SSE2033

SUNG KYUN KWAN UNIVERSITY

Fall 2019

# System Software Experiment 2

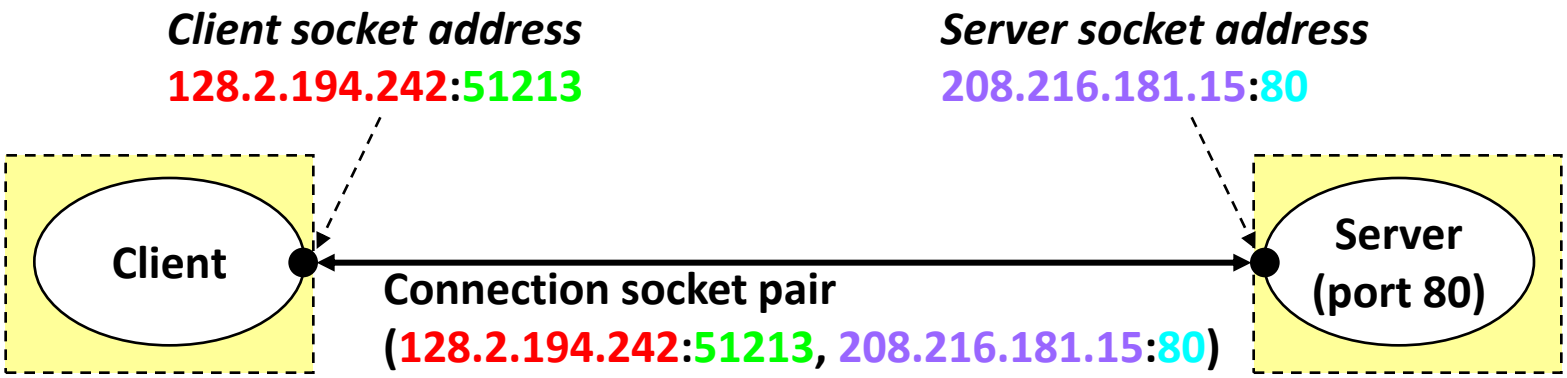
# Sockets

Prof. Jinkyu Jeong (*jinkyu@skku.edu*)
TA – Gyusun Lee (*gyusun.lee@csi.skku.edu*)
TA – Jiwon Woo (*jiwon.woo@csi.skku.edu*)
Computer Systems and Intelligence Laboratory (*http://csi.skku.edu*)
Sungkyunkwan University

# 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)




**Client host address**
**128.2.194.242**

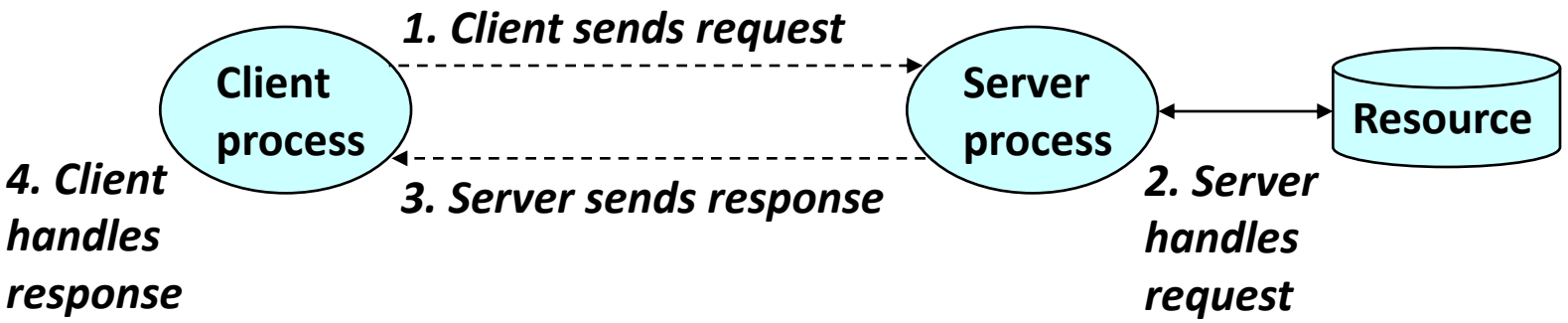
**Server host address**
**208.216.181.15**

***Note: 51213 is an ephemeral port allocated by the kernel***

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

# Client-Server Model

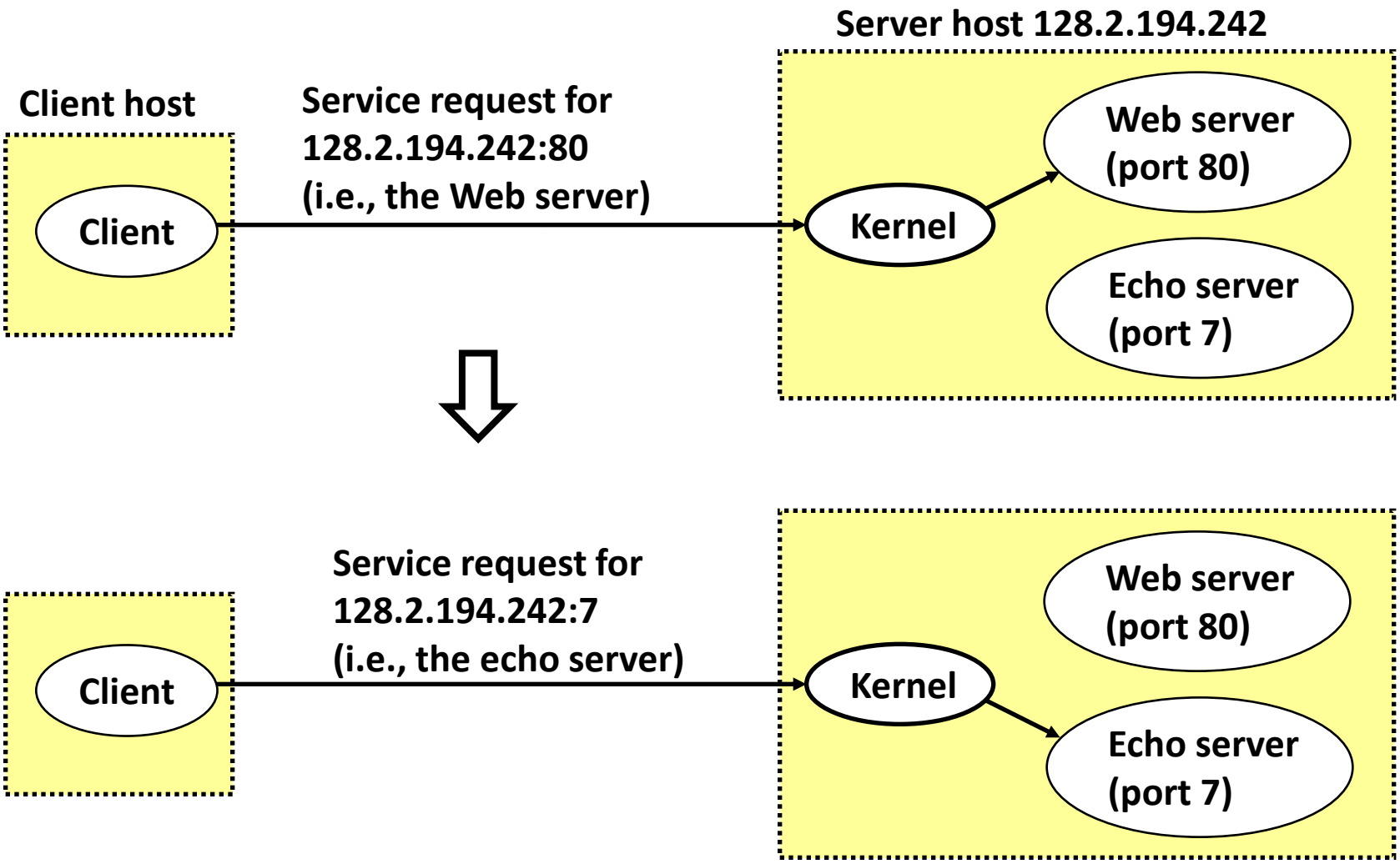
- Most network application is based on the client-server model:
  - A server process and one or more client processes
    - Clients and servers are processes running on hosts (can be the same or different hosts)
  - Server manages some resource
  - Server provides service by manipulating resource for 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 (1)

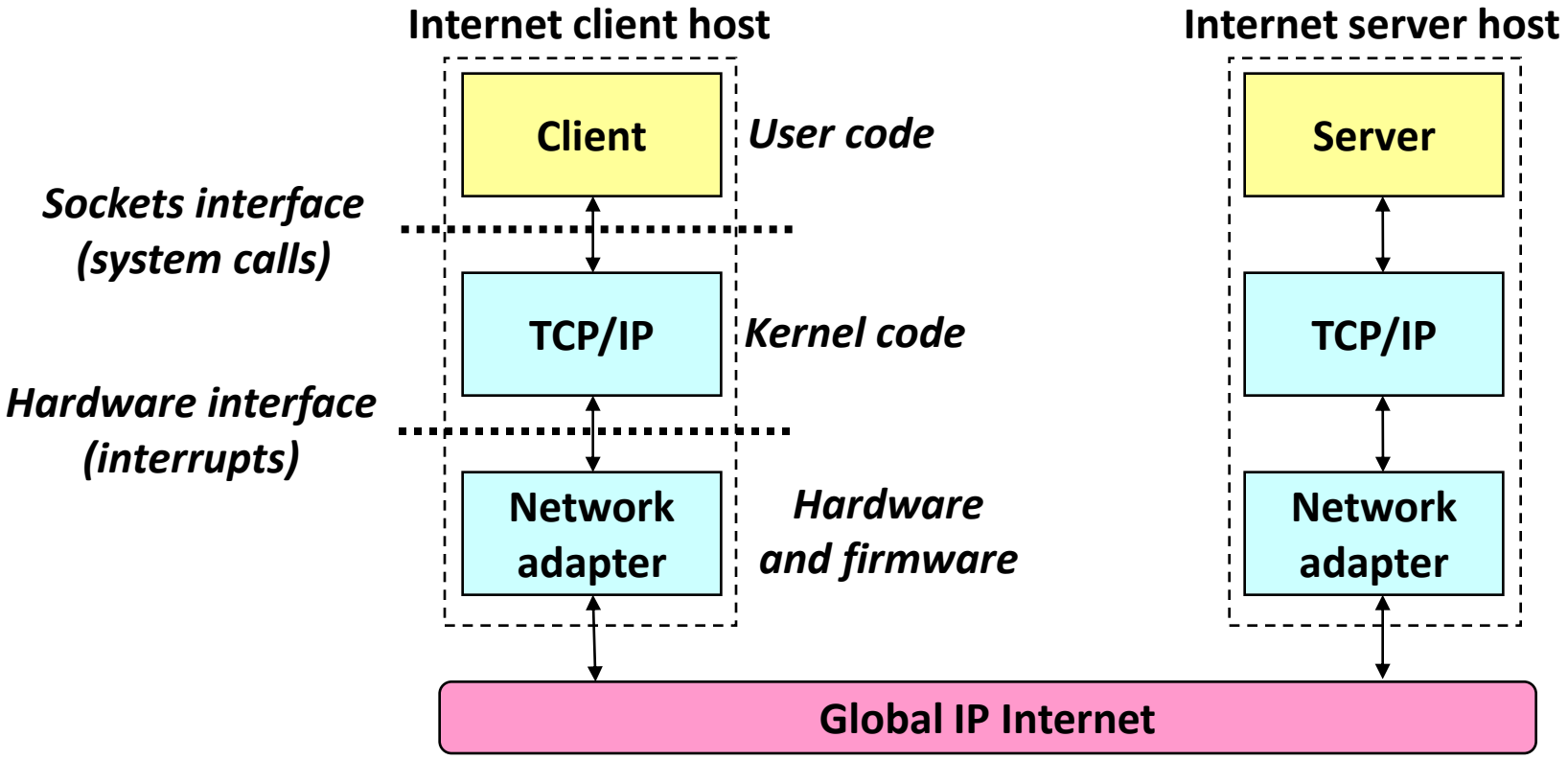
- Sockets interface
  - Introduced in BSD4.1 UNIX, 1981.
  - Provides a user-level interface to the network.
  - Explicitly created, used, released by applications.
  - Based on client/server paradigm
  - Two types of transport service
    - Unreliable datagram
    - Reliable, connection-oriented byte stream
  - Underlying basis for all Internet applications

# Sockets (2)

- 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.

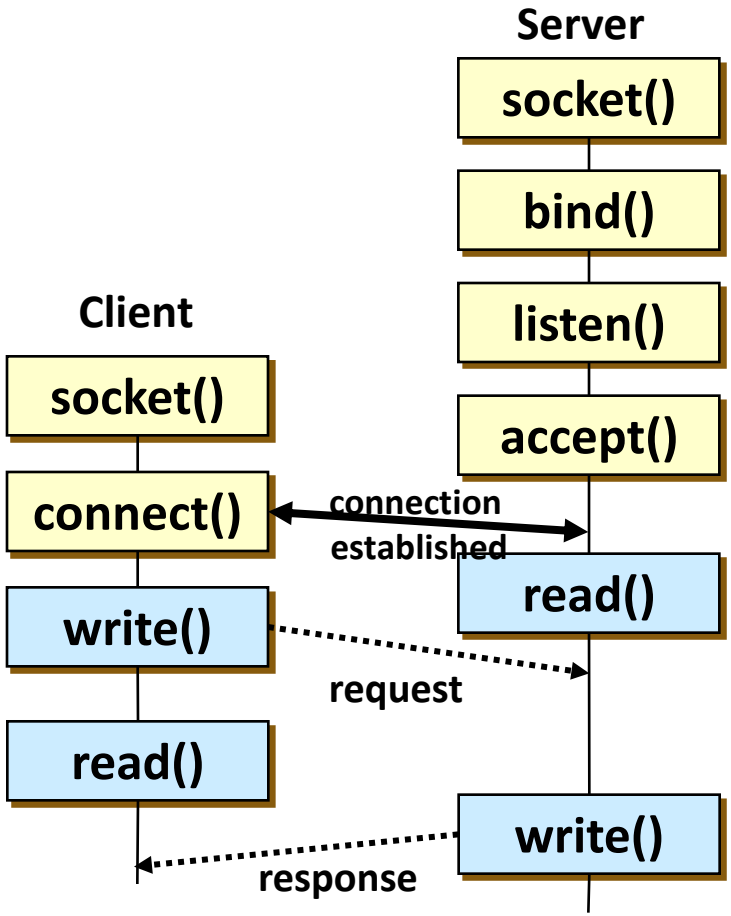
# Sockets (3)

- Hardware/Software organization of an Internet application

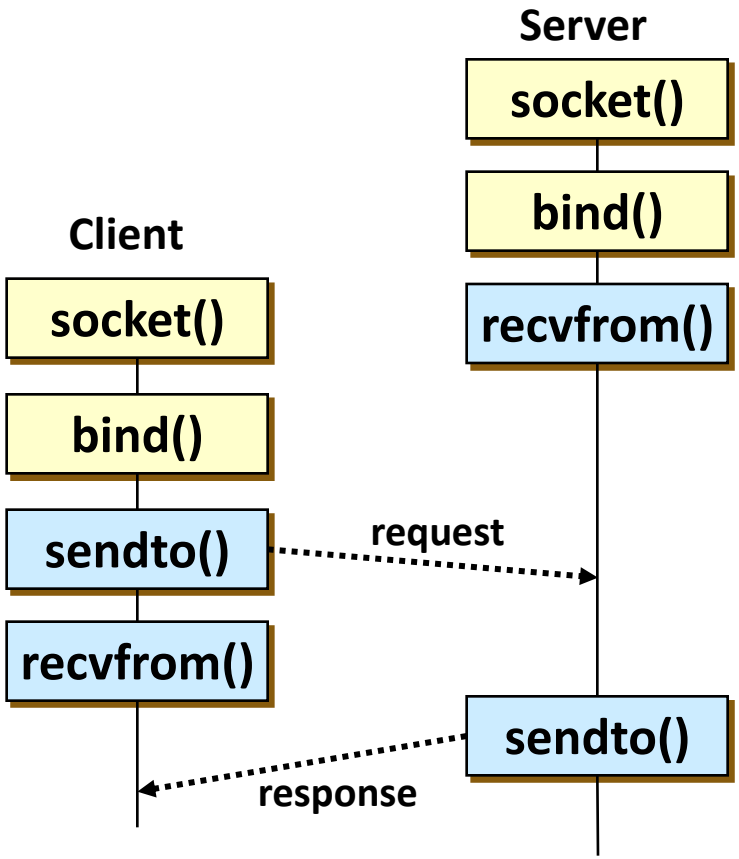


# Sockets (4)

## 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)
    - **SOCK_RAW**: provides raw network protocol access
  - **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);
}
```

# bind()

- int bind (int sockfd, struct sockaddr *myaddr,
  socklen_t addrlen)
  - **bind()** gives the socket **sockfd** the local address **myaddr**.
  - **myaddr** is **addrlen** bytes long.
  - Servers bind their well-known port when they start.
  - If a TCP server binds a specific IP address to its socket, this restricts the socket to receive incoming client connections destined only to that IP address.
  - Normally, a TCP client let the kernel choose an ephemeral port and a client IP address.

# listen()

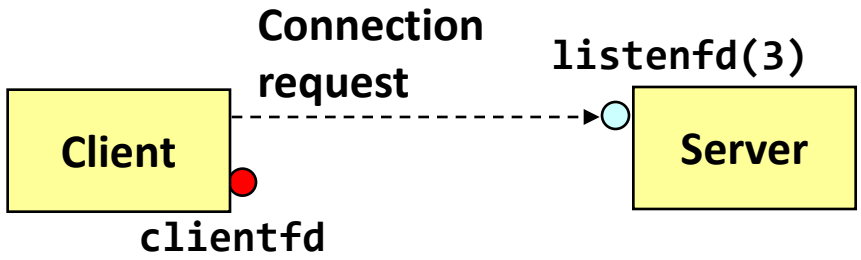
- int listen (int sockfd, int backlog)
  - **listen()** converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests.
    - When a socket is created, it is assumed to be an active socket, that is, a client socket that will issue a **connect().**
  - **backlog** specifies the maximum number of connections that the kernel should queue for this socket.
  - Historically, a backlog of 5 was used, as that was the maximum value supported by 4.2BSD.
    - Busy HTTP servers must specify a much larger backlog, and newer kernels must support larger values.

# accept() (1)

- int accept (int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen)
  - **accept()** blocks waiting for a connection request.
  - **accept()** returns a connected descriptor with the same properties as the listening descriptor.
    - The kernel creates one connected socket for each client connection that is accepted.
    - Returns when the connection between client and server is created and ready for I/O transfers.
    - All I/O with the client will be done via the connected socket.
  - The **cliaddr** and **addrlen** arguments are used to return the address of the connected peer process (the client)

# accept() (2)


*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`.*

# accept() (3)

- Listening descriptor
  - End point for client connection requests
  - Created once and exists for lifetime of the server
- Connected descriptor
  - End point of the connection between client and server
  - A new descriptor is created each time the server accepts a connection request from a client.
  - Exists only as long as it takes to service client.
- Why the distinction?
  - Allows for concurrent servers that can communicate over many client connections simultaneously.

# Echo Server (1)

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

#define MAXLINE 80

int main (int argc, char *argv[]) {
   int n, listenfd, connfd, caddrlen;
   struct hostent *h;
   struct sockaddr_in saddr, caddr;
   char buf[MAXLINE];
   int port = atoi(argv[1]);

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

# Echo Server (2)

```
bzero((char *)&saddr, sizeof(saddr));
saddr.sin_family = AF_INET;
saddr.sin_addr.s_addr = htonl(INADDR_ANY);
saddr.sin_port = htons(port);
if (bind(listenfd, (struct sockaddr *)&saddr,
         sizeof(saddr)) < 0) {
   printf("bind() failed.\n");
   exit(2);
}
if (listen(listenfd, 5) < 0) {
   printf("listen() failed.\n");
   exit(3);
}
while (1) {
   caddrlen = sizeof(caddr);
   if ((connfd = accept(listenfd, (struct sockaddr *)&caddr,
                        &caddrlen)) < 0) {
      printf ("accept() failed.\n");
      continue;
   }
```

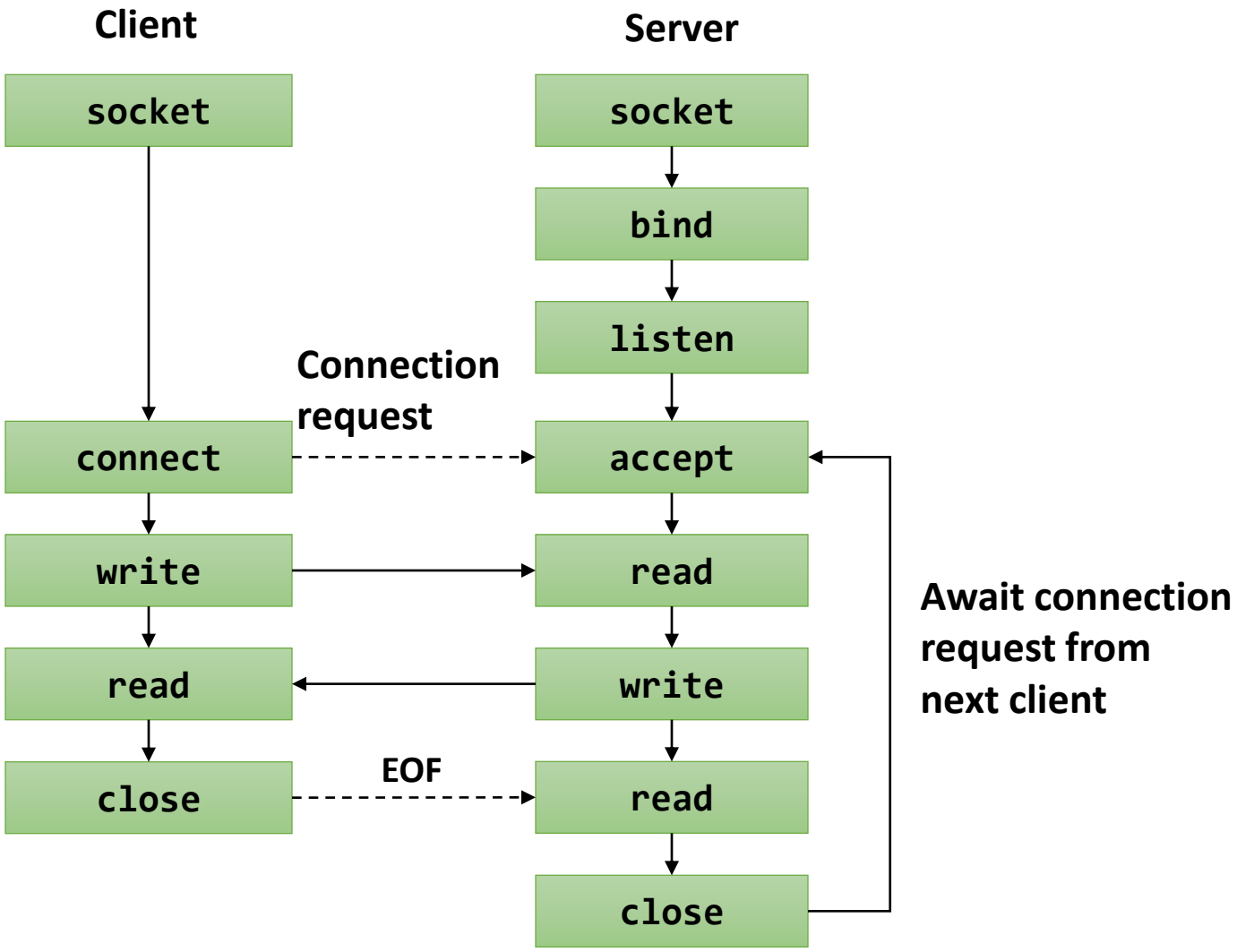
# Echo Server (3)

```
        h = gethostbyaddr((const char *)&caddr.sin_addr.s_addr,
            sizeof(caddr.sin_addr.s_addr), AF_INET);
        printf("server connected to %s (%s)\n",
            h->h_name,
            inet_ntoa(*(struct in_addr *)&caddr.sin_addr));

        // echo
        while ((n = read(connfd, buf, MAXLINE)) > 0) {
           printf ("got %d bytes from client.\n", n);
           write(connfd, buf, n);
        }

        printf("connection terminated.\n");
        close(connfd);
    }
}
```

# Echo Server (4)



# Exercise #1

- “Guess the Number”
  - The server generate a random number(0 to 99) when the client is connected.
  - The client guess the number the server generate.

```
$./server [port] &
$./client 127.0.0.1 [port]
Guess? 32
Up
Guess? 74
Down
Guess? 34
Correct!
$
```