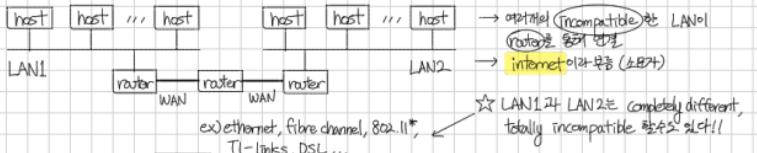
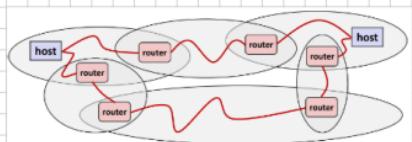




## Next Level: Internets (WAN)



## Logical Structure of an Internet



- o Ad-hoc interconnection of networks.
  - no particular topology.
  - vastly different router & link capacities
  - + Adhoc은 노드들에 의해 자율적으로 형성된다.
  - 기본 구조가 있는 네트워크
  - "No particular topology"는 컴퓨터도 다를고, 링크 콜링도 다를 수 있다.
- o send packets from source to destination by hopping through networks
  - router forms bridge from one network to another
  - different packets may take different routes

## The Notion of an Internet Protocol

여러 LAN을 과 WAN들은 서로 다른 다른 네트워크 기술을 기반으로 함 (Incompatible)

하지만 (Protocol) software 을 각 router와 host에 설치.

- o 각 네트워크에서 다른 네트워크로 정보가 전달되려면  
host와 router가 서로 다양한 역할에 대한 set of rules
- o smooth out differences between different networks

## What Does an Internet Protocol Do?

### (1) naming scheme 제공

→ host address이 있고 uniform format

→ 각 host와 router는 고유한 unique하게 구별할 수 있는 확장한 형태의 internet address를 갖게 됨

### (2) Delivery Mechanism을 제공

→ standard transfer unit인 packet 을 정의

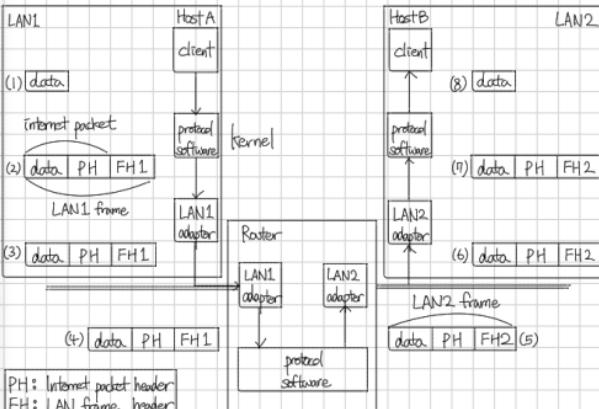
→ packet은 header + payload로 구성

t) ethernet segment에서는 frame 단위로  
internet에서는 packet 단위로

packet header: packet size, source, destination address 같은 정보

packet payload: 실제 보내고자 하는 data

## Transferring Internet Data Via Encapsulation



예시) Host A:client / Host B:server

- o Internet: 서로 다른 기술을 가진 네트워크들을 프로토콜 소프트웨어로 연결
- o LAN A에 속한 Host A가 Protocol을 이용해 Host B에게 보낸 데이터(Payload)와 header를 둘러 packet을 형성, 네트워크 단위로 chunk 크기의 Frame 단위로, Frame header를 둘러 Frame 형성
- o Adapter는 각각의 router에게 전송. Router는 Protocol Software 통해, 여기서是从 LAN A에 있는 청취로 변경 Frame Header를 각각에게 이를 수령
- o LAN B는 청취한 후, LAN 내부의 Protocol을 거쳐 해제됨

## Global IP [Internet (upper case)]

→ Internet의 가장 유용한 예시

→ TCP/IP Protocol family를 기반으로.

## o IP (Internet Protocol)

host 간에 전송 가능한 scheme (host address)을 제공  
Unreliable delivery capability of packets (datagrams) from host-to-host

## o UDP (Unreliable Datagram Protocol)

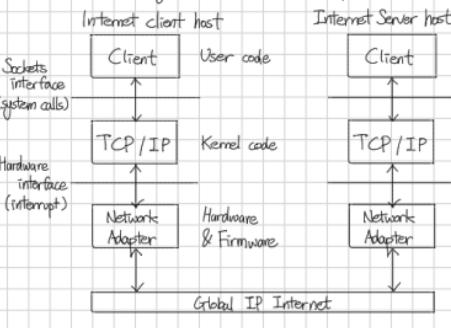
IP protocol을 이용한 process-to-process + unreliable datagram delivery

## o TCP (Transmission Control Protocol)

IP protocol을 이용한 process-to-process + reliable byte stream 제공

2단계로 byte stream 처리?

## Hardware &amp; Software Organization of an Internet Application



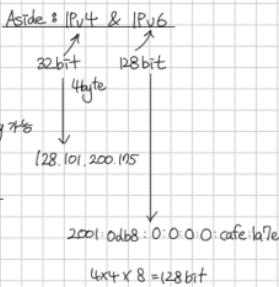
→ network adapter는 hardware이.  
2단계로 host device 및 firmware.  
↳ hardware embedded software인 firmware로.  
→ OS Kernel로 TCP/IP Protocol로.  
↳ OS 활용하기 위해 Host of Application이다.  
↳ system call로.  
↳ "socket interface"  
↳ 예전은 UNIX File I/O는 파일형.  
(A/B)  
+> Asynchronous interrupt 방식임.

## A Programmer's View of the Internet

(1) Host는 32bit의 IP address로 mapping되는 32bit.  
ex) 128.2.203.179 → 48bit × 4.

(2) IP 주소의 경우는 Internet domain names로 mapping되는 128bit mapping 74bit.  
ex) 128.2.203.179 → www.cs.cmu.edu  
many-to-many?

(3) Internet은 process로 Internet process로 connection되는 소통 가능



## (1) IP Addresses

→ IP Address 2진수로 32bit IP address가 저장됨.

설명 예제 10110111... Network Byte Order (Big Endian : MSB가 먼저 전송에 포함)로 저장  
그리고 CPU는 대체로 Little-Endian → 전송은, 표준화되어 있음.

struct in\_addr {  
 uint32\_t s\_addr; // 네트워크  
};

→ "network byte order" + "host byte order"간의 변환

#include <arpa/inet.h>

uint32\_t htonl((uint32\_t hostlong);  
uint16\_t htons((uint16\_t hostshort);  
short

network byte order 3.

uint32\_t ntohs((uint32\_t hostlong);  
uint16\_t htonl((uint16\_t hostshort);  
short

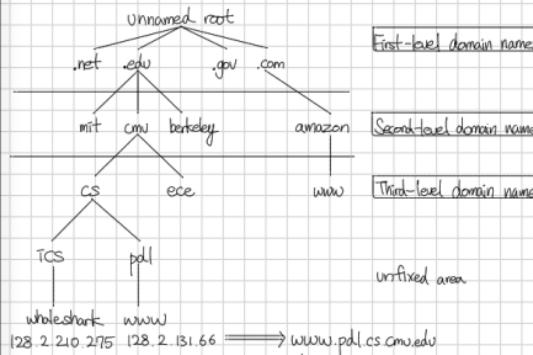
host byte order 3.

## (2) Dotted Decimal Notation

ex) 0x00b27f2 → 128.2.194.242

getaddrinfo 함수와 getnameinfo 함수를 통해 IP주소와 Dotted Decimal Notation 간의 변환 수행 가능

## (2) Internet Domain Names



## → Domain Naming System (DNS)

- Global IP Internet은 IP address 와 Domain Name의 mapping 관계를 (DNS)에 저장
- 인터넷 관점에서 DNS 데이터베이스는 수많은 host entry의 집합
- host entry는 IP주소와 domain name의 mapping 관계이며
- host entry는 domain name, IP주소의 클래스이다.

### Properties of DNS Mappings

- nslookup은 기본
- host.local.host.localhost를 넣으면 127.0.0.127.0.0.1로 domain name은 local host와 같다.  
hostname은 local host의 이름(localhost) 자체가 domain name으로 host마다 다름.

```
linux> nslookup localhost
Address: 127.0.0.1
      linux> hostname
      whaleshark.ics.cs.cmu.edu
      linux> hostname -i
      128.2.210.175
```

### one-to-one mapping

```
linux> nslookup whaleshark.ics.cs.cmu.edu
Address: 128.2.210.175
```

### Multiple domain name mapped to same IP address

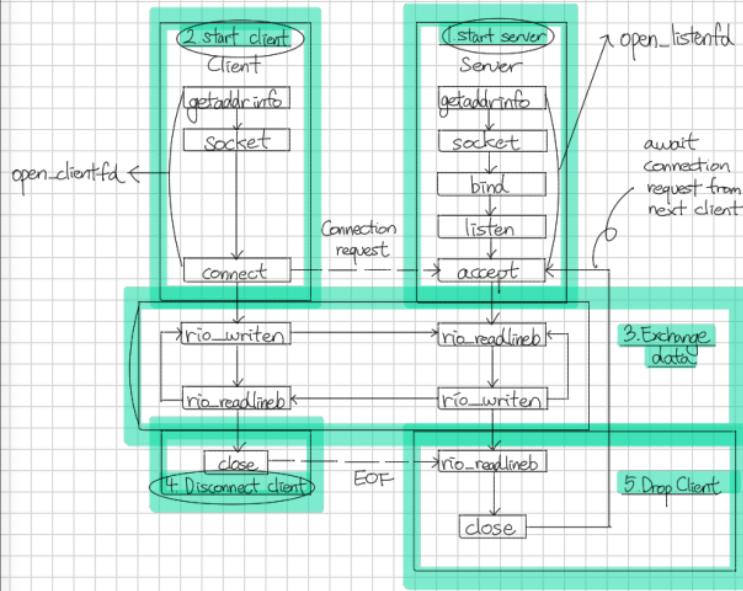
```
linux> nslookup cs.mit.edu
Address: 18.62.1.6
linux> nslookup eecs.mit.edu
Address: 18.62.1.6
```

### Multiple domain names mapped to multiple IP addresses

Some valid domain names don't map to any IP address

```
linux> nslookup www.twitter.com
Address: 199.16.156.6
Address: 199.16.156.70
Address: 199.16.156.102
Address: 199.16.156.230
      linux> nslookup ics.cs.cmu.edu
      *** Can't find ics.cs.cmu.edu: No answer
      linux> nslookup twitter.com
      Address: 199.16.156.102
      Address: 199.16.156.230
      Address: 199.16.156.6
      Address: 199.16.156.70
```

## Sockets interface



socket 함수: socket descriptor 생성

bind 함수: socket과 server의 경로를 연결 (선택 host와/or server 연결 정보와 socket descriptor 연결)

listen 함수: 어떤 컴퓨터로부터 요청이 와도 수락할 수 있게 대기 상태에 들어가는 함수

accept 함수: Server socket과 client를 연결하는 함수

connect 함수: Server에 연결 요청을 하는 함수

Server가 반응한 대기열에 넣어다가 대기하면 accept

### (3) Internet Connections

→ client와 server는 connection 객체 byte stream을 통해 M2M 통신 가능

connection의 3단계 정리

① **Point-to-Point** : Process들을 각각 point로 View, process의 pair끼리 connection을 형성한다.

② **Full-Duplex** : data connection의 양방향에서 동시에 전송 가능

③ **Reliable** : Stream of bytes 순서가 그대로 유지.

→ connection의 양 끝 endpoint는 socket으로 표기.

\* Socket의 주소: IP address: port의 조합

port는 16bit integer: 프로토콜 번호 + 4부

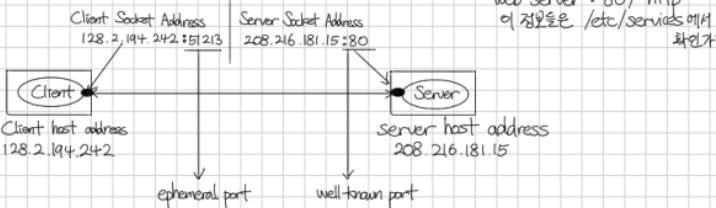
→ port에는 2 종류.

1) Ephemeral port: OS Kernel이 Client의 Connection Request가 발생할 때 알아서 자동으로 할당  
well-known port 대비로 port number

2) Well-known port: Server에 의해 재利用率 서비스와 연결된, 미리 예상한 port number

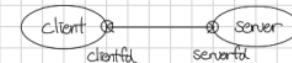
→ Anatomy of a Connection

connection은 양쪽 endpoint socket의 socket address의 pair로 4부



### (4) Sockets Interface

→ UNIX 1.10를 사용해 network application을 만드는데  
use set of system-level functions (system calls)



#### Socket API

→ Kernel에게 endpoint of connection

→ Application에게는 file descriptor

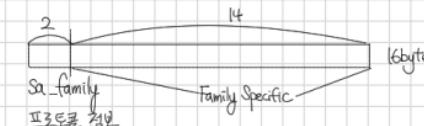
→ Client와 Server는 socket descriptor를 read/write 할 때 communicate

→ 일반적인 regular file I/O와 socket I/O의 차이: application에서 이를 이용하기 open하지

#### Socket Address Structures

Generic socket address (16bytes)

```
struct sockaddr {
    uint16_t sa_family; // 프로토콜 정보
    char     sa_data[4]; // 가족 정보
};
```



typedef struct sockaddr SA;

→ 예제: echo는 sockaddr\_in을 포드로 정의할 수 있어 유용

2004년 connect(), bind(), accept() 같은 함수들은 인터페이스를 하나로 통합하여 했음. (IPv4와 IPv6의 유통 소켓API)

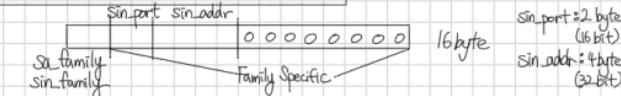
2004년 매개변수가 sockaddr로 선언되어 있어서 헷갈릴 때마다 (struct sockaddr \*)

#### Internet (IPv4) specific socket address (16 bytes)

```
struct sockaddr_in {
    uint16_t sin_family; // 주소계정 필드 (AF_INET)
    uint16_t sin_port; // 포트 정보 저장 (network byte order)
    struct in_addr sin_addr; // IPv4 주소 저장 (network byte order)
    unsigned char sin_zero[8]; // A必定히 있는 필드, 0으로 채운다.
};
```

```
+)-> struct in_addr {
    uint32_t s_addr; // 네트워크 주소
};
```

이전 내용에 나왔던 것처럼  
IP 주소만 저장하는 구조이다.



prototype(218): int connect(int clientfd, (struct sockaddr \*)addr, socklen\_t addrlen);

caller: struct sockaddr\_in \*servaddr;

connect (clientfd, (struct sockaddr \*)servaddr, addrlen);

) type casting

+/-의 차이로 충돌

### Host and Service Conversion : getaddrinfo

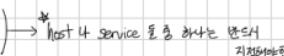
- hostname, host address, port, service name은 반드시 socket address 구조체에 반영
- client 쪽 hostname은 www.google.com (예제) → socket, connect 함수에 쓰임  
port number는 "80" → socket address structures 반영
- server 쪽은 "[listening port number]" → socket, bind 함수에 쓰임  
socket address structures 반영

→ 단점 ① restraint ← thread programming에서 사용할 때 험  
 ② portable한 'Protocol Independent'한 code 작성 가능 (IPv4, IPv6 적용 가능)

→ 단점 : somewhat complex → 그나마 사용자에게 험

### int getaddrinfo(const char \*host, const char \*service, const struct addrinfo \*hints, struct addrinfo \*\*result)

인자들) host domain 이름 또는 dotted IP address



service service name (ex: "http") 또는 port number (ex: "80")

socket address

hints getaddrinfo가 반환하는 소켓 주소 목록의 세부옵션 설정

ai\_family, ai\_socktype, ai\_protocol, ai\_flags는 설정 가능 / 반드시 0 or NULL

bitmask임.

#### • AI\_ADDRCONFIG

- 내 컴퓨터가 지원하는 주소 타입만 가져와
- ex) 컴퓨터가 IPv4만 설정되어 있으면 IPv4 주소만 반환해줌
- IPv6 주소에서 쓰면 풀됨.
- ex) 컴퓨터가 지원하는 주소 타입은 IPv6이지만도 불구하고 IPv4 주소만 가져와야 시간 낭비

#### • AI\_NUMERICSERV

- service라는 문자열(포트번호)으로 처리됨.
- 예전에는 문자열(포트번호)으로 처리됨.
- 예전에는 "http" 같은 문자열을 /etc/services에서 찾아옴.

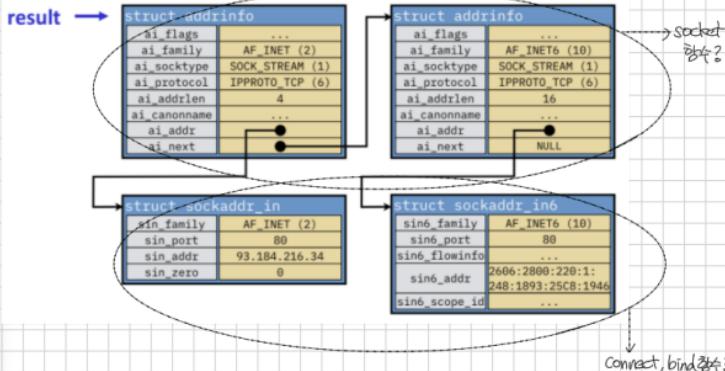
#### • AI\_PASSIVE

- "inetd이나 리스닝 소켓 만화"에
- 서버 프로그램에서 일반적으로 중요
- host argument는 NULL여야함.

### void freeaddrinfo(struct addrinfo \*result);

### const char \*gai\_strerror (int encode);

```
struct addrinfo{
    int ai_flags;
    int ai_family;
    int ai_socktype;
    int ai_protocol;
    char *ai_canonname;
    size_t ai_addrlen;
    struct sockaddr *ai_addr;
    struct addrinfo *ai_next;
};
```



Client: 연결리스트를 순회하며, socket에 대한 요청과 연결의 성공 여부까지 socket address를 출력

Server: 연결리스트를 순회하며, socket에 대한 요청과 binding의 성공 여부까지 socket address를 출력

### Host and Service Conversion : getnameinfo

- getaddrinfo의 inverse
- gethostbyaddr, getservbyport를 대체
- reentrant & protocol independent

```
int getnameinfo (const SA *sa, socklen_t salen,
                char *host, size_t hostlen,
                char *serv, size_t servlen,
                int flags);
```

→ socket addr ) in  
 → host ) out  
 → service ) out

### Conversion Example.

```
#include "csapp.h"

int main(int argc, char **argv)
{
    struct addrinfo *p, *listp, hints;
    char buf[MAXLINE];
    int rc, flags;

    /* Get a list of addrinfo records */
    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_INET; /* IPv4 only */
    hints.ai_socktype = SOCK_STREAM; /* Connections only */
    if ((rc = getaddrinfo(argv[1], NULL, &hints, &listp)) != 0)
        fprintf(stderr, "getaddrinfo error: %s\n", gai_strerror(rc));
    exit(1);
}

/* Walk the list and display each IP address */
flags = NI_NUMERICHOST; /* Display address instead of name */
for (p = listp; p; p = p->ai_next) {
    Getnameinfo(p->ai_addr, p->ai_addrlen,
                buf, MAXLINE, NULL, 0, flags);
    printf("%s\n", buf);
}
whaleshark> ./hostinfo localhost
127.0.0.1

/* Clean up */
Freeaddrinfo(listp);
whaleshark> ./hostinfo whaleshark.ics.cs.cmu.edu
128.2.210.175

exit(0);
}

whaleshark> ./hostinfo twitter.com
199.16.156.230
199.16.156.38
199.16.156.102
199.16.156.198

whaleshark> ./hostinfo google.com
172.217.15.110
2607:f8b0:4004:802::200e
```

요약) 시스템에서는 getaddrinfo 함수를 호출해 IP address + Port Number를萃取 정보얻어옴.

→이걸 이용해 socket 만들고

→만든 socket을 binding

→binding 된 socket은 listening : server가 client의 connection request를 받게됨.

Client가 connection request

## Sockets Interface (Client/Server): socket

```
int socket (int domain, int type, int protocol)
```

→ ex) int clientfd = Socket (AF\_INET, SOCK\_STREAM, 0);

→ socket descriptor 양식

### domain

- AF\_INET : IPv4 주소
- AF\_INET6 : IPv6 주소
- AF\_UNSPEC : 다른 것들

### type

- SOCK\_STREAM : TCP
- SOCK\_DGRAM : UDP
- SOCK\_RAW : raw, ICMP, customized
- 0 : system default 사용

### protocol

- IPPROTO\_TCP
- IPPROTO\_UDP

```
getaddrinfo (char *host, char *service, struct addrinfo *hints, struct addrinfo **res);
```

```
int clientfd = Socket (res->ai_family, res->ai_socktype, res->ai_protocol);
```

## Sockets Interface (Client/Server): connect

client's connect 를 통해 server의 connection request를 보낸다. \* blocking 상태.

server의 request는 accept()로 connection이 되면 fd를

client의 clientfd에 할당된다.

```
int connect (int clientfd, SA *addr, socklen_t addrlen);
```

```
struct sockaddr {
    uint16_t sa_family;
    char sa_data[14];
};
```

```
struct sockaddr_in {
    uint16_t sin_family;
    uint16_t sin_port;
    struct in_addr sin_addr;
    unsigned char sin_zero[8];
};
```

```
struct addrinfo {
    ...
    struct sockaddr *;
    ...
};
```

type casting

result

```
getaddrinfo(... struct addrinfo **result...);
```

결과: (clientaddress : ephemeral port number, addr.sin\_addr : addr.sin\_port)

## client code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
```

```
int main() {
    struct addrinfo hints, *res;
    int sockfd;
```

### // 1. hints 구조체 초기화

```
memset(&hints, 0, sizeof(hints));
hints.ai_family = AF_UNSPEC; // IPv4 or IPv6 모두 사용
hints.ai_socktype = SOCK_STREAM; // TCP 스트림 소켓
```

### // 2. 주소 정보 찾기

```
int status = getaddrinfo("www.example.com", "80", &hints, &res);
if (status != 0) {
    fprintf(stderr, "getaddrinfo error: %s\n", gai_strerror(status));
    return 1;
}
```

### // 3. 소켓 생성

```
sockfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
if (sockfd == -1) {
    perror("socket");
    freeaddrinfo(res);
    return 2;
}
```

### // 4. (선택) 연결 시도

```
if (connect(sockfd, res->ai_addr, res->ai_addrlen) == -1) {
    perror("connect");
    close(sockfd);
    freeaddrinfo(res);
    return 3;
}
```

```
printf("Connected to www.example.com\n");
```

```
// 5. 자원 해제
freeaddrinfo(res);
close(sockfd);
return 0;
}
```

### Sockets interface (Client / Server): bind.

→ int bind(int sockfd, SA \*addr, socklen\_t addrlen);  
 → 프로세스 socket descriptor를 가지는 sockfd를 이용해 데이터를 전송하는 것.  
 connection에서 주로 byte stream을 다룰 수 있게 함

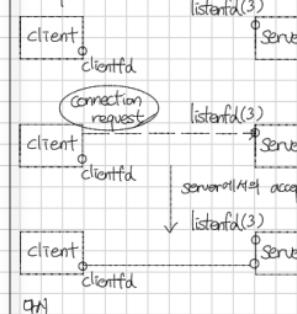
ex) int listenfd = socket(AF\_INET, SOCK\_STREAM, 0);

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

### Sockets interface (Client / Server): listen

Kernel은 default 설정으로 인해 socket descriptor 만들기 가능으로 기본

### accept illustrated



### ① client: connect

Server의 listenfd를 통해 상호  
 Server는 listen fd, accept를 호출해  
 connection request를 처리 가능함.

### ② client: connect 후에 listenfd를 통해 connection request 처리

OS가 queue에 넣는다.  
 → Server는 dequeue하여 connection을 처리함.

### ③ backlog 설정, server는 accept를 통해 connection을 받아옴.

connection은 queue에.  
 → connection과 clientfd는 polling channel로.  
 → client는 clientfd를 통해 연결, server는 connection을 통해 관리함.

### connected vs listening descriptors

#### listening descriptor

- client가 connection requests를 처리할 endpoint
  - HTTP는 서버의 청진 포트 8080이, Server의 lifetime은 짧아
- ∴ client가 connection을 concurrent하게 이용하기 위해 필요

#### connected descriptor

- client와 server가 connection의 endpoint
- server는 connection을 accept로 처리할 때만 만들어짐.
- client는 clientfd를 통해 연결, server는 connection을 통해 관리함.

### Sockets Helper: open\_clientfd

```
int Open_clientfd(char *hostname, char *port) {
    int clientfd;
    struct addrinfo hints, *listp, *p;

    /* Get a list of potential server addresses */
    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_socktype = SOCK_STREAM; /* Open a connection */
    hints.ai_flags = AI_NUMERICSERV; /* ...using numeric port arg. */
    hints.ai_flags |= AI_ADDRCONFIG; /* Recommended for connections */
    Getaddrinfo(hostname, port, &hints, &listp);

    /* Walk the list for one that we can successfully connect to */
    for (p = listp; p; p = p->ai_next) {
        /* Create a socket descriptor */
        if ((clientfd = socket(p->ai_family, p->ai_socktype,
                               p->ai_protocol)) < 0)
            continue; /* Socket failed, try the next */

        /* Connect to the server */
        if (connect(clientfd, p->ai_addr, p->ai_addrlen) != -1)
            break; /* Success */
        Close(clientfd); /* Connect failed, try another */
    }

    /* Clean up */
    Freeaddrinfo(listp);
    if (!p) /* All connects failed */
        return -1;
    else /* The last connect succeeded */
        return clientfd;
}
```

csapp.c

### Sockets Helper: open\_listenfd

```
int Open_listenfd(char *port)
{
    struct addrinfo hints, *listp, *p;
    int listenfd, optval=1;

    /* Get a list of potential server addresses */
    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_socktype = SOCK_STREAM; /* Accept connect. */
    hints.ai_flags = AI_PASSIVE | AI_ADDRCONFIG; /* ...on any IP addr */
    hints.ai_flags |= AI_NUMERICSERV; /* ...using port no. */
    Getaddrinfo(NULL, port, &hints, &listp);

    /* Walk the list for one that we can bind to */
    for (p = listp; p; p = p->ai_next) {
        /* Create a socket descriptor */
        if ((listenfd = socket(p->ai_family, p->ai_socktype,
                               p->ai_protocol)) < 0)
            continue; /* Socket failed, try the next */

        /* Eliminates "Address already in use" error from bind */
        Setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
                   (const void *)&optval, sizeof(int));

        /* Bind the descriptor to the address */
        if (bind(listenfd, p->ai_addr, p->ai_addrlen) == 0)
            break; /* Success */
        Close(listenfd); /* Bind failed, try the next */
    }

    /* Clean up */
    Freeaddrinfo(listp);
    if (!p) /* No address worked */
        return -1;

    /* Make it a listening socket ready to accept conn. requests */
    if (listen(listenfd, LISTENQ) < 0) {
        Close(listenfd);
        return -1;
    }
    return listenfd;
}
```

csapp.c

### Echo Client: Main routine.

```
#include "csapp.h"

int main(int argc, char **argv)
{
    int clientfd;
    char *host, *port, buf[MAXLINE];
    rio_t rio;

    host = argv[1];
    port = argv[2];

    clientfd = Open_clientfd(host, port);
    Rio_readinitb(&rio, clientfd);

    while (Fgets(buf, MAXLINE, stdin) != NULL) {
        Rio_writen(clientfd, buf, strlen(buf));
        Rio_readlineb(&rio, buf, MAXLINE);
        Pputs(buf, stdout);
    }
    Close(clientfd);
    exit(0);
}
```

### Iterative Echo Server: Main Routine.

```
#include "csapp.h"
void echo(int connfd);

int main(int argc, char **argv)
{
    int listenfd, connfd;
    socklen_t clientlen;
    struct sockaddr_storage clientaddr; /* Enough room for any addr */
    char client_hostname[MAXLINE], client_port[MAXLINE];

    listenfd = Open_listenfd(argv[1]);
    while (1) {
        clientlen = sizeof(struct sockaddr_storage); /* Important! */
        connfd = Accept(listenfd, (SA *) &clientaddr, &clientlen);
        Getnameinfo((SA *) &clientaddr, clientlen,
                    client_hostname, MAXLINE, client_port, MAXLINE, 0);
        printf("Connected to (%s, %s)\n", client_hostname, client_port);
        echo(connfd);
        Close(connfd);
    }
    exit(0);
}
```

