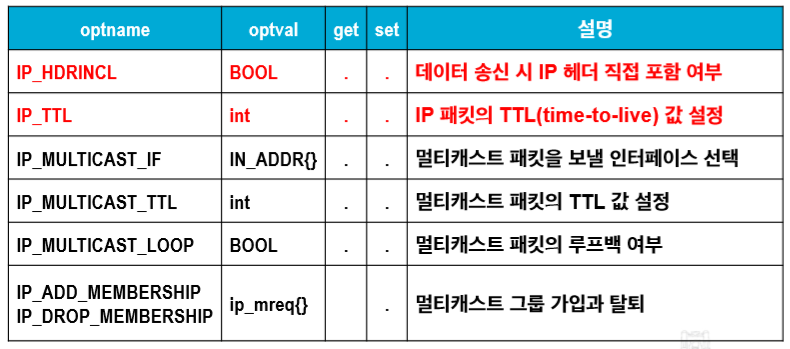
* 1. 현재 설정된 소켓 옵션 값 얻기



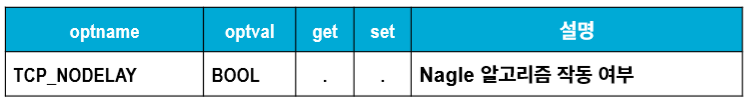
* 소켓 옵션 종류
  1. Level = SOL\_SOCKET



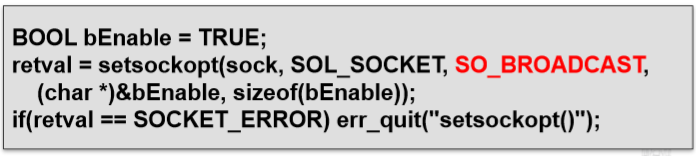
* 1. Level = IPPROTO\_IP



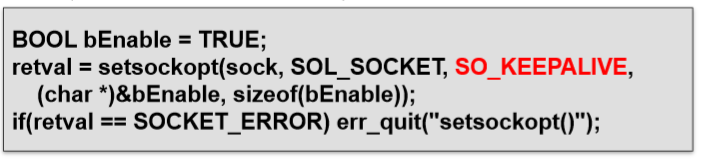
* 1. Level = IPPROTO\_TCP



* SO\_BROADCAST 옵션
  1. 용도
     1. 브로드캐스트 데이터 전송 기능 활성화
     2. TCP 소켓에는 사용할 수 없고 UDP 소켓에만 사용 가능
  2. 사용 예
     1. 브로드캐스트가 기본값이 아니라 옵션인 이유?
        + 네트워크 효율성
        + 굳이 수신이 불필요한 컴퓨터까지 보낼 필요 없음



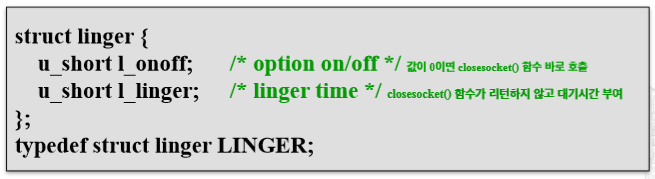
* SO\_KEEPALIVE 옵션
  1. 용도
     1. TCP 프로토콜 수준에서 연결 여부를 확인하려고 상대 TCP에 주기적(약 2시간 간격)으로 TCP 패킷을 보냄
  2. 사용 예(상대 TCP의 반응에 따른 동작)



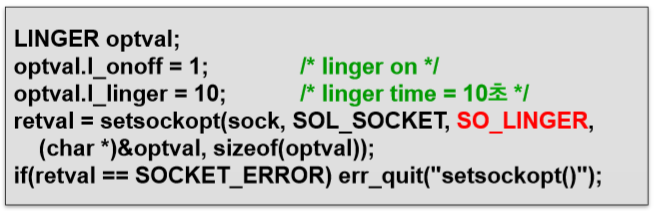
* + 1. 상대 TCP가 정해진 시간 안에 응답하는 경우
       - TCP 연결에 문제가 없는 상태
       - 응용프로그램 정상 동작
    2. 상대 TCP가 정해진 시간 안에 응답하지 않은 경우
       - TCP연결이 일시적으로 문제가 있는 상태
       - 몇 번 더 TCP 패킷을 보낸 후 응답이 없으면 자동으로 소켓을 닫음
    3. 상대 TCP가 RST(reset)패킷으로 응답하는 경우
       - 예상치 못한 패킷이 도달하면 비정상적인 상황으로 간주해 강제 종료
    4. SO\_KEEPALIVE옵션을 설정해 두면, TCP 프로토콜 수준에서 주기적으로 끊어진 연결을 감지해 해당 소켓을 닫으므로 불필요한 시스템 자원 소모를 방지
* SO\_LINGER 옵션
  1. 용도
     1. 소켓 송신 버퍼에 미 전송 데이터가 있을 때, closesocket()함수의 동작 변경



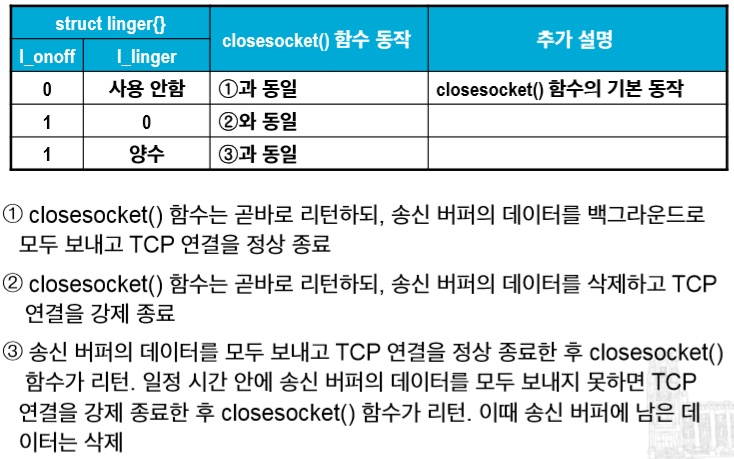
* + 1. TCP 소켓을 사용할 경우, closesocket() 함수는 두가지 기능을 한다.
       - 소켓을 닫고 할당된 운영체제 자원을 반환. Closesocket() 함수 리턴 후에 해당 소켓을 사용할 수 없음
       - TCP 프로토콜 수준에서 연결 종료 절차를 시작
  1. 옵션 값
     1. Closesocket()함수의 두가지 기능을 세부적으로 제어하는데 사용



* 1. 사용 예



* + - * 송신 버퍼의 데이터를 모두 보내고 TCP 연결을 정상 종료한 후 closesocket()함수가 리턴하며, 10초 이내에 송신 버퍼의 데이터를 모두 보내지 못하면 TCP 연결을 강제 종료한 후 closesocekt()함수 리턴
      * 이 경우 송신 버퍼에 남은 데이터는 삭제
  1. 옵션 값에 따른 closesocket()함수 동작



* SO\_SNDBUF, SO\_RCVBUF 옵션
  + - * 데드락을 방지해줌
* SO\_REUSEADDR 옵션
  + - * TCP 서버에서 무조건으로 사용한다고 생각하면 편함
      * 유닉스 서버에서는 정말 필요한 옵션

**예상 문제**

Q. 체크섬으로 오류 판단하는 프로토콜

A. **TCP, UDP**

* 체크섬
  1. 중복 검사의 한 형태
  2. 송신된 자료의 무결성을 보호하는 단순한 방법

Q. 네트워크 상태와 별명 보는 명령어

A. **nslookup**

Q. 클라이언트에서 bind()역할 하는 함수

A. **connect()**

Q. 신호, 비신호 체크하는 함수

A. **WaitForSingleObject(), WaitForMultipleObjects()**

Q. 호스트 대기시간을 알려주는 명령어

A. **ping**

Q. 윈속 종료를 알려주고 리소스를 반환해주는 함수

A. **WSACleanup()**

Q. 함수의 인자로 쓰이는 구조체

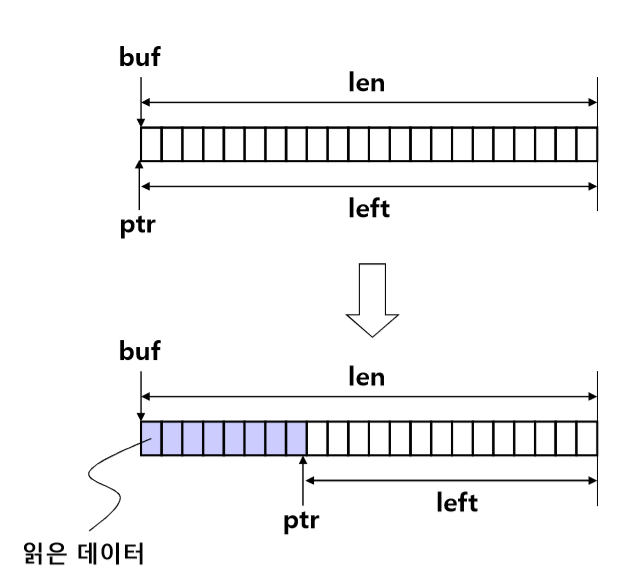
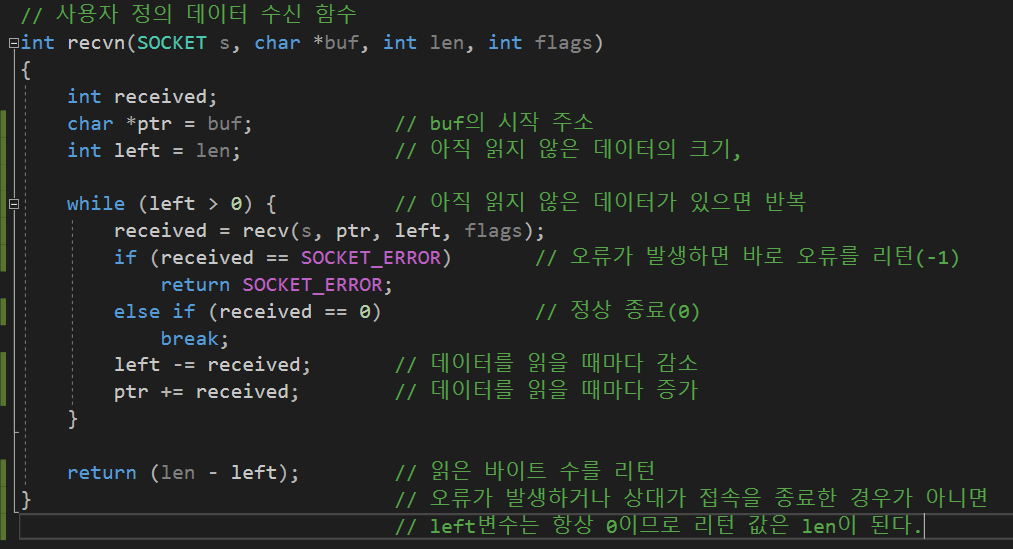
A. **SOCKADDR**

Q. 스레드를 일시정지하는 함수

A. **SuspendThread()**

Q. recvn()을 쓰는 이유? 동작원리?

A. 예를 들면 네트워크의 상황에 따라 10byte가 묶음으로 보내지는 경우도 있지만 5, 3, 2 byte처럼 쪼개져서 보내질 수도 있어서 recv()함수를 반복해서 호출해야 한다. 이때 내가 받아야하는 크기를 알아야 하기 때문에 recvn()함수를 사용한다.



Q. 임계영역이 스레드 동기화에 적합한지, 그 이유

A. O, 일반 동기화 객체보다 빠르고 효율적이다.

임계영역은 한 순간에 하나의 스레드만 접근이 요구되는 공유 리소스에 접근하는 코드 블록을 의미한다. 동시에 공유 자원 접근 차단이 가능하지만 순서 제어가 안되는 단점이 있다.

Q. 임계영역과 이벤트 방식을 설명하고 각 장단점

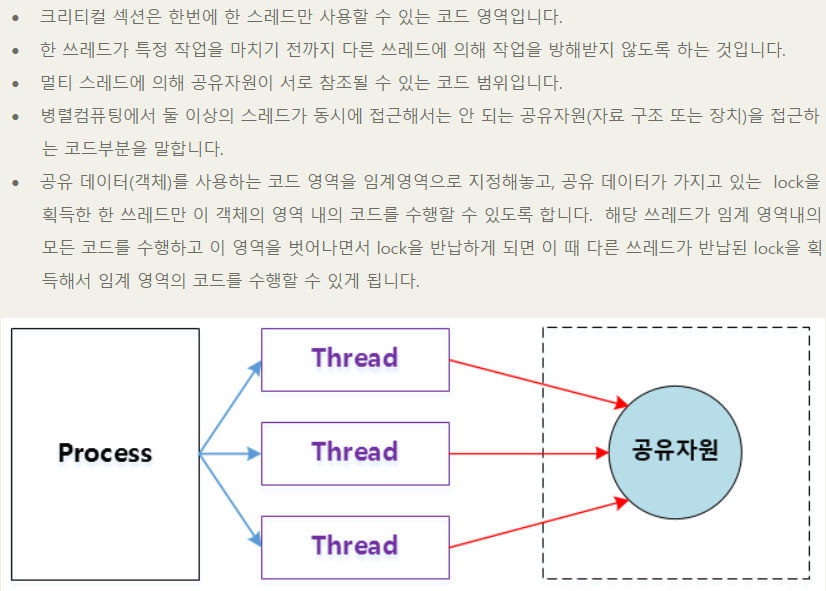
A.

**임계 영역** : 둘 이상의 스레드가 공유 자원에 접근할 때, 오직 한 스레드만 접근을 허용해야 하는 경우에 사용

1. CRITICAL\_SECTION 구조체 변수를 전역 변수로 선언
2. 임계 영역을 사용하기 전에 InitializeCriticalSection()함수를 호출해 초기화
3. 공유 자원에 접근하기 전에 EnterCriticalSection()함수를 호출. 공유 자원을 사용하고 있는 스레드가 없다면 이 함수는 곧바로 리턴한다. 하지만 공유자원을 사용하고 있는 스레드가 있다면 이 함수는 리턴하지 못하고 스레드는 대기 상태가 됨
4. 공유 자원 사용을 마치면 LeaveCriticalSection()함수를 호출. 이 때 이 함수에서 대기 중인 스레드가 있다면 하나만 선택되어 깨어남.
5. 임계 영역을 사용하는 모든 스레드가 종료하면 DeleteCriticalSection()함수를 호출해 삭제

장점 : 일반 동기화 객체보다 빠르고 효율적임

단점 : 순서 제어가 안됨

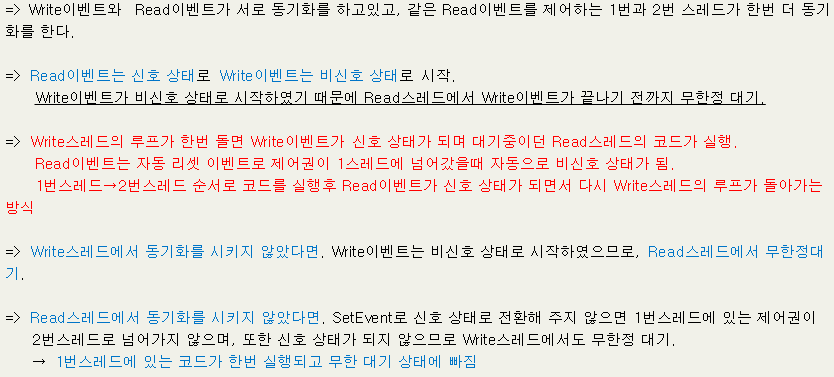


**이벤트** : 사건 발생을 다른 스레드에 알리는 동기화 기법

1. 이벤트를 비신호 상태로 생성
2. 한 스레드가 작업을 진행하고, 나머지 스레드는 이벤트에 대해 Wait\*()함수를 호출해 이벤트가 신호 상태가 될 때까지 대기(sleep)
3. 스레드가 작업을 완료하면 이벤트를 신호 상태로 바꾼다
4. 기다리고 있던 스레드 중 하나 혹은 전부가 깨어난다.(wakeup)

장점 : 순서 제어가 가능

단점 : 스레드 동기화가 제대로 되지 않았다면 무한 대기 상태에 빠짐



Q. 블로킹소켓 + send()

A. 블로킹 소켓을 대상으로 send()함수를 호출하면 송신 버퍼의 여유 공간이 send()함수의 세번째 인자인 len보다 작을 경우 해당 프로세스는 대기 상태가 된다. 송신 버퍼에 충분한 공간이 생기면 프로세스는 깨어나고 len크기만큼 데이터 복사가 일어난 후 send()함수가 리턴(값 = len)이 된다.

Q. 넌블로킹소켓 + send()

A. 넌블로킹소켓을 대상으로 send()함수를 호출하면 송신버퍼의 여유공간만큼 데이터를 복사한 후 실제 복사한 바이트 수를 리턴(최소 1 최대 len)한다.

Q. UDP가 TCP에 비해 비신뢰성 프로토콜인 이유

A. TCP는 데이터가 제대로 전송이 될 때까지 전송하지만 UDP는 깨진 데이터를 재전송하지 않는다. 즉 프로토콜 수준에서 신뢰성 있는 데이터 전송을 보장하지 않으므로, 응용프로그램 수준에서 신뢰성 있는 데이터 전송 기능을 구현해야 한다.