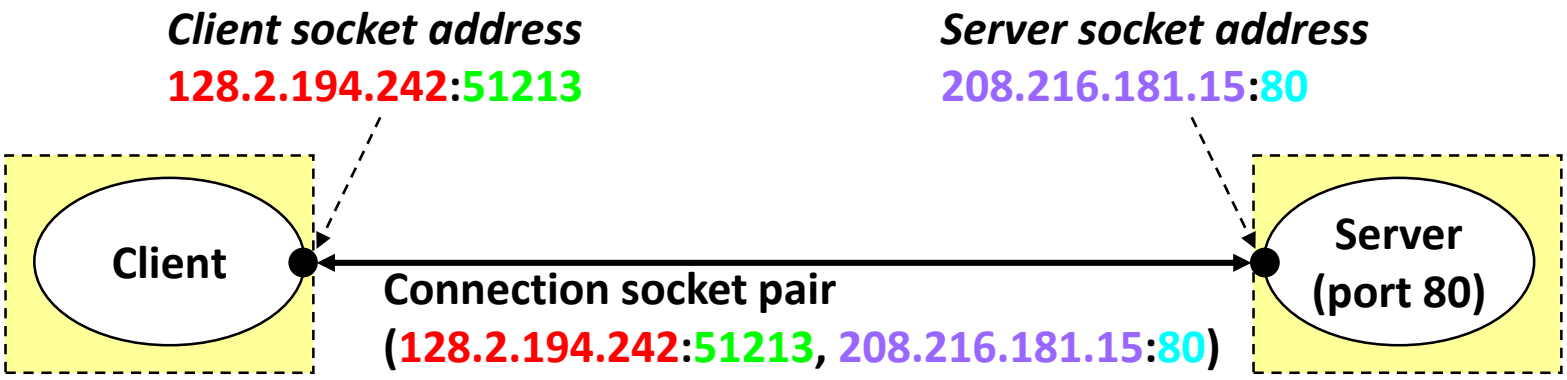
Sockets

Dong-kun Shin
Embedded Software Laboratory
Sungkyunkwan University
http://nyx.skku.ac.kr

Internet Connections (1)

- **Connection**
  - Clients and servers communicate by sending streams of bytes over connections:
    - Point-to-point, full-duplex, and reliable.
  - A socket is an endpoint of a connection
    - Socket address is an <IP address : port> pair
  - A port is a 16-bit integer that identifies a process
    - Ephemeral port: assigned automatically on client when client makes a connection request
    - Well-known port: associated with some service provided by a server (e.g. port 80 is associated with web servers.)
  - A connection is uniquely identified by the socket addresses of its endpoints (socket pair)
    - <client IP:client port, server IP:server port>

Internet Connections (2)

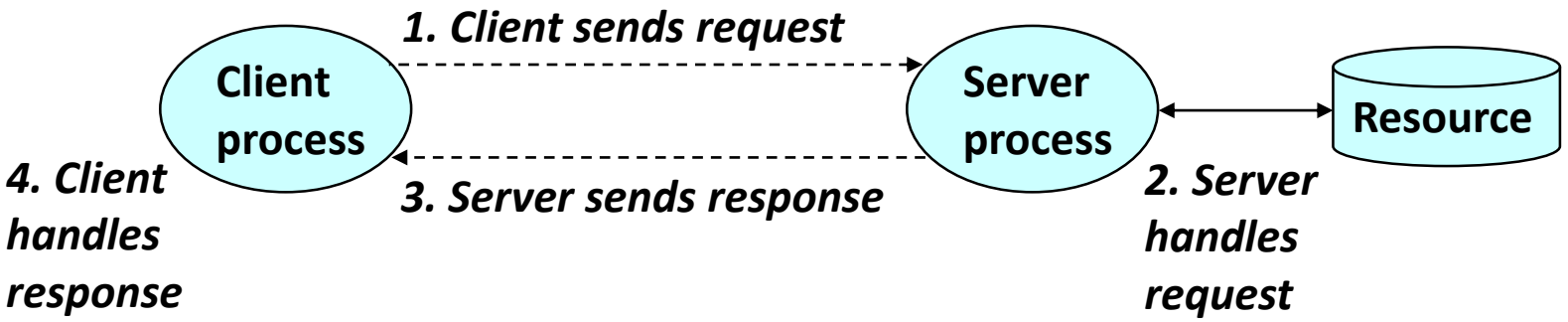


Note: 51213 is an ephemeral port allocated by the kernel

Note: 80 is a well-known port associated with Web servers

Client-Server Model

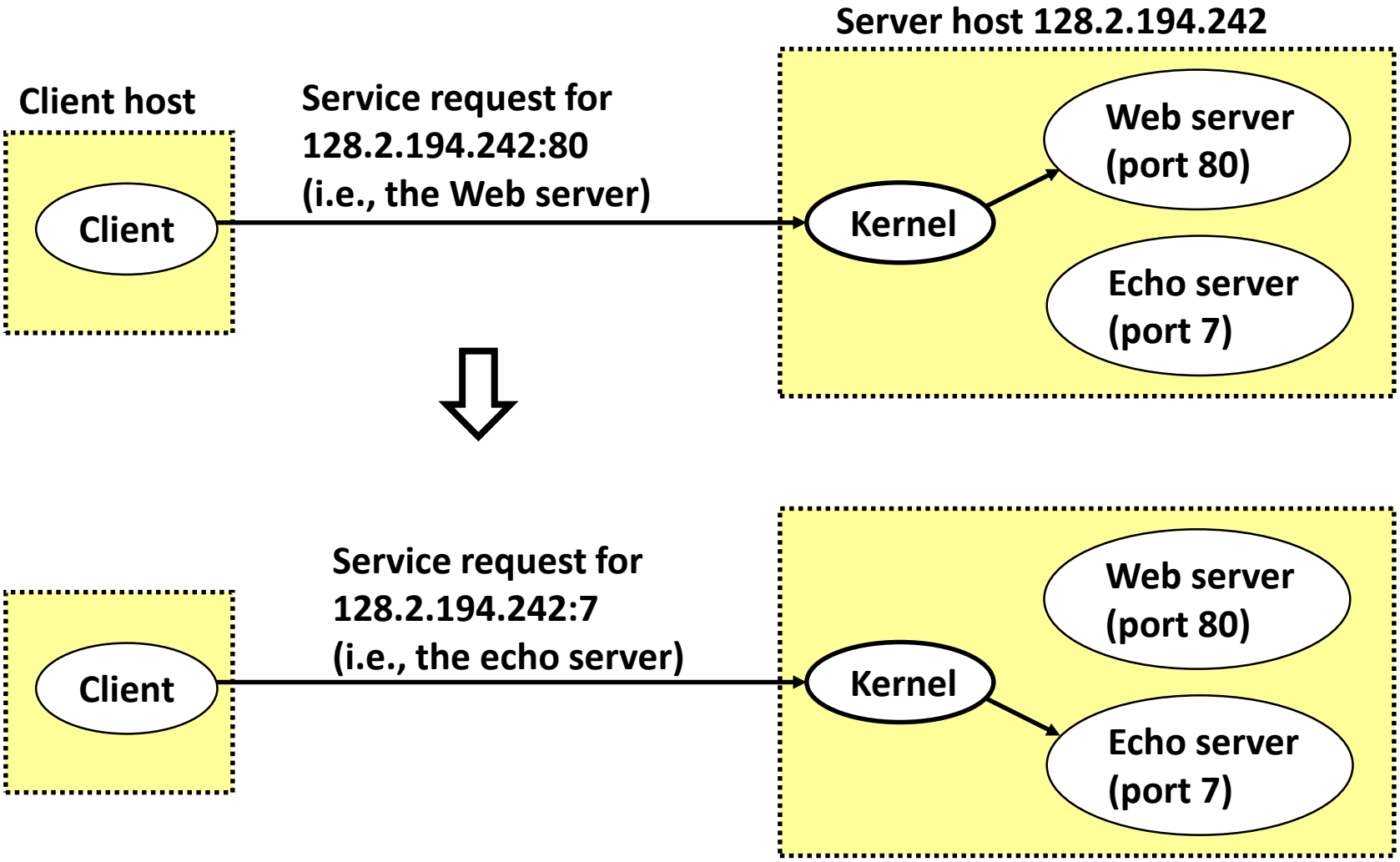
- **Most network application is based on the client-server model:**
  - A server process and one or more client processes
    - Clients and servers are processes running on hosts (can be the same or different hosts)
  - Server manages some resource
  - Server provides service by manipulating resource for clients



Clients

- **Examples of client programs**
  - Web browsers, ftp, telnet, ssh
- **How does a client find the server?**
  - The IP address in the server socket address identifies the host.
  - The (well-known) port in the server socket address identifies the service, and thus implicitly identifies the server process that performs that service
  - Examples of well-known ports (cf. **/etc/services**)
    - Port 21: ftp
    - Port 23: telnet
    - Port 25: mail
    - Port 80: web

Using Ports


Server host 128.2.194.242
Client host
Service request for
128.2.194.242:80
(i.e., the Web server)
Client
Kernel
Web server
(port 80)
Echo server
(port 7)
Service request for
128.2.194.242:7
(i.e., the echo server)
Client
Kernel
Web server
(port 80)
Echo server
(port 7)

Servers

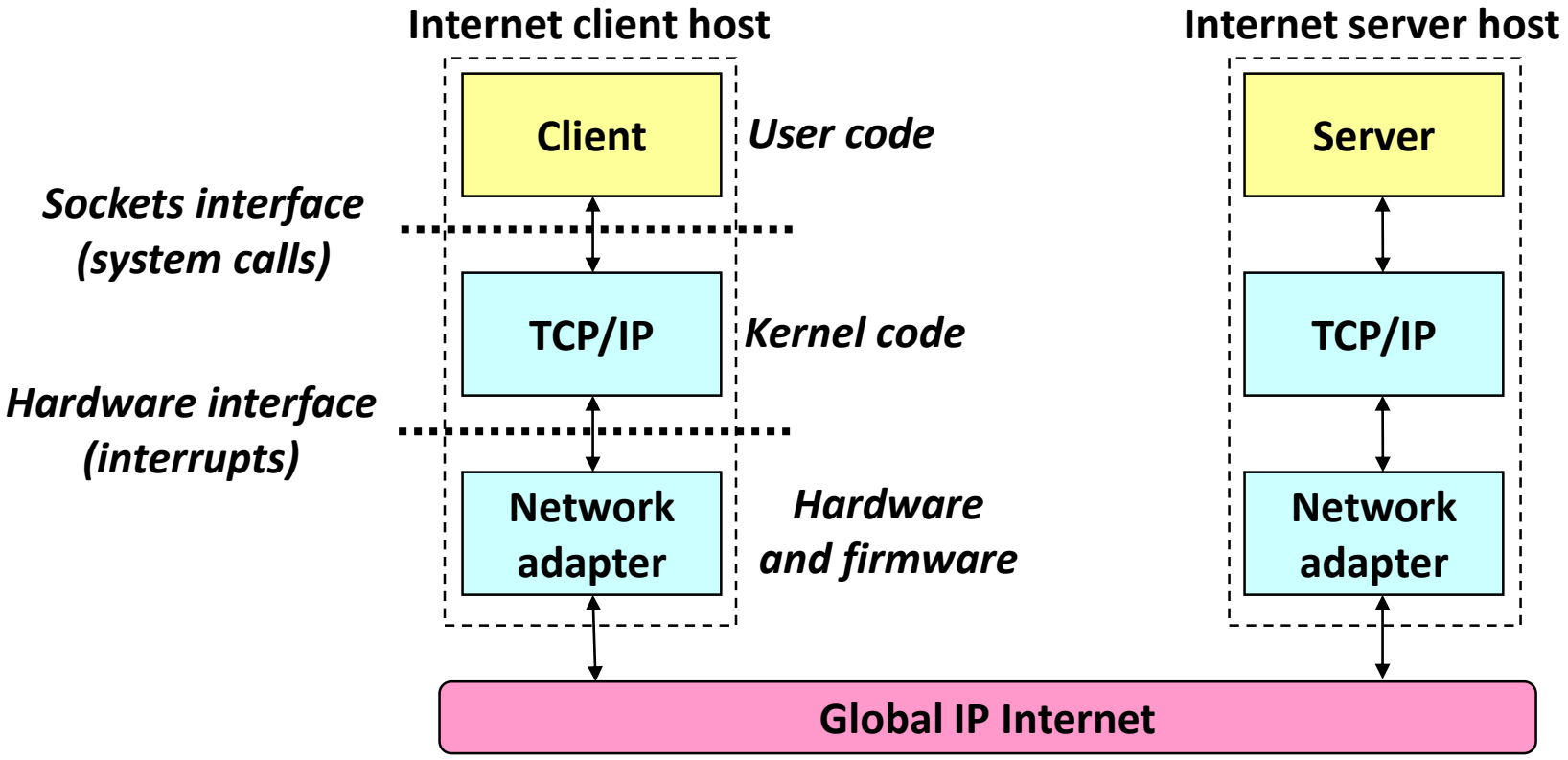
- **Servers are long-running processes (daemons)**
  - Created at boot-time (typically) by the init process (process 1)
  - Run continuously until the machine is turned off.
- **Each server waits for requests to arrive on a well-known port associated with a particular service**
  - Port 21: ftp server
  - Port 23: telnet server
  - Port 25: mail server
  - Port 80: HTTP server
- **A machine that runs a server process is also often referred to as a "server"**

Sockets

- **What is a socket?**
  - A host-local, application-created/owned, OS-controlled interface to network (a "door")
    - To the kernel, a socket is an endpoint of communication.
    - To an application, a socket is a file descriptor.
      - Applications read/write from/to the network using the file descriptor.
      - Remember: All Unix I/O devices, including networks, are modeled as files.
  - Clients and servers communicate with each by reading from and writing to socket descriptors.
    - The main distinction between regular file I/O and socket I/O is how the application "opens" the socket descriptors.

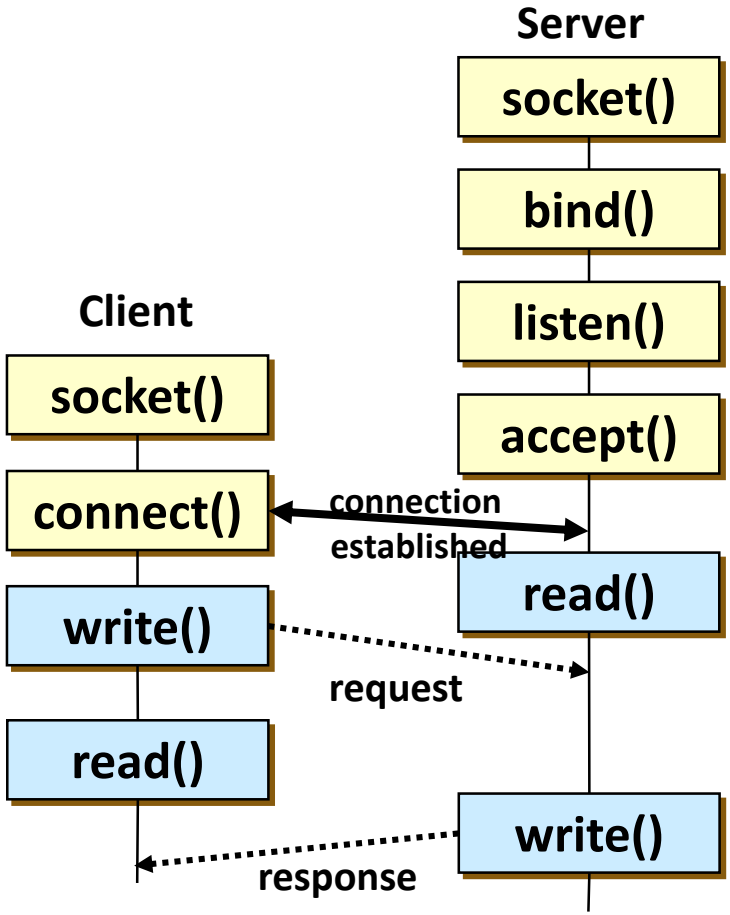
Sockets

- Hardware/Software organization of an Internet application

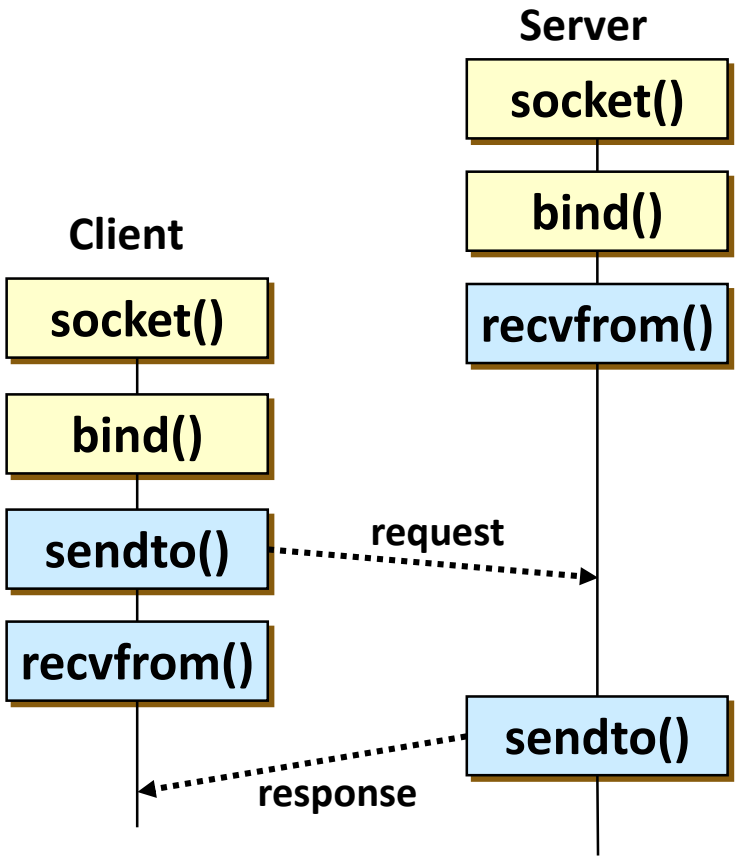


Sockets

Connection-oriented service



Connectionless service



Socket Address Structure

- **Generic socket address**
  - For address arguments to **connect(), bind(),** and **accept()**

```
struct sockaddr {
  unsigned short  sa_family;    /* protocol family */
  char            sa_data[14];  /* address data.  */
};
```

- **Internet-specific socket address**
  - Must cast **(sockaddr_in *)** to **(sockaddr *)** for **connect(), bind(),** and **accept()**

```
struct sockaddr_in  {
  unsigned short  sin_family;  /* address family (always AF_INET) */
  unsigned short  sin_port;    /* port num in network byte order */
  struct in_addr  sin_addr;    /* IP addr in network byte order */
  unsigned char   sin_zero[8]; /* pad to sizeof(struct sockaddr) */
};
```

socket()

- **int socket (int family, int type, int protocol)**
  - **socket()** creates a socket descriptor.
  - **family** specifies the protocol family.
    - **AF_UNIX**: Local Unix domain protocols
    - **AF_INET**: IPv4 Internet protocols
  - **type** specifies the communication semantics.
    - **SOCK_STREAM**: provides sequenced, reliable, two-way, connection-based byte streams
    - **SOCK_DGRAM**: supports datagrams (connectionless, unreliable messages of a fixed maximum length)
  - **protocol** specifies a particular protocol to be used with the socket.

connect()

- **int connect (int sockfd, const struct sockaddr *servaddr, socklen_t addrlen)**
  - Used by a TCP client to establish a connection with a TCP server.
  - **servaddr** contains <IP address, port number> of the server.
  - The client does not have to call **bind()** before calling **connect().**
    - The kernel will choose both an ephemeral port and the source IP address if necessary.
  - Client process suspends (blocks) until the connection is created.

Echo Client (1)


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

#define MAXLINE 80

int main (int argc, char *argv[]) {
   int n, cfd;
   struct hostent *h;
   struct sockaddr_in saddr;
   char buf[MAXLINE];
   char *host = argv[1];
   int port = atoi(argv[2]);

   if ((cfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
      printf("socket() failed.\n");
      exit(1);
   }
```

Echo Client (2)


```
    if ((h = gethostbyname(host)) == NULL) {
        printf("invalid hostname %s\n", host);
        exit(2);
    }
    bzero((char *)&saddr, sizeof(saddr));
    saddr.sin_family = AF_INET;
    bcopy((char *)h->h_addr, (char *)&saddr.sin_addr.s_addr, h->h_length);
    saddr.sin_port = htons(port);

    if (connect(cfd,(struct sockaddr *)&saddr,sizeof(saddr)) < 0) {
        printf("connect() failed.\n");
        exit(3);
    }
    while ((n = read(0, buf, MAXLINE)) > 0) {
        write(cfd, buf, n);
        n = read(cfd, buf, MAXLINE);
        write(1, buf, n);
    }
    close(cfd);
}
```

Exercise