



Socket Programming

Lecture 35

Klara Nahrstedt

CS241 Administrative

- Read Stallings Chapter 13, R&R 18.1-18.3
- LMP3 (Part I) Due April 23
- LMP3 (Part II) Due April 30
- Last Quiz will be on Friday, April 27 on Networking

Connection-oriented Communication Protocol

Server monitors a **passive end-point** whose address is known to clients

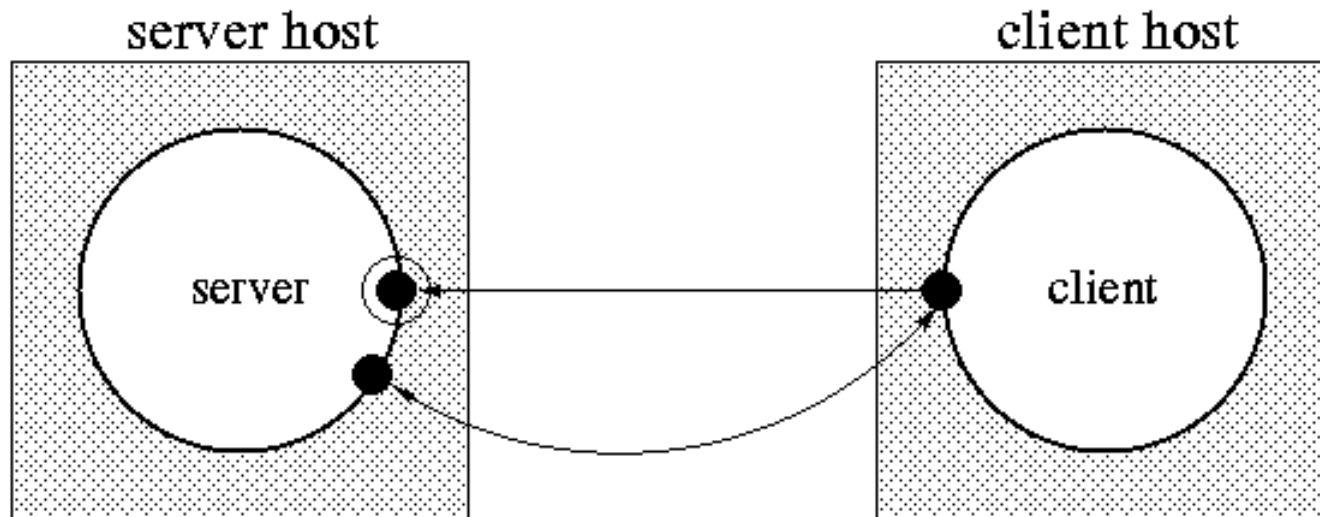
Listening (passive) endpoints have resources for queuing client connection requests and establishing client connections

Action of accepting a client request creates a new endpoint for private, **two-way symmetric communication** with that client

Client and server communicate by using **handles (file descriptors)** and do not explicitly include addresses in their messages

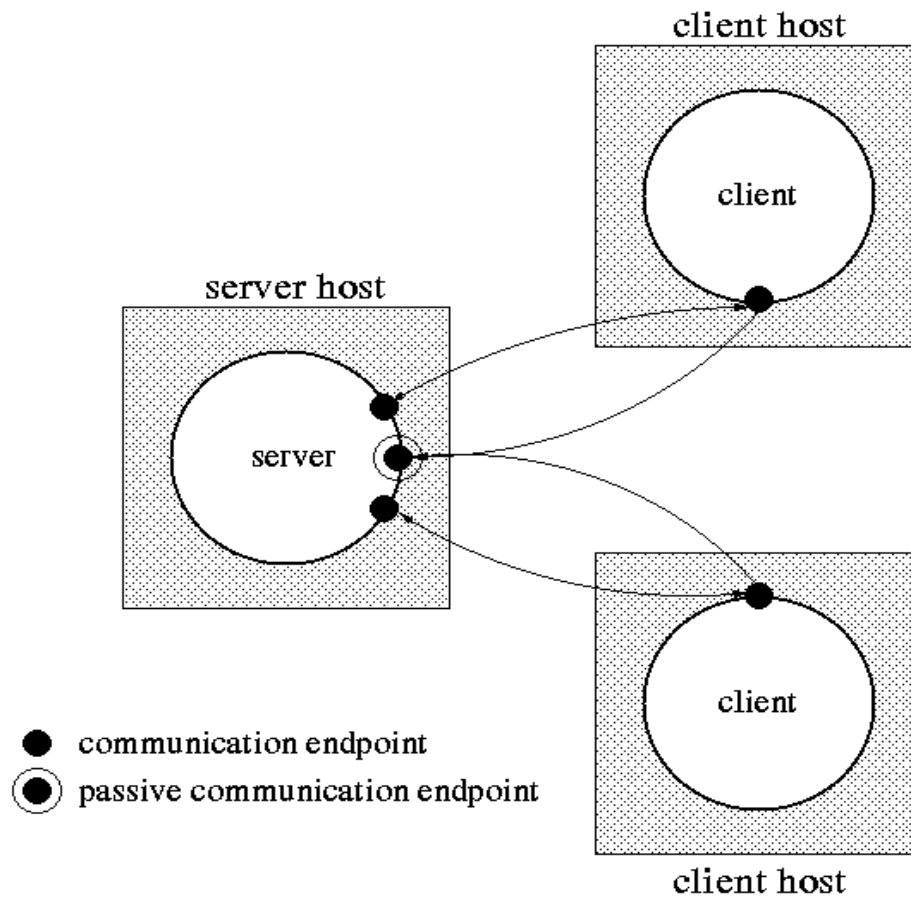
When finished, client and server **close their file descriptors**, system releases resources associated with the connection

Connection-Oriented Communications Illustration

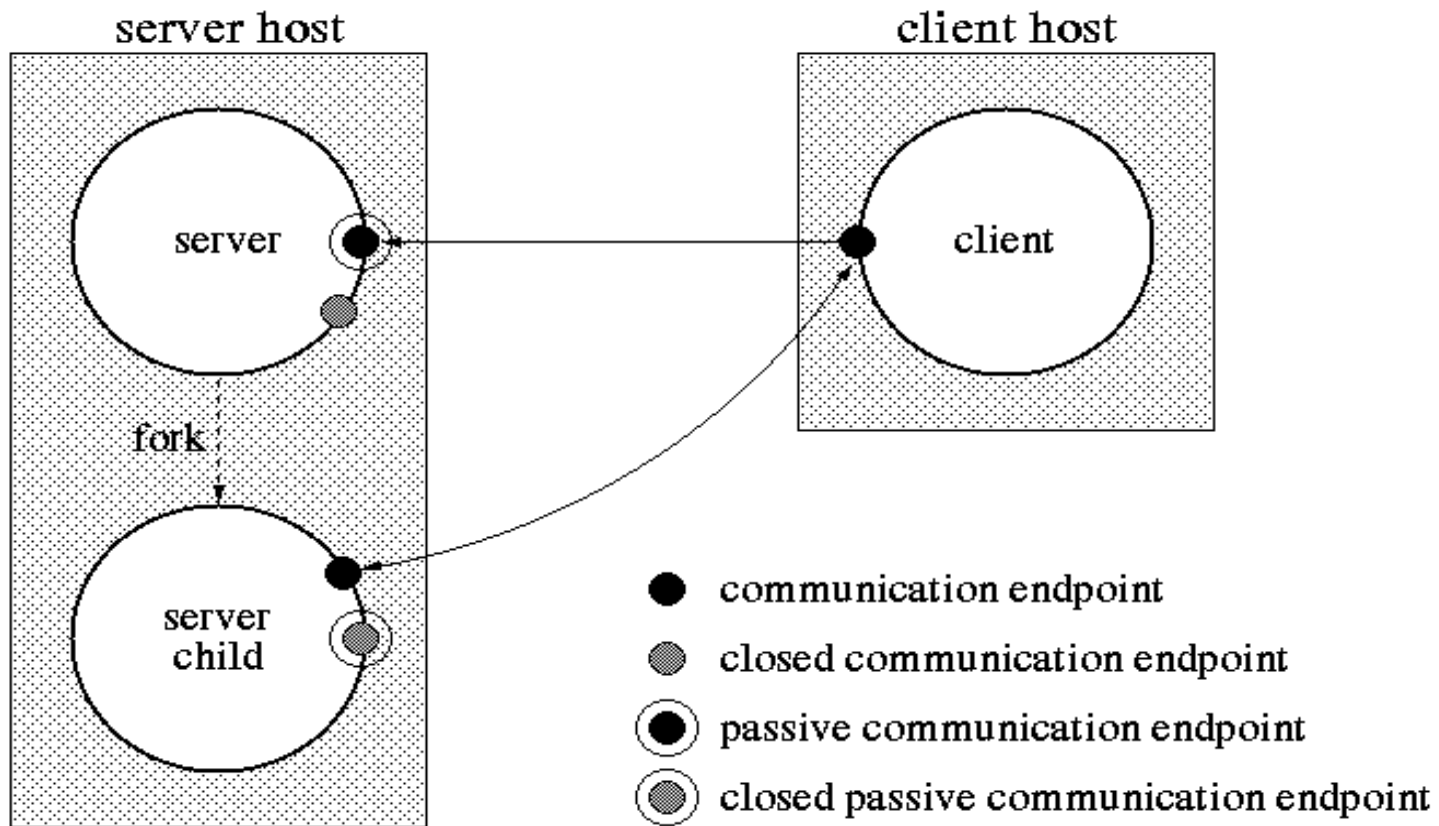


- communication endpoint
- passive communication endpoint

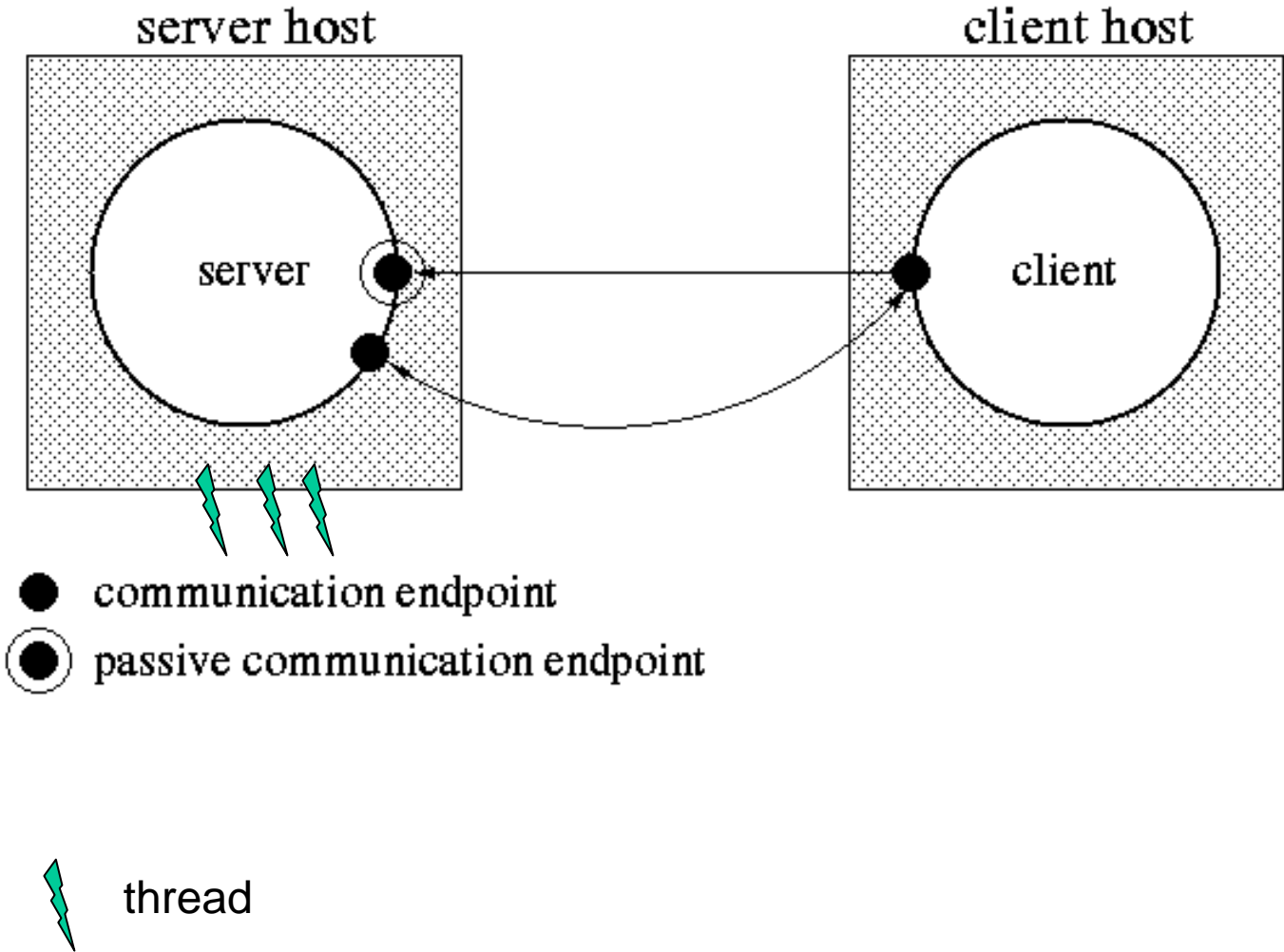
Multiple Clients (Serial-server Strategy)



Parent-server Strategy



Threaded-server Strategy



What is a socket?

An **interface between application and network**

The application creates a socket

The socket *type* dictates the style of communication

reliable vs. best effort

connection-oriented vs. connectionless

Once configured, the application can

pass data to the socket for network transmission

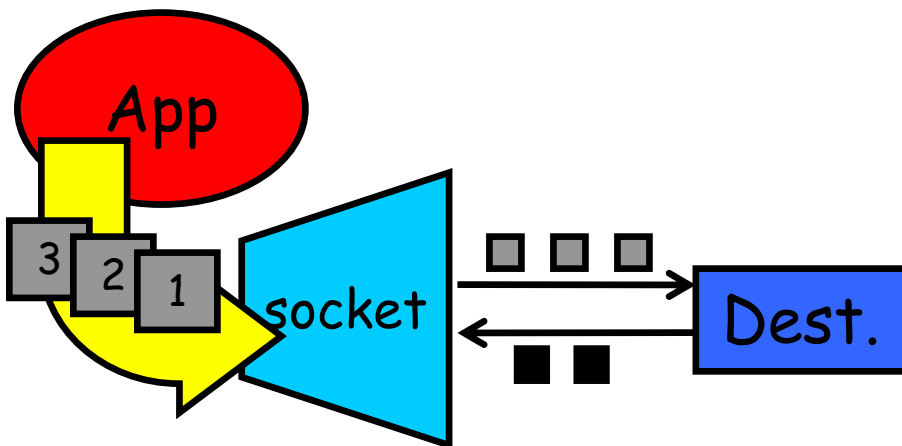
receive data from the socket (transmitted through the network by some other host)



Two essential types of sockets

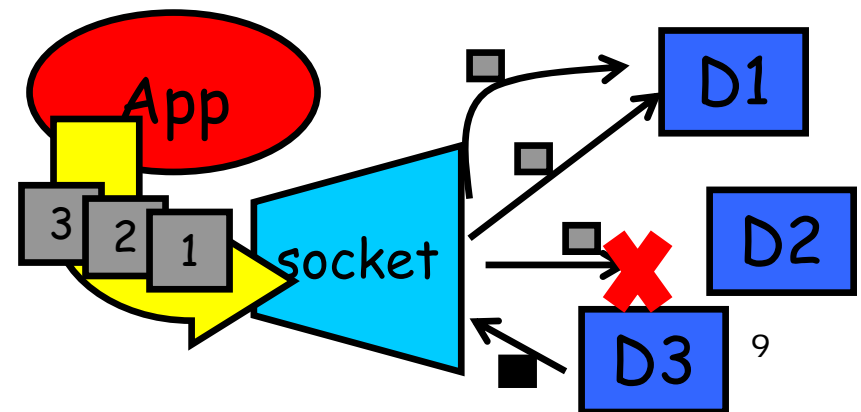
- **SOCK_STREAM**

- a.k.a. TCP
- reliable delivery
- in-order guaranteed
- connection-oriented
- bidirectional



- **SOCK_DGRAM**

- a.k.a. UDP
- unreliable delivery
- no order guarantees
- no notion of “connection” – app indicates dest. for each packet
- can send or receive



Socket-Based APIs

```
#include <sys/socket.h>
```

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

```
int bind (int s, const struct sockaddr *address,  
size_t address_len);
```

```
int listen (int s, int backlog);
```

```
int accept (int s, struct sockaddr *restrict address,  
int *restrict address_len);
```

Socket Creation in C: socket

```
int s = socket(domain, type, protocol);
```

s: socket descriptor, an integer (like a file-handle)

domain: integer, communication domain

e.g., AF_INET typically used (Internet domain)

type: communication type

SOCK_STREAM: reliable, 2-way, connection-based service

SOCK_DGRAM: unreliable, connectionless,

other values: need root permission, rarely used, or obsolete

protocol: specifies protocol (see file /etc/protocols for a list of options) - usually set to 0 (IP)

NOTE: socket call does not specify where data will be coming from, nor where it will be going to – it just creates the interface!

Addresses, Ports and Sockets

Like apartments and mailboxes

You are the application

Your apartment building address is the address

Your mailbox is the port

The post-office is the network

The socket is the key that gives you access to the right mailbox

Q: How do you choose which port a socket connects to?

Bind Function

```
int bind(int s, const struct sockaddr *address, size_t  
address_len);
```

Bind function

Associates the handle for a socket communication endpoint with a specific logical network connection.

Note: Internet domain protocols specify logical connection by a port number

s is the file descriptor returned by `socket()`

address contains info about the family, port and machine

address_len is the size of the structure used for the address

Socket Structure (1)

The format of the address `struct sockaddr` is determined by the address family (domain).

For `AF_INET` it is a `struct sockaddr_in`

Socket structure is defined in `netinet/in.h`

Socket structure has at least the following members in network byte order:

```
sa_family_t sin_family;  
in_port_t sin_port; /* Host Port Number */  
struct in_addr sin_addr; /* IP Address*/
```

Socket Structure (2)

```
sa_family_t sin_family;  
in_port_t sin_port;  
struct in_addr sin_addr;
```

The `sin_family` should be `AF_INET`.

The `sin_port` should be the port number in network byte order. You can convert a port number to **network byte order** using:

```
uint16_t htons(uint16_t port);
```

For a server, the `sin_addr` can be set to `INADDR_ANY` to accept connections on any interface card.

For a client, it must be filled in with the IP address of the remote machine in **network byte order**.

Socket Structure: Example

```
struct sockaddr_in server;
```

```
int sock;
```

```
server.sin_family = AF_INET;
```

```
server.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
server.sin_port = htons((short)8652);
```

```
if (bind(sock, (struct sockaddr *)&server, sizeof(server)) == -1)  
    perror("Failed to bind the socket to port");
```

Sending / Receiving Data - UDP

Without a connection (SOCK_DGRAM):

```
int count = sendto(sock, &buf, len, flags, &addr, addrlen);
```

count, sock, buf, len, flags: same as send

addr: struct sockaddr, address of the destination

addrlen: sizeof(addr)

```
int count = recvfrom(sock, &buf, len, flags, &addr,  
                    &addrlen);
```

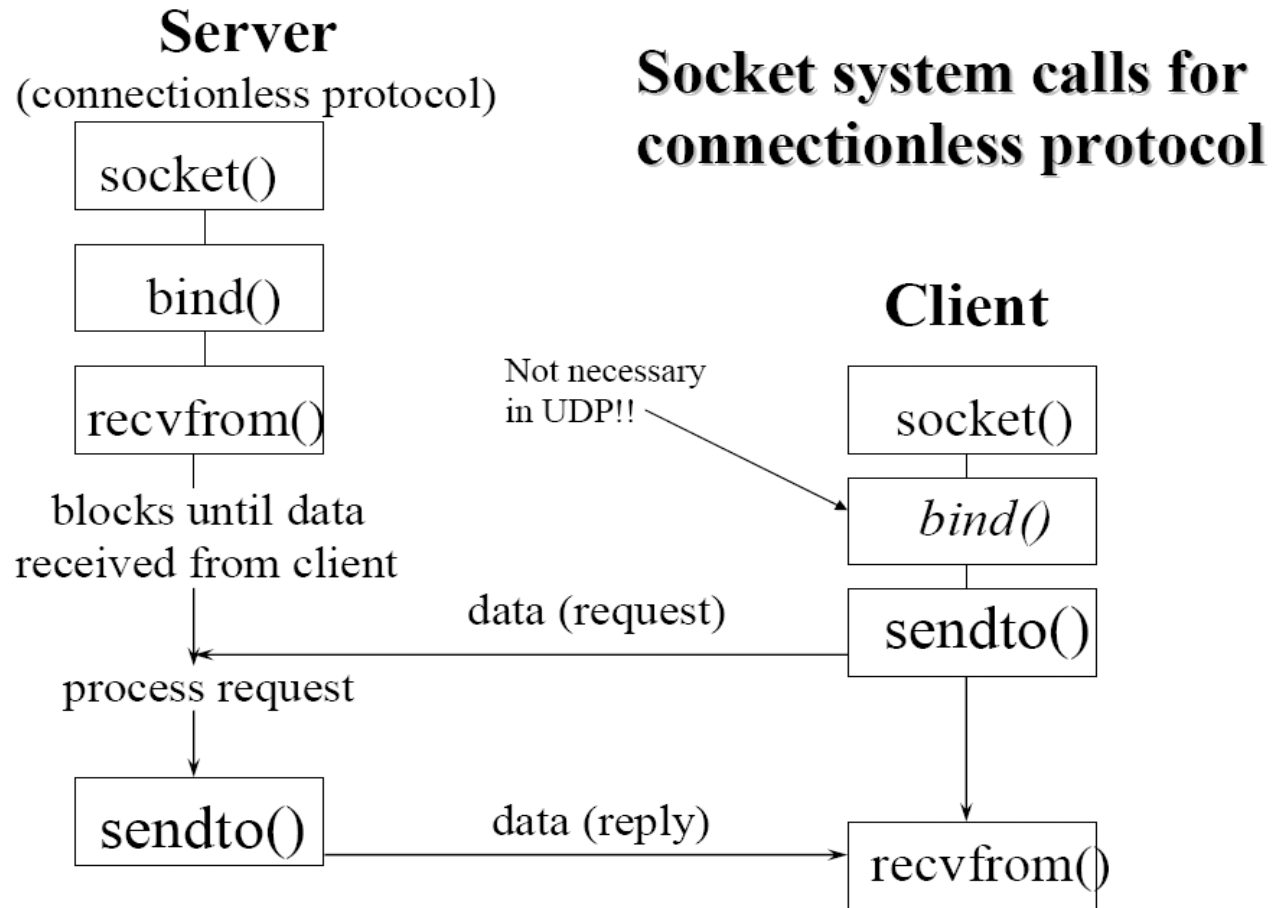
count, sock, buf, len, flags: same as recv

name: struct sockaddr, address of the source

namelen: sizeof(name): value/result parameter

Calls are **blocking** [returns only after data is sent (to socket buf) / received]

Typical UDP Server-Client



Summary

Client-Server Process Communication

Connection-oriented Server Strategies

Implementation of UDP