CSE320 System Fundamentals II
Network Programming

YoungMin Kwon

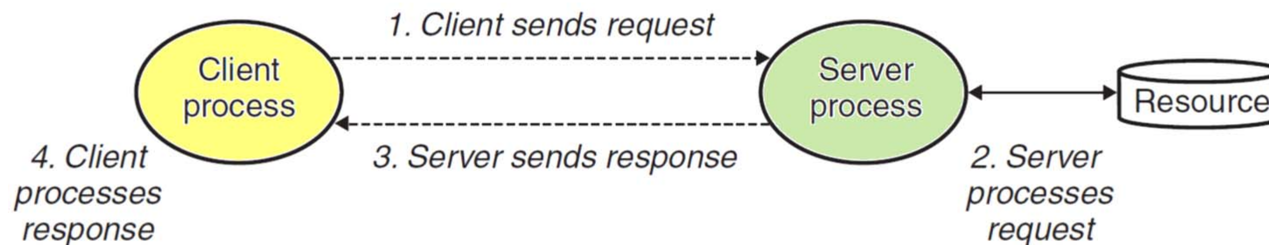
SUNY Korea
The State University of New York

Client-Server Model

- Client-Server model
  - Application: a server process and one or more client processes
    - Clients and servers are not machines (hosts) but processes
  - Server process
    - Manages resources and provide services to clients
  - Example
    - Web server, FTP server, email server

Client-Server Model

- Transaction

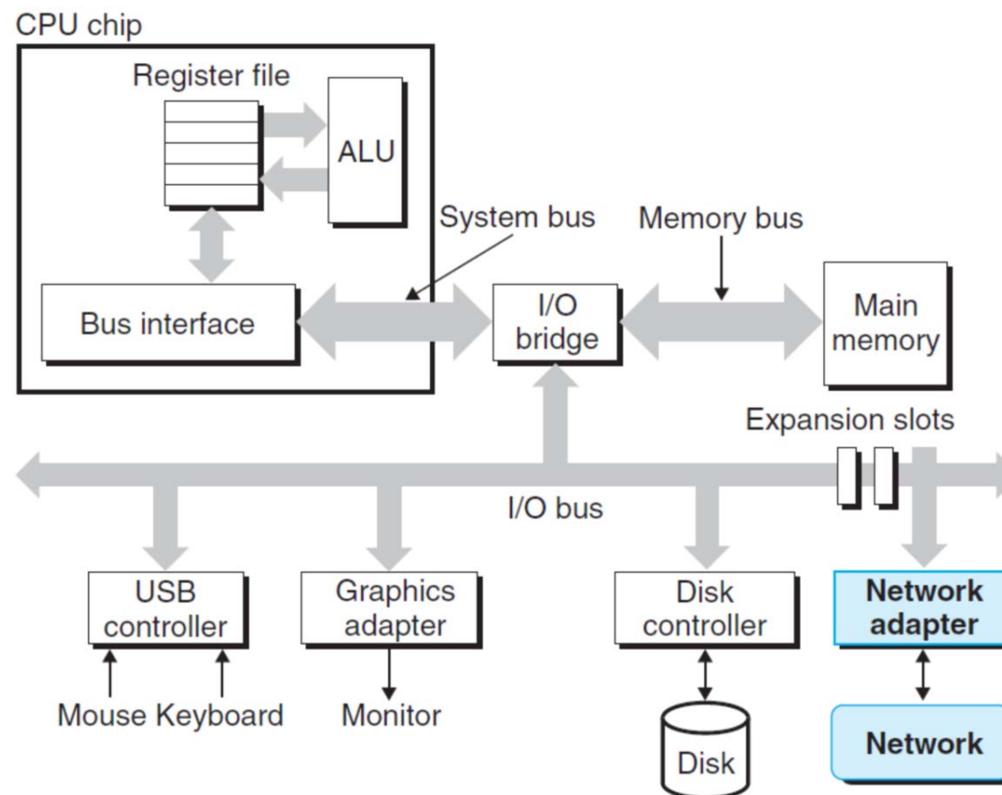


  - Client initiates a transaction by sending a request
  - Server interprets the request and manipulate its resources
  - Server sends a response
  - Client processes the response

SUNY Korea

Networks

- Network
  - To a host, a network is just another I/O device that serves as a source and sink for data

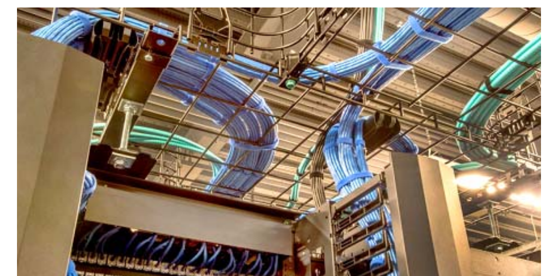


SUNY Korea

Networks

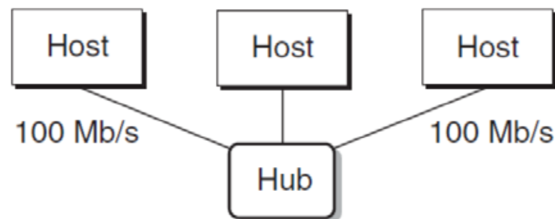
- Adapter
  - Physical interface to the network
  - Copy data between host and network

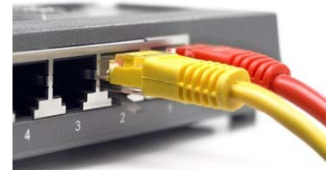
- Local Area Network (LAN)
  - Spans a building or a campus
  - Ethernet is the most popular LAN



Networks (LAN: Ethernet)

- Ethernet segment
  - Consists of wires (twisted pairs of wires) and a hub
  - Spans a room or a floor in a building



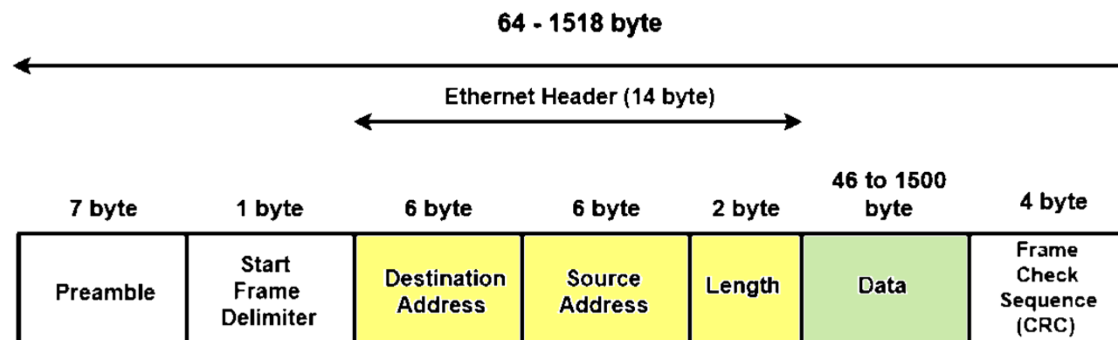


  - Wires
    - Have the same maximum bit bandwidth: 100 Mb/s, 1 Gb/s
    - Connects an adapter and a port on the hub
  - Hub
    - Copies every bit it receives from a port to other ports

SUNY Korea

Networks (LAN: Ethernet)

- Ethernet segment
  - Each Ethernet adapter has a MAC address, a globally unique 48-bit address
  - A host can send a frame to any other host on the same segment
  - Frame
    - Header: source/destination, frame length
    - Payload: data bits

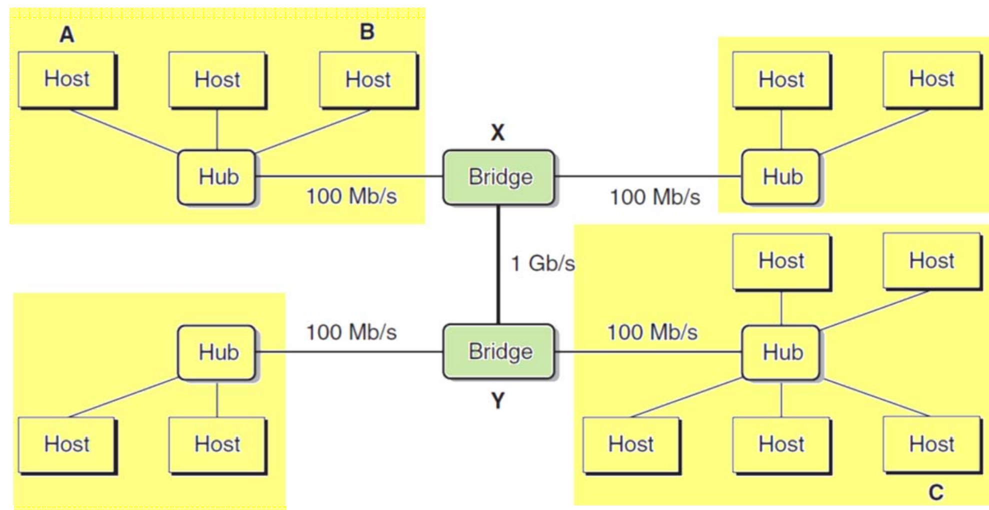


IEEE 802.3 Ethernet Frame Format

SUNY Korea

Network (LAN: Ethernet)

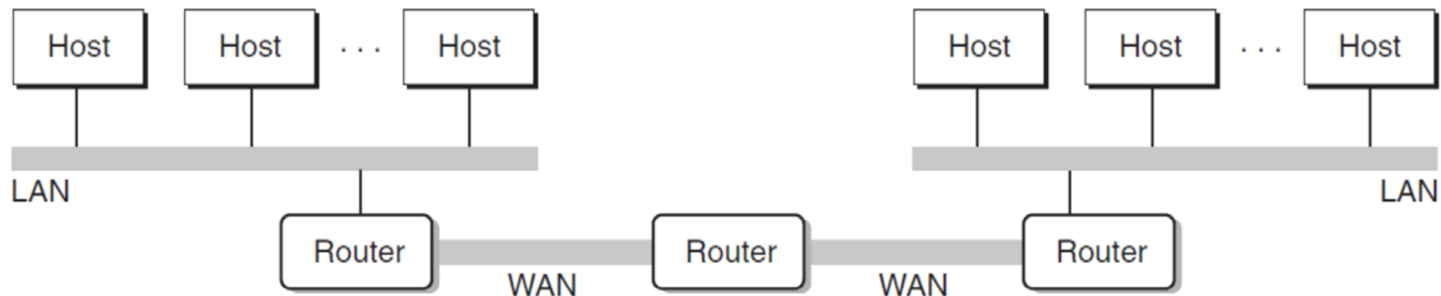
- Bridge
  - Connects multiple **Ethernet segments**
  - Knows which hosts are reachable from which port
  - Selectively copy frames from a port to another when necessary
  - The bandwidth of the wires can be different



SUNY Korea

Network (internet)

- internet (with small i)
  - Routers
    - Connect multiple **incompatible** LANs
    - Connect WANs (Wide Area Networks, e.g. point-to-point phone connections)



SUNY Korea

Network (internet)

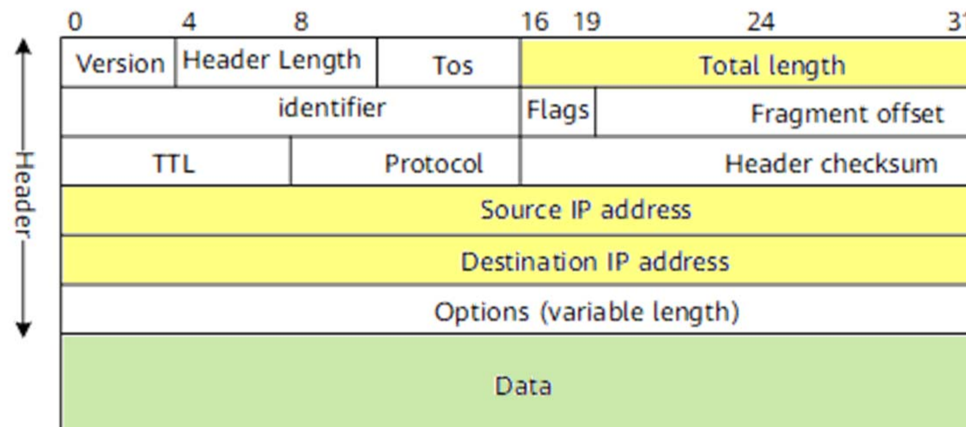
- Protocol software (runs on hosts and routers)
  - How a source host can send data to a destination host across incompatible networks?

- Naming scheme
  - Different LAN technologies have different ways of assigning addresses to hosts
  - internet protocol defines a unique format for host addresses

SUNY Korea

Network (internet)

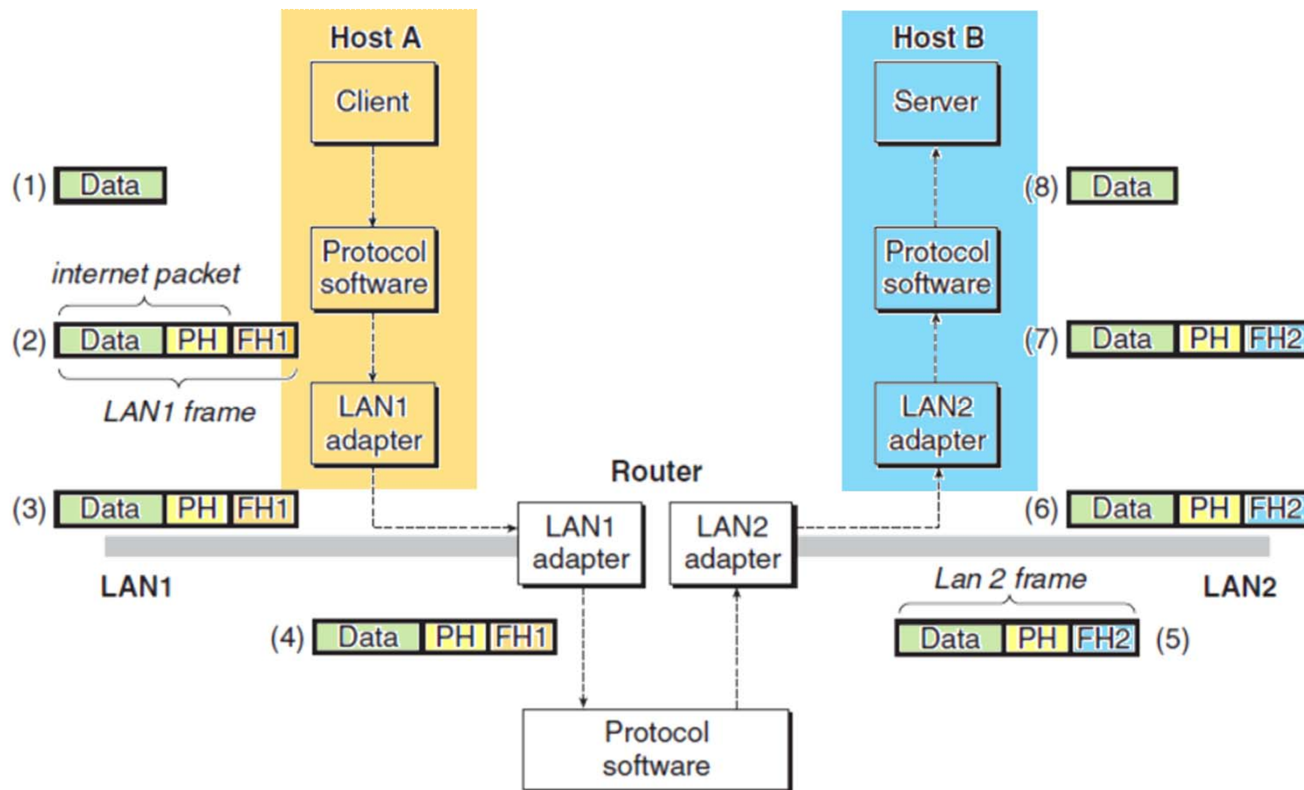
- Delivery mechanism
  - Packet: uniform way to bundle up data bits into discrete chunks
    - Header: source/destination address, packet size
    - Payload: data
  - E.g. IP packet:



SUNY Korea

Network (internet)

- How data travel from one host to another on an internet
  - PH: internet packet header
  - FH1: frame header for LAN1
  - FH2: frame header for LAN2

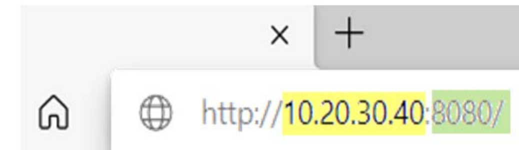


SUNY Korea

IP Internet

- Global IP Internet (with capital I)
  - Most famous implementation of an internet

- Protocols
  - Between devices
    - IP (Internet Protocol): naming scheme and delivery mechanism for packets (called datagram)

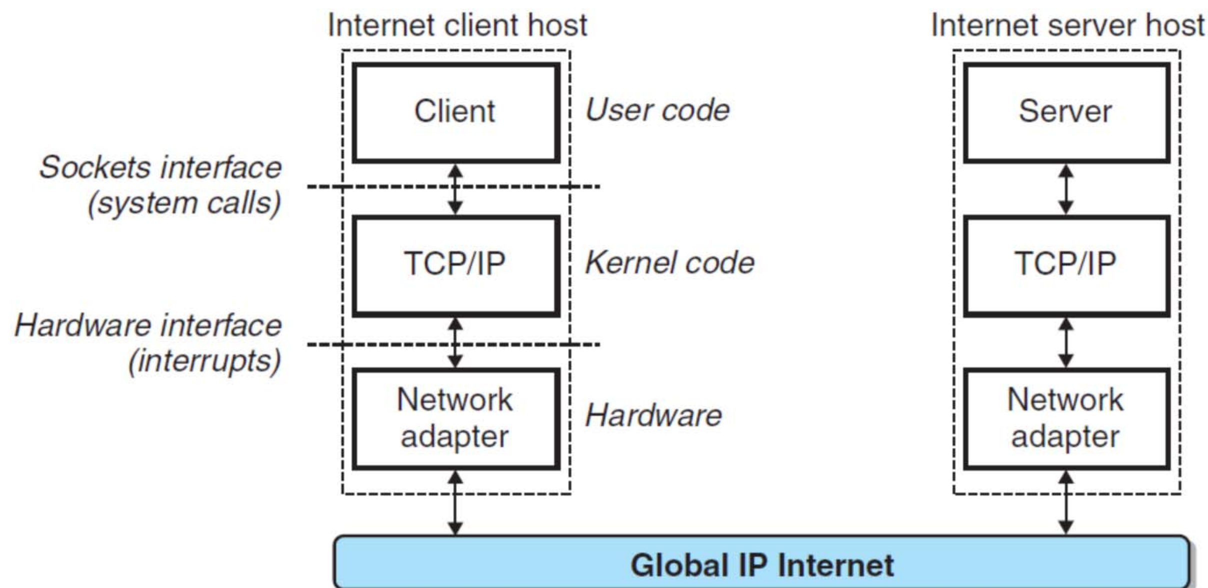


  - Between processes (port # to identify services)
    - TCP (Transmission Control Protocol): reliable bidirectional connection (streaming)
    - UDP (Unreliable Datagram Protocol): packets can be lost or duplicated

SUNY Korea

IP Internet

- Internet clients and servers communicate using
  - Socket interface functions
  - Unix I/O functions



Hardware and software organization of an Internet application

SUNY Korea

IP Internet

- IP Addresses
  - IP address is an unsigned 32-bit integer
  - Presented as a dotted-decimal notation

```
//IP address structure
struct in_addr {
    uint32_t s_addr;    //Address in network byte order (big-endian)
};

#include <arpa/inet.h>

//Return values in network byte order (big-endian)
uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);

//Return values in host byte order (can be big-endian or little-endian)
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

SUNY Korea

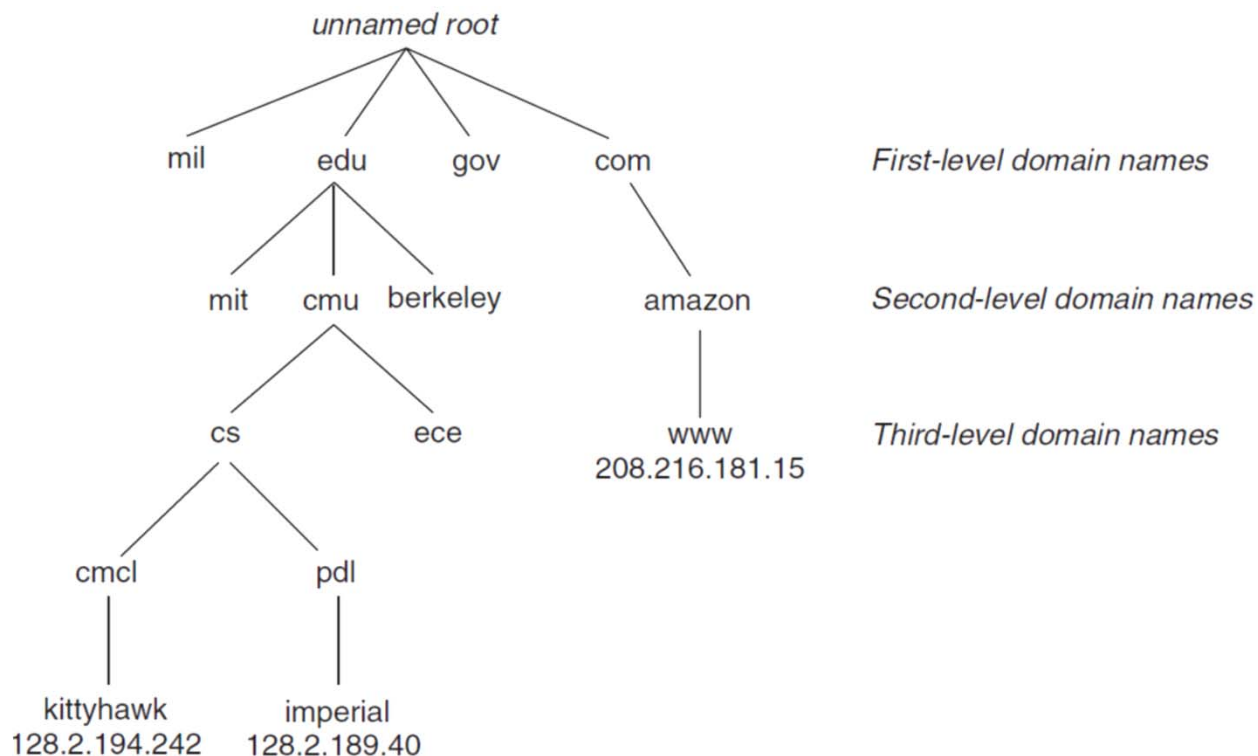
```c
#include <stdio.h>
#include <arpa/inet.h>
int main() {
    uint32_t hl = 0x12345678;
    uint16_t hs = 0x1234;
    uint32_t nl = htonl(hl);
    uint32_t ns = htons(hs);
    unsigned char *p;

    p = (unsigned char*) &hl;
    printf("hl: %x %x %x %x\n", p[0], p[1], p[2], p[3]);

    p = (unsigned char*) &nl;
    printf("nl: %x %x %x %x\n", p[0], p[1], p[2], p[3]);

    p = (unsigned char*) &hs;
    printf("hs: %x %x\n",       p[0], p[1]);

    p = (unsigned char*) &ns;
    printf("ns: %x %x\n",       p[0], p[1]);
}
```

```
hl: 78 56 34 12
nl: 12 34 56 78
hs: 34 12
ns: 12 34
```

SUNY Korea

IP Internet

- Internet Domain Names
  - Human friendly names instead of large integers
  - Mechanism to map domain names to IP addresses
    - Set of domain names forms a hierarchy



SUNY Korea

IP Internet

- Internet Domain Names
  - Multiple IP addresses can be mapped to the same domain name ⇒ load balancing

```
$ nslookup www.twitter.com
...
Name:    twitter.com
Address: 104.244.42.193
Name:    twitter.com
Address: 104.244.42.1

$ nslookup twitter.com
...
Name:    twitter.com
Address: 104.244.42.1
Name:    twitter.com
Address: 104.244.42.193
```

SUNY Korea

IP Internet

- TCP/IP connection
  - Point-to-point: it connects a pair of processes
  - Full-duplex: data can flow in both directions
  - Reliable: stream of data sent will be received in the same order
- Socket
  - An end point of a connection
  - Socket address: IP address : port number
    - e.g. 128.2.194.242:80

SUNY Korea

IP Internet

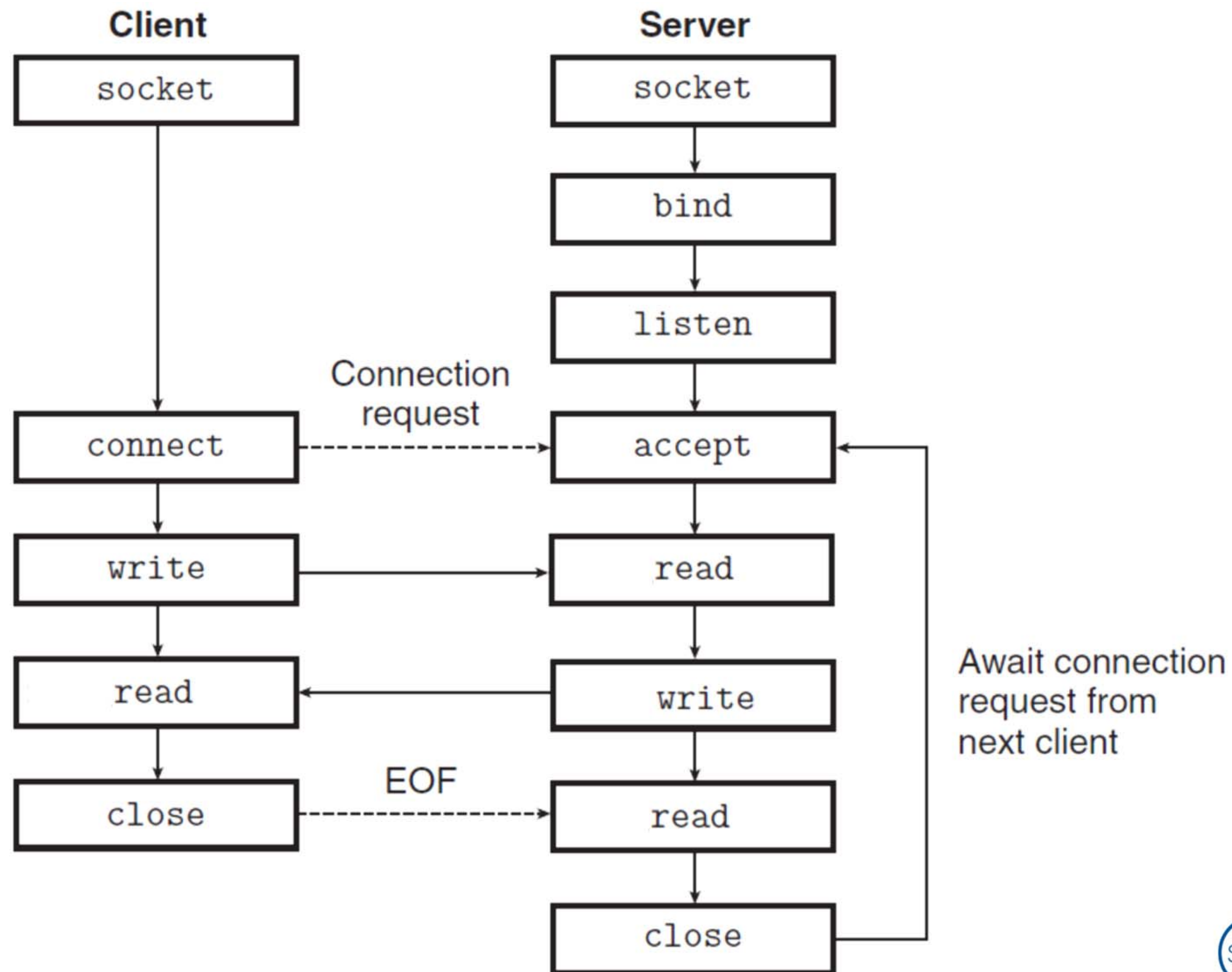
- Port number
  - A 16-bit integer
  - Ephemeral port
    - Assigned by the kernel
    - e.g. client's socket
  - Well-known port
    - Server's socket address
    - e.g. webservers 80, email servers 25, …

```
$ cat /etc/services
...
ftp             21/tcp
ssh             22/tcp                          # SSH Remote Login Protocol
ssh             22/udp
telnet          23/tcp
smtp            25/tcp          mail
...
http            80/tcp          www             # WorldWideWeb HTTP
```

SUNY Korea

Sockets Interface

- Overview of network applications based on the sockets interface



SUNY Korea

Sockets Interface

- Socket address structures

```
//Generic socket address structure (for connect, bind, accept)
struct sockaddr {
    uint16_t sa_family;     //Protocol family
    char     sa_data[14];   //Address data
};

//IP socket address structure
struct sockaddr_in {
    uint16_t        sin_family; //Protocol family (always AF_INET)
    uint16_t        sin_port;   //Port number in network byte order
    struct in_addr  sin_addr;   //IP address in network byte order
    unsigned char   sin_zero[8];//Pad to sizeof(struct sockaddr)
};
```

SUNY Korea

Sockets Interface

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);

- Clients and servers use the socket function
  to create a socket descriptor

int fd = socekt(AF_INET, SOCK_STREAM, 0);
- fd is only partially opened and cannot be used for read/write yet
```

```
int connect(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

- A client establishes a connection with a server by calling connect
- connect function blocks until a connection is established
- If successful sockfd is ready for reading and writing
```

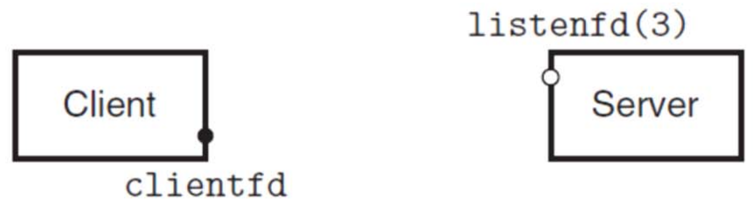
SUNY Korea

Sockets Interface

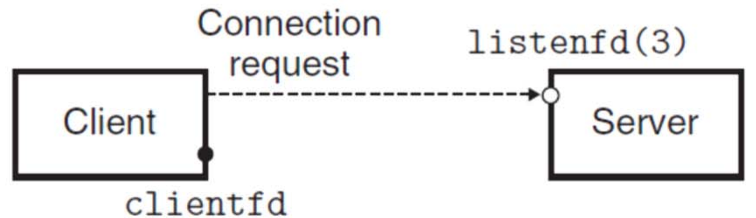
```
int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

- Asks the kernel to associate the server’s socket address with sockfd


int listen(int sockfd, int backlog);

- Converts sockfd a listening socket that can accept connection requests
  from clients
- backlog: the # of connection requests the kernel should queue
  (typically set to a large value like 1024)


int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

-  Servers wait for connection request from clients by calling accept
-  Fills the client’s socket address in addr
-  Returns a connected descriptor that can be used for Unix I/O functions
```

SUNY Korea

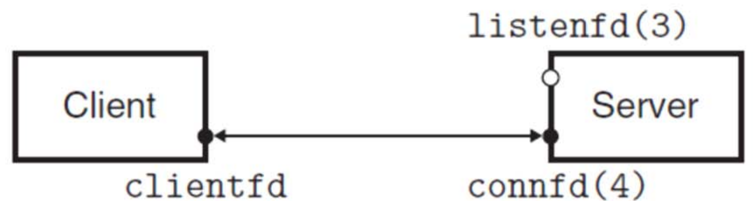
Sockets Interface



1. Server blocks in `accept`*, waiting for connection request on listening descriptor* `listenfd`*.*



2. Client makes connection request by calling and blocking in `connect`*.*



3. Server returns `connfd` *from* `accept`*. Client returns from* `connect`*. Connection is now established between* `clientfd` *and* `connfd`*.*

SUNY Korea

Echo Server Example

```c
//echo_server.c
//
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#include "common.h"
#include "util.h"

#define SERV_PORT   8000

void echo(int fd) {
    while(1) {
        char line[MAX_LINE];
        int n = read_line(fd, line, MAX_LINE);
        ON_FALSE_EXIT(n >= 0, "readline error");

        if(n == 0)  //connection terminated
            return;
        else
            ON_FALSE_EXIT(writen(fd, line, n) == n, "writen error");
    }
}
```

SUNY Korea

```c
int main(int argc, char **argv) {
    int sfd;
    struct sockaddr_in saddr;

    //prepare sadder with INADDR_ANY and SERV_PORT
    memset(&saddr, 0, sizeof(saddr));
    saddr.sin_family = AF_INET;
    saddr.sin_addr.s_addr = htonl(INADDR_ANY);
    saddr.sin_port = htons(SERV_PORT);

    ON_FALSE_EXIT((sfd = socket(AF_INET, SOCK_STREAM, 0)) >= 0,
                  "cannot open stream socket");
    ON_FALSE_EXIT(bind(sfd, (struct sockaddr*)&saddr, sizeof(saddr)) >= 0,
                  "cannot bind local address");
    ON_FALSE_EXIT(listen(sfd, 1024/*backlog*/) >= 0,
                  "listen failed");

    while(1) {
        struct sockaddr_in caddr;
        int cfd, clen = sizeof(caddr);
        ON_FALSE_EXIT((cfd = accept(sfd, (struct sockaddr*) &caddr, &clen)) >= 0,
                      "accept error");
…
```

SUNY Korea

…

```
        if(fork() == 0) {   //child
            close(sfd);
            printf("pid: %d, client: %s:%d\n", getpid(),
                                                inet_ntoa(caddr.sin_addr),
                                                caddr.sin_port);
            echo(cfd);

            close(cfd);
            printf("pid: %d done\n", getpid());
            exit(0);
        }
        else {
            close(cfd);
        }
    }
    close(sfd);
    return 0;
}
```

SUNY Korea

```c
//echo_client.c
//
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#include "common.h"
#include "util.h"

#define SERV_ADDR   "127.0.0.1"
#define SERV_PORT   8000

void copy(int sfd) {
    char sline[MAX_LINE], rline[MAX_LINE];
    while(fgets(sline, MAX_LINE, stdin) != NULL) {
        int n = strlen(sline);
        ON_FALSE_EXIT( writen(sfd, sline, n) == n, "writen error" );

        ON_FALSE_EXIT( (n = read_line(sfd, rline, MAX_LINE)) >= 0, "readline error" );
        rline[n] = 0;
        fputs(rline, stdout);
    }
    ON_FALSE_EXIT( ferror(stdin) == 0, "cannot read file" );
}
```

SUNY Korea
The State University of New York

```c
int main(int argc, char **argv) {
    int sfd;
    struct sockaddr_in saddr;

    memset(&saddr, 0, sizeof(saddr));
    saddr.sin_family = AF_INET;
    saddr.sin_addr.s_addr = inet_addr(SERV_ADDR);
    saddr.sin_port = htons(SERV_PORT);

    ON_FALSE_EXIT((sfd = socket(AF_INET, SOCK_STREAM, 0)) >= 0,
                  "cannot open stream socket");
    ON_FALSE_EXIT(connect(sfd, (struct sockaddr*)&saddr, sizeof(saddr)) >= 0,
                  "cannot connect to server");

    printf("server: %s:%d\n", inet_ntoa(saddr.sin_addr), saddr.sin_port);

    copy(sfd);

    close(sfd);
    return 0;
}
```

Programming Assignment 9

- Implement a remote shell server and client
  - Server:
    - Wait for a connection from a client
    - Execute shell (from the previous assignment) with its input/output redirected to the client socket
  - Client:
    - Connect to the server
    - Copy input in stdin to the socket
    - Copy input in the socket to stdout
- Download shell_remote.zip and implement all TODOs
- Due date: TBD

SUNY Korea

```c
//
//shell_server.c
//

int main(int argc, char **argv) {
    int sfd;
    struct sockaddr_in saddr;

    //TODO: prepare sadder with INADDR_ANY and SERV_PORT

    //TODO: make a server socket sfd

    int enable = 1;
    ON_FALSE_EXIT(setsockopt(sfd, SOL_SOCKET, SO_REUSEADDR, &enable,
                             sizeof(int)) >= 0, "cannot set socket option");

    //TODO: bind saddr to the server socket

    //TODO: listen to the server socket with the backlog size of 1024
…
```

```
    while(1) {
        struct sockaddr_in caddr;
        int cfd, clen = sizeof(caddr);

        //TODO: accept a connection to the server socket

        printf("[request] from client: %s:%d\n", inet_ntoa(caddr.sin_addr),
                                                  caddr.sin_port);

        if(fork() == 0/*child*/) {
            //TODO: dupe cfd to the stdin (0), the stdout (1) and the stderr (2)

            //TODO: close cfd and sfd

            //TODO: execute shell (from the previous assignment)
            //hint: use execvp

        }
        else {
            //TODO: close cfd

        }
    }
    return 0;
}
```

SUNY Korea

```
//
//shell_client.c
//
void repeat(int sfd) {
    char buf[MAX_LINE];
    int n;
    fd_set fds;

    while(1) {
        //copy input from stdin to sfd and from sfd to stdout

        //TODO: make fds the empty set
        //hint: use FD_ZERO

        //TODO: add 0 and sfd to fds
        //hint: use FD_SET

        //TODO: wait for input in fds
        //hint: use select

        //TODO: if there are input in stdin, send it to sfd
        //hint: use FD_ISSET, read, and writen

        //TODO: if there are input in sfd, send it to stdout
        //hint: use FD_ISSET, read, and writen

    }
}
```

SUNY Korea

```c
int main(int argc, char **argv) {
    int sfd;
    struct sockaddr_in saddr;

    //TODO: prepare sadder with SERV_ADDR and SERV_PORT

    //TODO: make a socket sfd

    //TODO: connect to the server

    printf("server: %s:%d\n", inet_ntoa(saddr.sin_addr), saddr.sin_port);

    repeat(sfd);

    //TODO: close the socket

    return 0;
}
```