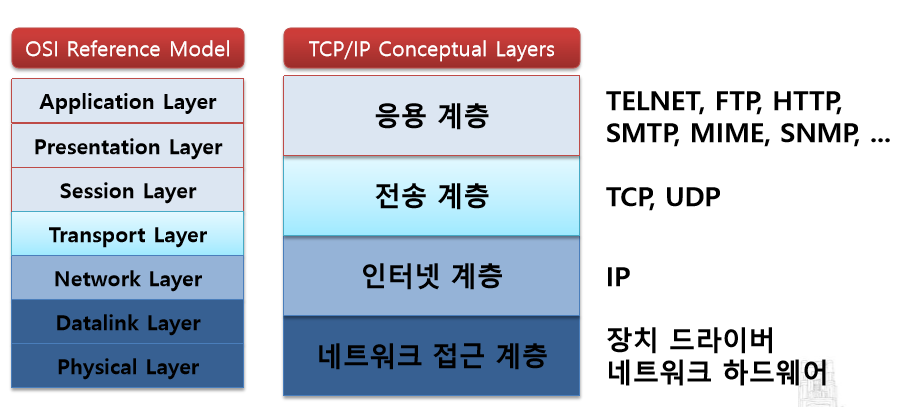
1장

호스트 – 최종 사용자의 응용프로그램을 수행하는 주체

라우터 – 호스트에서 생성된 데이터를 여러 네트워크를 거쳐 전송함으로써 서로 다른 네트워크에 속한 호스트 간에 데이터를 교환할 수 있게 하는 장비

통신 프로토콜 – 호스트와 라우터, 라우터와 라우터, 호스트와 호스트가 통신하기 위한 정해진 방법



네트워크 접근계층 (데이터링크 / 물리)

- 물리적 네트워크를 통한 데이터 송수신, 데이터를 전송하는 케이블에 프레임 송수신

- 네트워크 하드웨어, 장치 드라이버로 구성

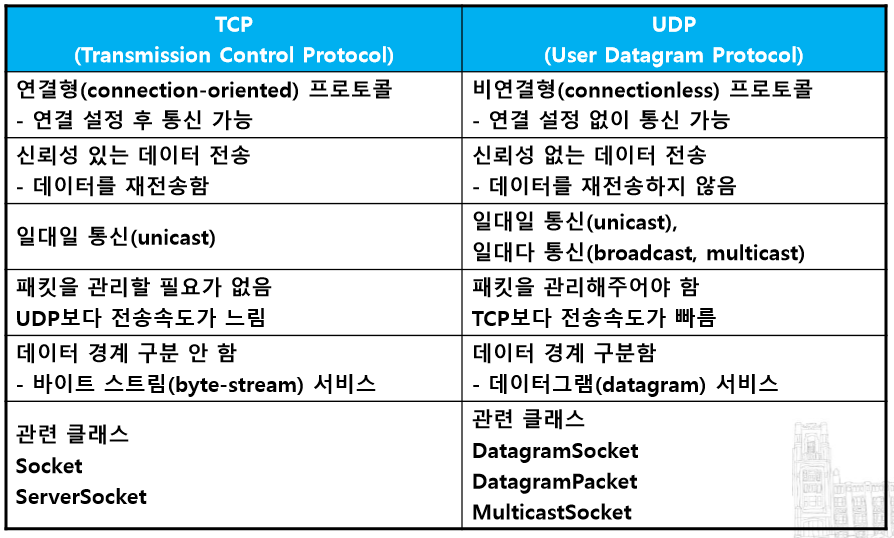
인터넷계층

-네트워크 접근 계층의 도움을 받아 데이터를 목적지까지 전달, ip주소를 관리하고 라우팅하는 역할

전송계층

-최종통신목적지를 지정하고 오류없이 데이터를 전송, 포트번호로 주소를 지정

-신뢰성을위해 흐름제어, 오류제어기능수행, TCP, UDP사용



응용계층

-전송계층을 기반으로 한 다수의 프로토콜과 이 프로토콜을 사용하는 응용프로그램, 어플리케이션이 네트워크에 접근가능하도록 해주는 역할 RTP, HTTP, SMTP

패킷 – 각 프로토콜에서 정의한 제어정보(ip, 포트, 오류체크코드) + 데이터 앞쪽이 헤더 뒤쪽은 트레일러

패킷전송형태 (인터넷)

- 응용계층, 전송계층 -> 하부계층이 제공하는 가상적인 연결을 사용해 동작

- 인터넷계층 -> ip주소와 라우팅 기능을 이용해 패킷전송경로 결정

- 네트워크 접근 계층 -> 물리주소를 사용해 실제 패킷 전송

IP주소

인터넷에 있는 호스트와 라우터의 식별자

IPv4는 32비트 IPv6는 128비트 IPv4는 8비트단위로 .(dot)으로 구분해서 10진수 4개로 표기 147.46.114.70

IPv6는 16비트단위로 : 으로 구분 16진수 8개로 표기 2001:0230:abcd:ffab:eb00:ffff:1111

호트번호

-인터넷 통신의 종착점을 나타내는 식별자, tcp와 udp는 16비트 정수를 사용 범위는 0~65535

도메인 이름 - ip주소에 대한 별명 실제 통신때는 ip주소롤 변환해야한다.

데이터타입

-파일 디스크립터 혹은 핸들과 유사한 개념

-통신 종단점은 응용프로그램은 자신의 소켓이 상대편 소켓과 연결된것으로 생각하고 데이터를 주고받음

**소켓의개념**

데이터타입 – 파일디스크럽터 혹은 핸들과 유사한 개념, 생성과 설정과정이 끝나면 운영체제의 통신관련 정보를 참조해 다양한 작업을 편리하게 할 수 있는 데이터 타입

**윈도우소켓 – (버클리 유닉스에서 개발한 네트워크프로그래밍 인터페이스를 윈도우에서 사용가능하게 만든것)**

윈도우소켓과 유닉스 소켓의 차이점 – 윈도우 소켓은 dll을 통해 대부분의 기능이 제공되므로 dll초기화와 종료작업을 위한 함수가 필요하다.

윈도우 프로그램은 대개 gui를 갖추고 메시지 구동 방식으로 동작, 이를위한 확장 함수가 존재

윈도우는 운영체제 차원에서 멀티스레드를 지원하므로 멀티스레드 환경에서 안정적으로 동작하는 구조와 이를 위한 함수가 필요하다.

윈속의 장점

-유닉스 소켓과 소스코드 수준에서 호환성이 높다. 기존 코드를 이식하여 활용하기 쉽다.

-가장널리 사용하는 네트워크 프로그래밍 인터페이스

-tcp/ip외의 프로토콜도 지원하므로 최소코드 수정으로 응용 프로그램이 사용할 프로토콜 변경가능

-비교적 저수준 프로그래밍 인터페이스이므로 세부제어가 가능하며 고성능 네트워크 프로그램 개발 가능

윈속의 단점

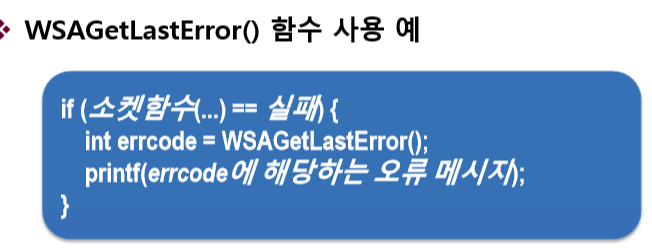
-응용프로그램 수준의 프로토콜을 프로그래머가 직접 설계해야한다.

-서로다른 바이트 정렬 방식을 사용하거나 데이터 처리 단위가 서로다르면 응용프로그램수준에서 데이터를 변환해야한다.

2장

오류처리유형

1. 오류를 처리할 필요가 없는경우 – 리턴값이 없거나 호출시 항상 성공하는 일부소켓함수
2. 리턴값만으로 오류를 처리 – WSAStartup()함수
3. 리턴값으로 오류발생을 확인하고 구체적인 내용은 오류코드로 확인



Err\_display() 함수는 오류발생시 응용프로그램이 종료되어버리는 것을 방지, 개발자 로그 확인용으로 적합

윈속초기화

모든 윈속프로그램은 소켓함수를 호출하기 전에 반드시 윈속 초기화함수인 WSAStartup()을 호출해야함

WSAStartup() 함수는 프로그램에서 사용할 윈속버젼을 요청함으로서 윈속 라이브러리 WS2\_32.DLL을 초기화하는역할

종료할때는 WSACleanup()호출

소켓타입 – 소켓타입은 사용할 프로토콜의 특성을 나타내는 값

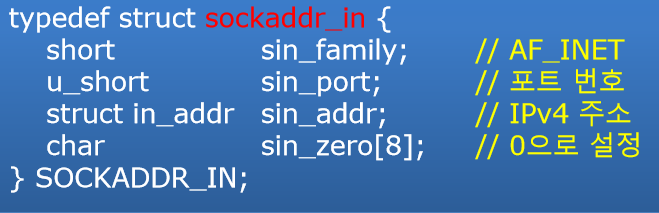
SOCK\_STREAM – 연결형프로토콜, SOCK\_DGRAM – 비연결형프로토콜

소켓생성 예 – TCP : socket(AF\_INET, SOCK\_STRAM, 0) UDP : socket(AF\_INET, SOCK\_DGRAM, 0)

IPv6는 AF\_INET6로 변경

3장

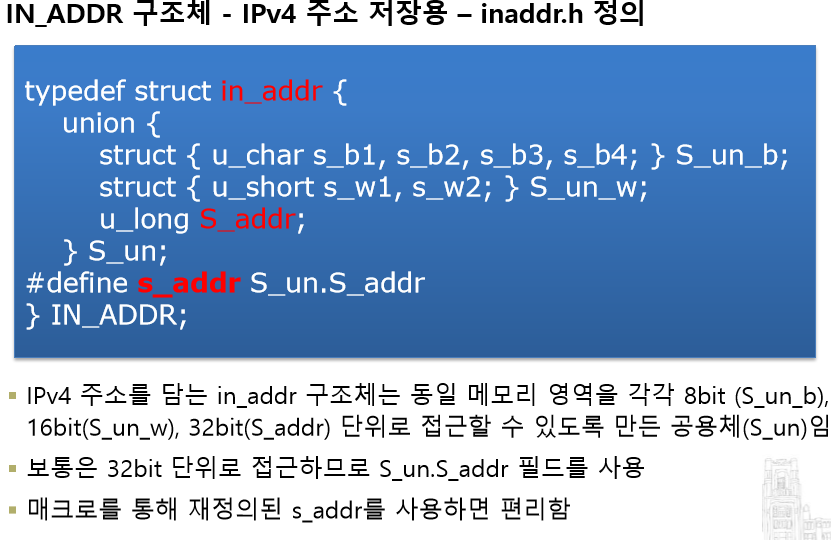
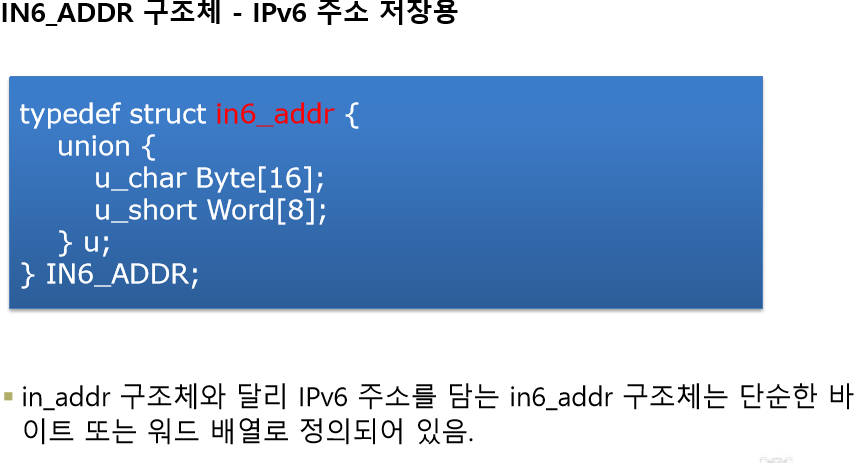
소켓주소구조체 – 네트워크에서 필요로 하는 주소정보를 담고있는 구조체로 다양한 소켓함수의 인자로 사용, 프로토콜 체계에 따라 다양한 형태가 존재한다, 기본형은 SOCKADDR 구조체

 SOCKADDR\_IN 구조체 IPv4전용

sin\_family – 주소 체계를 의미 AF\_INET / AF\_INET6 사용

sin\_port – 포트번호를 의미, 부호없는 16비트 정수값

sin\_addr - ip주소를 의미, 32비트 구조체 (ipv6면 128비트 구조체

) 

소켓 주소 구조체 사용예

-반드시 SOCKADDR 포인터형으로 변환해야한다.

**네트워크 바이트 정렬**

-컴퓨터와 네트워크마다 사용하는 언어형식이 다름. 즉 바이트순서가 다르다

-cpu에따라 레지스터에서 읽어오는 방식이 다름

-네트워크 바이트 정렬은 빅엔디안으로 표준화

모든 수신메시지에 대한 바이트 오더링을 고려해야함 네트워크는 빅엔디안으로 통일

프로토콜 구현을 위해 필요한 정보

-호스트와 라우터가 IP주소의 바이트정렬방식을 약속하지 않으면 아이피주소해석이 달라 라우팅문제 발생

바이트정렬 – 메모리에 데이터를 저장할 때 바이트 순서, cpu 운영체제에 따라 상이함 빅엔디안, 리틀엔디안

바이트 정렬방식으로 고려해야 하는경우

-프로토콜 구현을 위해 필요한 정보들 ip주소 포트번호들에 대한 정렬방식을 약속하지 않으면 ip주소 해석이 달라져문제발생

바이트정렬함수

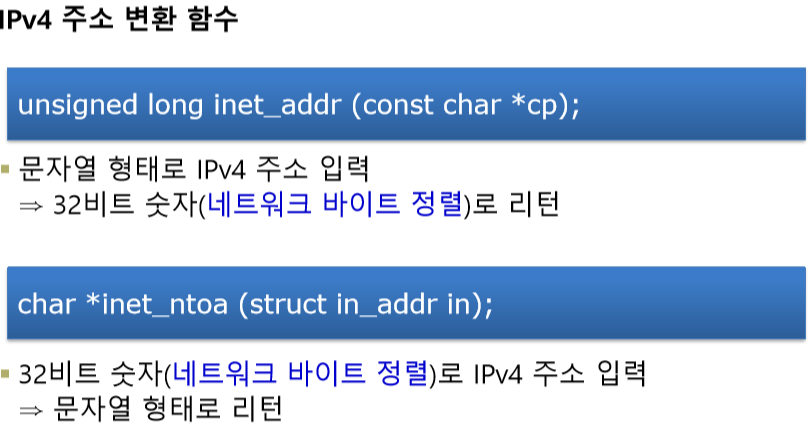
hton\*() 함수는 응용프로그램이 소켓 함수에 데이터를 넘겨주기 전에 호출되며, ntoh\*() 함수는 소켓 함수가 결과로 리턴한 데이터를 응용 프로그램이 출력등의 목적으로 이용 하기 전에 호출

ipv4는 포트번호와 어드레스주소를 네트워크 바이트 정렬

ipv6는 포트번호, 어드레스주소, flowinfo를 네트워크 바이트 정렬

IP주소 변환

-컴퓨터는 이진데이터만 해석가능하기 때문에 인터넷주소(문자열) <-> 이진데이터(네트워크 바이트) 로 변경



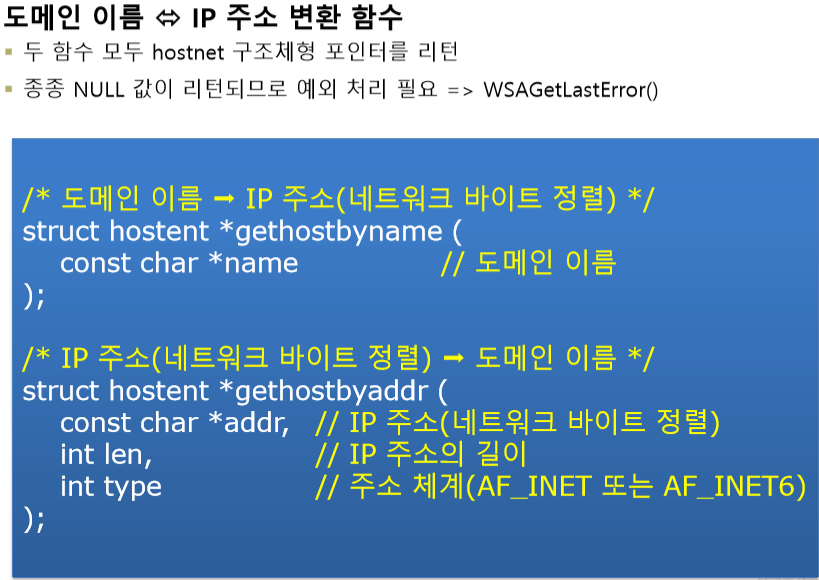
**도메인주소변환**

-도메인 주소는 사람이 쉽게 이해할 수 있는 인터넷주소

-인터넷에서는 이진 인터넷 주소만사용하기 때문에 도메인을 사용하기위해서는 변환시스템 (DNS)가 필요하다.

-Gethostbyname()함수지원

도메인이름 -> 아이피주소의 고유한 식별자 ex) [www.naver.com](http://www.naver.com)



4장 TCP서버 클라이언트

HTTP는 TCP에 기반한 프로토콜이므로 웹서버-클라이언트는 대표적인 TCP서버 클라이언트 프로그램이다.

WinSock -유닉스/리눅스는 소켓을 파일루 구성하지만 Windows는 소켓을 객체로 인식한다.

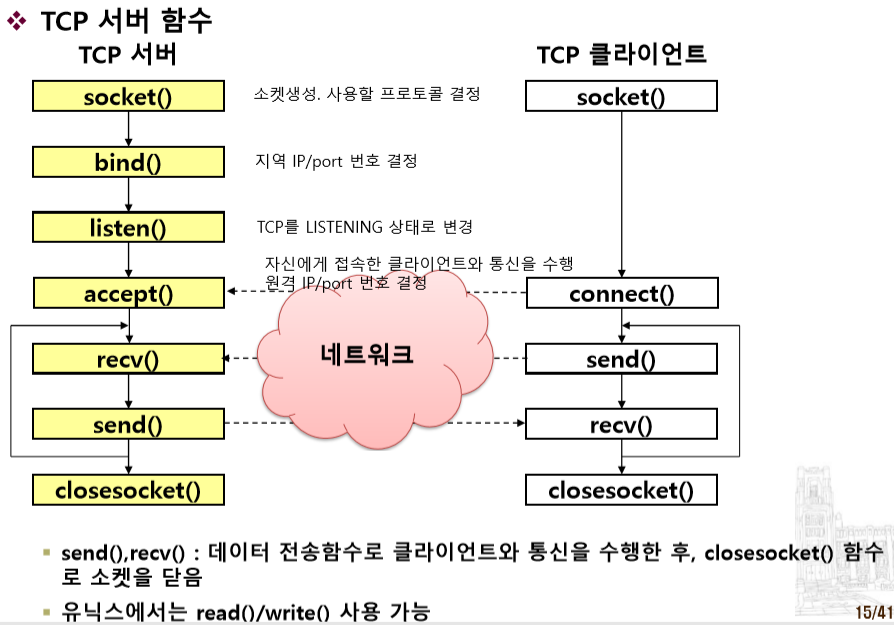
TCP서버 클라이언트 동작원리

소켓으로 통신을 할때 listen소켓, accept소켓, connect소켓세개를 생성해서 통신한다.

TCP연결 절차가 끝나면 새로운 소켓을 생성해서 기존의 소켓이 새로운 클라이언트 접속을 수용하는 용도로 사용

서버에는 총 3개의 소켓이 존재하고 그 중 두개는 접속한 클라이언트와 통신하는 용도로 사용한다

포트번호 설정은 운영체제가 비어있는 포트번호로 설정, 서버가 소켓 2개 클라이언트가 소켓 1개를 생성하여 통신



서버함수

Bind()함수 – 소켓의 지역ip주소와 지역포트번호 결정, 직접적인 소켓 연결 INADDR\_ANY -> 모든 주소를 기다림

Listen() 함수 – 소켓의 TCP포트 상태를 LISTENING으로 변경, 수신대기열 생성, accept()를 통해 연결

Accept()함수 – 접속한 클라이언트와 통신할 수 있도록 새로운 소켓을 생성해서 리턴, 접속한 클라이언트의 IP주소와 포트번호를 알려줌

backlog: 서버가 당장 처리하지 않아도 접속가능한 클라이언트 개수

- 서버가 당장 처리하지 않음, 즉 accept를 호출하지 않음

클라이언트 함수

클라이언트에서는 bind를 사용하지않는다(운영체제에서 알아서 포트번호를 설정해주기때문에)

Connect()함수 – TCP 프로토콜 수준에서 서버와 논리적 연결을 설정 ,소켓구조체를 서버주소로 초기화하여 전달한다.

Send()함수 – 으용프로그램 데이터를 운영체제의 송신버퍼에 복사함으로써 데이터전송, 첫번쩨 인자로 전달되는 소켓의 특성에 따라 블로킹/논블로킹 소켓 값으로 리턴할 수 있음

Recv()함수 – 운영체제의 수진버퍼에 도착한 데이터를 응용프로그램 버퍼에 복사 len이 응용프로그램버퍼보다 크면안된다

송신버퍼는 데이터를 전송하기 전에 임시 저장

수신버퍼는 받은 데이터를 응용프로그램에서 처리하기 전에 임시저장

recv 받은 데이터를 응용프로그램 버퍼로 복사하는것

클라이언트에서 bind를 사용해서 포트를 사용자가 직접 설정해서 사용하면 첫 연결시에는 연결이 되지만

연결이 끝난후 포트번호가 time wait로 대기 상태로 남아있기 때문에 다음연결시에는 포트번호가 사용중이어서

연결이 불가능하다

**5장**

응용프로그램 프로토콜

-응용 프로그램수준에서 주고받는 데이터의 형식과 의미 그리고 처리방식을 정의한 프로토콜

-개발자가 가장 많이 사용할 프로토콜, **소켓 프로토콜처럼 표준화되어있지는 않다.**

메시지 경계 구분 방법

① 송신자는 항상 고정 길이 데이터를 보냄. 수신자는 항상 고정 길이 데이터를 읽음

* 주고받을 데이터의 길이 변동폭이 크지않을 경우에 적합
* 가장긴 데이터를 기준으로 고정길이를 정해서 효율성이 떨어짐

② 송신자는 가변 길이 데이터를 보내고 끝 부분에 특별한 표시(EOR, End Of Record)를 붙임. 수신자는 EOR이 나올 때까지 데이터를 읽음

* 생성될 데이터의 길이를 미리 알수 없을 때 적합
* 데이터중간에 EOR과 똑 같은 패턴이 있으면 데이터가 완전히 전송되지않음
* 데이터를 한바이트씩 읽기 때문에 효율적으로 수신하는 방식의 알고리즘이 없으면 성능이 떨어짐

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

* 일반적으로 권장

④ 송신자는 가변 길이 데이터 전송 후 접속을 정상 종료. 수신자는 recv( ) 함수의 리턴 값 이 0(=정상 종료)이 될 때까지 데이터를 읽음

* 한쪽에서 일방적으로 데이터를 보내는 경우에 적합

⑤ 기타: 파일 데이터등으로 전송

파일을 직접 주고받을때 -> FTP : 신뢰성 있는 파일 전송 프로토콜, TFTP : 신뢰성 없는 파일 전송 프로토콜

xml로 파일 전송하면 파일의 경계 구분이 필요없다. 단점은 데이터 효율이 떨어진다. 따라서 짧은 데이터

를 보낼때 좋다.

구조체 멤버 맞춤 - #pragma pack

-구조체 멤버맞춤을 1바이트경계로 변경한다. (근래엔 많이 사용안함)

서버 성능저하는 줄이는법

1. 하드웨어(cpu)의 비용을 늘리는 방법

2. 서버의 비용을 늘리는 방법

**6장**

**멀티스레드**

동시에 여러 클라이언트를 받기위한 방법

1. 서버가 각 클라이언트와 통신하는 시간을 줄인다 -> 구현이쉽고 가장적게 시스템자원을 사용하지만 처리지연시간이 길어질 수 있음
2. 각 클라이언트를 스레드를 이용해 독립적으로 처리 -> 소켓 입출력 모델에 비해 구현이 쉬우나 가장 많은 시스템자원 사용
3. 소켓 입출력 모델 사용 -> 소수의 스레드를 이용해 다수의 클라이언트를 처리 but 구현이 어려움

교착상태

-서버와 클라이언트의 send와 recv 호출순서가 서로 안맞거나 데이터를 보내지않은 상태에서 양쪽에서 동시에 recv를 호출할때 발생할수있다.

-영원히 일어나지 않을 이벤트를 두개 이상의 프로세스가 기다리는 상황

-해결책 : 데이터송수신부분 잘 성계하기 -> 멀티스레드로 해결

소켓에 타임아웃 옵션 적용 – 구현이 쉬우나 다른방법에 비해 성능이 낮음

넌블로킹 소켓 사용 – 근본적으로 교착상태를 해결하나 구현이 복잡하고 시스템자원 낭비

소켓 입출력 모델 사용 – 넌블로킹 소켓의 단점을 보완 & 교착상태 해결

프로세스가 몸전체 라고 한다면 프로세스는 몸의 일부분(팔, 다리)

멀티스레드 장점 : 하나의 프로세스일때에는 스레드 동기화가 필요없음, 신속하게 원하는 코드를 만들수 있음, 데이터 교환 및 관리가 용이, CPU를 효율적으로 활용할 수 있음, 오래된 시간과 기술발전으로 신뢰성이 높음

단점 : 프로그래밍이 복잡, 디버깅이 어렵다

멀티프로세스 – 프로세스를 여러 개로 복사하는 것

Linix : 프로세스기반 (스레드는 linux, unix, mac등에서 공통으로 사용하기 위한 코드)

Windows : 프로세스 + 스레드기반

Pthread – POSIX 표준을 따르고 호환성이 높다. POSIX는 리눅스 유닉스 맥에서 공동으로 사용가능 범용성이 좋으나 최적화는 안좋다.

윈도우는 독자적인 전용함수를 사용한다.

-자원을 객체로 다룸, 핸들로 자원을 다룸

-create 로 시작하고 HANDLE 을 반환, closeHandle로 닫음

-wait() 함수를 스레드 동기화의 범용 함수로 사용

용어정리

프로세스 – 코드 데이터, 리소스를 파일에서 읽어들여 윈도우가 할당한 메모리 영억에 담고 있는 컨테이너 정작인개념

스레드 - cpu시간을 할당받아 프로세스 메모리 영역에 있는 코드르 수행. 동적인개념

주 스레드 or 메인스레드 – 프로그램실행시 최초로 생성되는 스레드

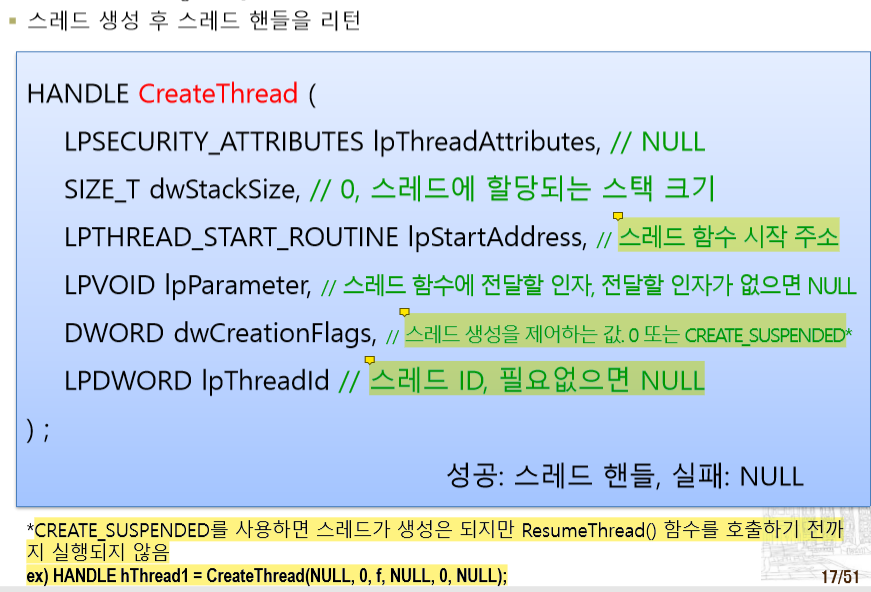
멀티스레드 응용프로그램 – 응용프로그램에서 주스레드와 별도로 동시에 수행하고자 하는 스레드를 생성해 작업수행

컨텍스트 전환( Context switch ) – 하드웨어와 소프트웨어의 협력으로 이루어지는 스레드 실행 상태의 저장

스레드생성에 필요한 요소

-스레드함수의 시작주소

-스레드 함수 실행시 사용할 스택의 크기 -> 모든 함수 실행중 인자 전달과 변수 할당을 위해 스택 필요, 스레드실행에 필요한 스택 생성은 운영체제가 자동으로 설정, 스택크기만 정의하면 됨



소켓에서는 소켓반환값

0은 바로실행 suspended는 일단 대기

윈도우는 핸들을 사용하기 때문에

한 프로세스에서 스레드는 32비트 컴퓨터의경우 2GB마다 2048개 만들수있다

스레드 종료방법

-스레드 함수 리턴, 스레드함수 내에서 ExitThread함수호출, 다른스레드가 TerminateThread함수호출, 주 스레드가 종료하면 모든스레드 종료

용어정리

스레드 스케줄링 – 윈도우가 각 스레드에 cpu 시간을 적절히 분배하기 위한 정책

우선순위 클래스 – 프로세스속성으로 같은 프로세스가 생성한 스레드는 우선순위 클래스가 모두같음

우선순위 레벨 – 스레드 속성으로 같은 프로세스에 속한 스레드 간 상대적인 우선순위 결정할 때 사용

기본우선순위 – 우선순위 클래스와 우선순위 레벨을 결합한 것, 스레드 스케쥴링에 사용

윈도우 운영체제에서 제공하는 우선순위 레벨 – 우선순위 레벨은 스레드 속성으로 같은 프로세스에 속한 스레드 간 상대적인 우선순위 정할 때 사용

WaitForSingleObject() 함수 ▪ 특정 스레드가 종료할 때까지 기다리기



WaitForMultipleObject()함수 – 여러 스레드가 종료하기를 기다리려면 이 함수한번 호출로 끝낼수있다.

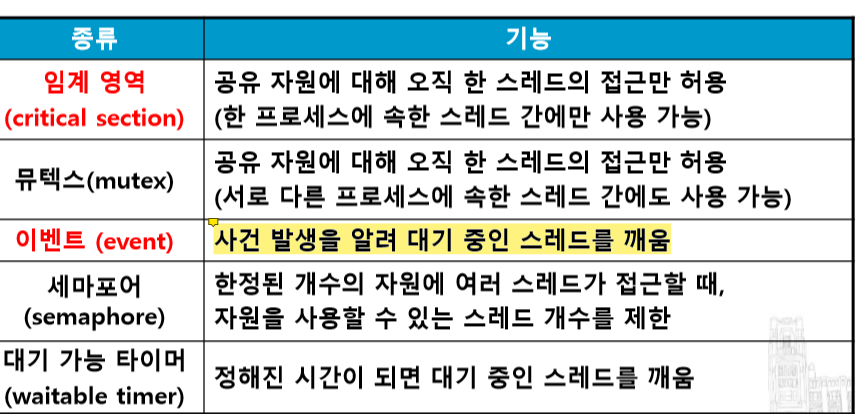
스레드 실행중지함수 SuspendThread() 함수를 호출했을때는 ResumeThread()함수를 사용해야 스레드가 다시시작됨

소켓과 연관된 주소 정보 얻기

-스레드 함수에 소켓만 전달한 경우에는 별도의 주소 정보가 없으므로, 소켓을 통해 주소 정보를 얻는 기능 구현이 필요

-getpeername()은 원격 IP/Port 리턴, getsockname()은 지역 IP/port 리턴

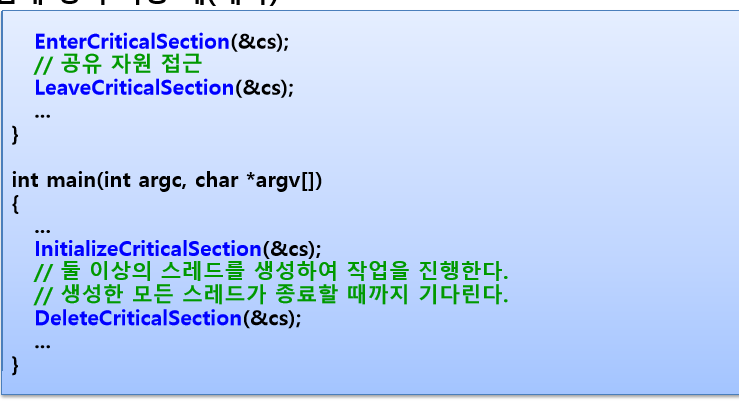
멀티스레드에서 두개 이상의 스레드가 공유데이터에 접근하면 다양한 문제가 발생한다. -> 스레드 동기화 필수

 이벤트는 사용가능할 때 활성화

임계영역

-둘 이상의 스레드가 공유자원에 접근할 때, 오직 한 스레드만 접근을 허용해야하는 경우에 사용

-특징: 프로세스의 유저 메모리 영역에 존재하는 단순한 구조체이므로 한 프로세스에 속한 스레드 간 동기화에만 사용, 일반동기화 객체보다 빠르고 효율적



이벤트

▪ 사건 발생을 다른 스레드에 알릴 때 사용

▪ 한 스레드가 작업을 완료한 후 기다리고 있는 다른 스레드에 알릴 때 사용

❖ 이벤트를 사용하는 전형적인 절차

➊ 이벤트를 비신호 상태로 생성

➋ 한 스레드가 작업을 진행하고, 나머지 스레드는 이벤트에 대해 Wait\*( ) 함수를 호출해 이벤트가 신호 상태가 될 때까지 대기 (sleep)

➌ 스레드가 작업을 완료하면 이벤트를 신호 상태로 바꿈

➍ 기다리고 있던 스레드 중 하나 혹은 전부가 깨어남 (wakeup)

멀티스레드 동작원리

스레드1이 실행 -> 스레드1이 실행중지하고 실행상태 저장 후 스레드2의 상태를 복원 -> 스레드2 실행

-> 스레드2 실행 중지하고 실행상태 저장 후 스레드1 의 상태복원

**7장**

❖ TCP와 UDP의 공통점

▪ 전송 계층 프로토콜

▪ 포트 번호를 이용해 주소를 지정

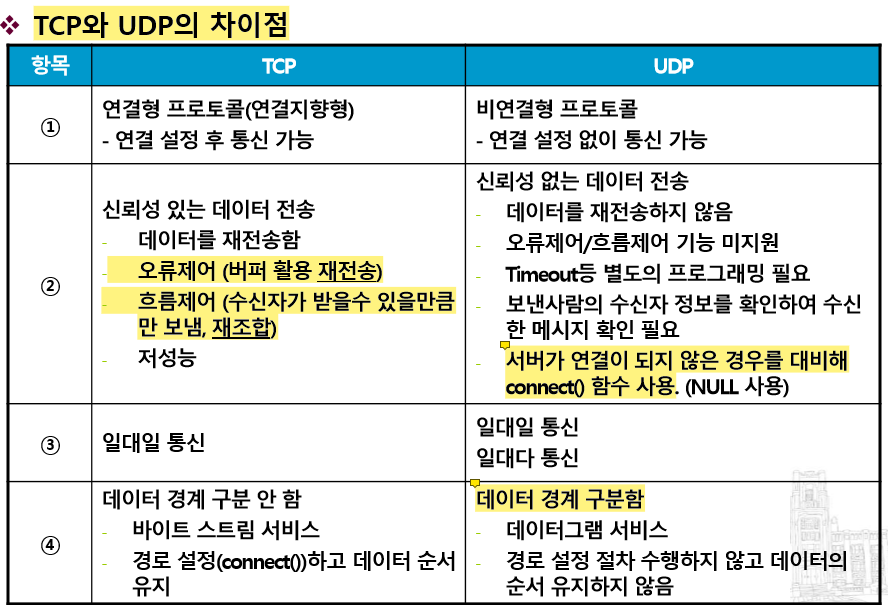
• 포트 번호를 이용하여 종단(응용 프로그램)간 전송

• 두 응용 프로그램이 TCP나 UDP를 이용해 통신하려면 반드시 포트 번호를 결정해야 함

▪ 데이터 오류를 체크

• 데이터 위변조 확인

• IP와 달리 TCP와 UDP는 헤더는 물론이고 데이터에 대한 오류도 체크함

.

ucp는 서버가 작동중인지 아닌지 알수없기때문에

connect함수를 사용한다. 서버가 접속불가능일때 ICMP에서 오류를 리턴해준다.

❖ UDP의 특징

▪ 연결 설정을 하지 않으므로 connect( ) 함수 불필요

▪ 프로토콜 수준에서 신뢰성 있는 데이터 전송을 보장하지 않으므로, 필요하다면 응용 프 로그램 수준에서 신뢰성 있는 데이터 전송 기능을 구현해야 함

▪ 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

▪ 데이터그램(메시지) 단위 전송이며 하나의 데이터그램은 65535바이트 크기로 제한됨 (65535 바이트 이상의 크기는 잘라서 보내야 함)

▪ Not ordered

▪ 높은 성능 - 신뢰성을 희생해서 성능 확보

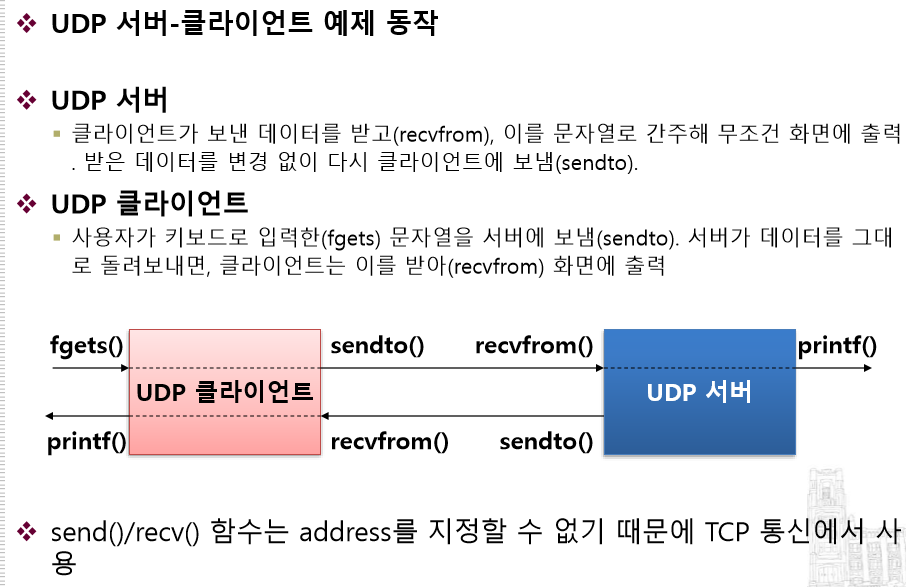
UDP를 사용하는 이유

-성능을 높일수있다, 실시간 멀티미디어 서비스 프로그램 개발이 용이, 개발이 쉽다

- Agent & Manager 모델에 적합

- 주로 파일, 스트리밍 콘텐츠 전송에 사용

- TCP가 전화로 비유되고 UDP는 우편으로 비유할수있음



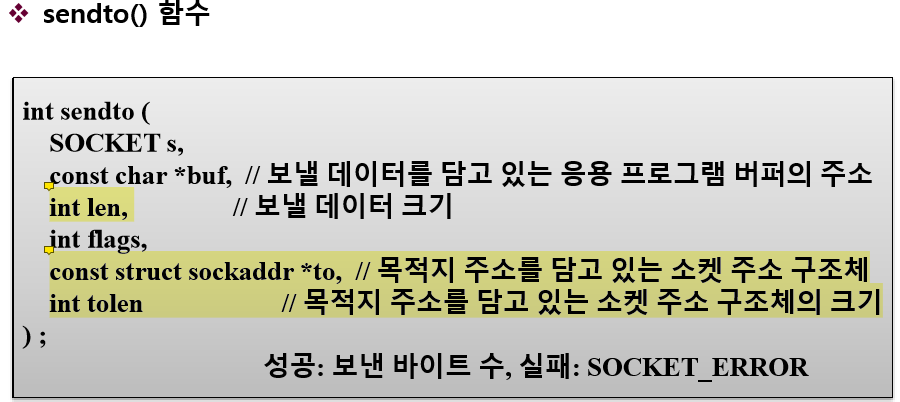
❖ 소켓 통신을 위해 결정해야 할 요소

① 프로토콜 - 통신 규약. 소켓을 생성할 때 결정

② 지역 IP 주소와 지역 포트 번호 - 서버 또는 클라이언트 자신의 주소

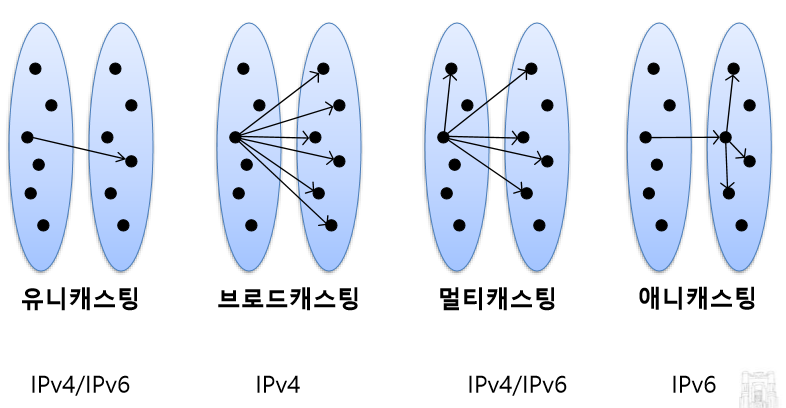
③ 원격 IP 주소와 원격 포트 번호 - 서버 또는 클라이언트가 통신하는 상대의 주소

UDP는 TCP와 다르게 송신버퍼가 필요없다.



Int len - udp에서는 데이터길이가 0, NULL이어도 전송할 수 있다. Tcp는 데이터길이가 0이면 접속종료로 간주

목적지 주소를 담고있는 소켓 주소 구조체 = tcp의 send와 다른점



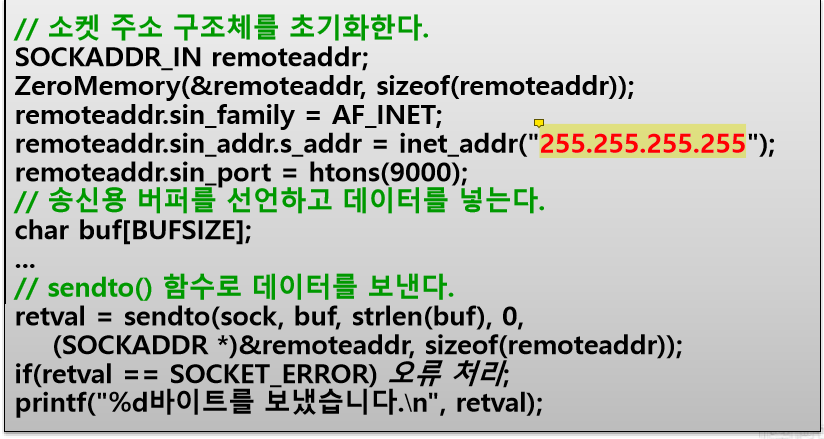
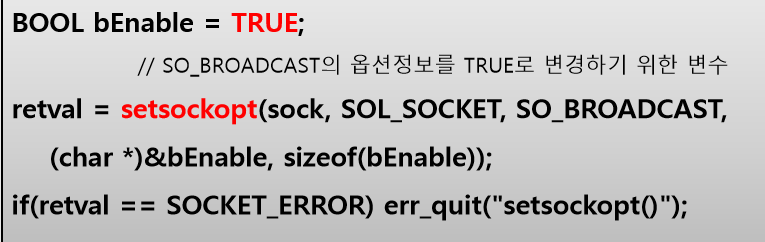
❖ 브로드캐스팅 개념

▪ 송신자가 보낸 데이터 하나를 다수의 수신자가 받는 방식

• 데이터 복사본을 여러 개 만들어 보내는 것이 아니므로 송신자 관점에서 보면 상당히 효율적인 기술임

브로드캐스트 데이터를 보내기 위한 절차

1. 브로드캐스팅 활성화
2. 브로드캐스트 주소로 데이터 보냄



255.255.255.255 = 자신과 연결되어있는 모든 머신들과 통신하겠다는 의미

