# Chapter 2
## Client/Server Model

A structured programming method for networks

Analogy

| | | |
|---|---|---|
| Customer | ------- | Waiter |
| (Client) | | (Server) |
| orders from menu | | delivers dinner |

| | | |
|---|---|---|
| Waiter | ------- | Cook |
| (Client) | | (Server) |
| submits order | | cooks food |

For each job you either request or deliver service

Analogy 2:

| | | |
|---|---|---|
| main program | ------- | procedure |
| (Client) | | (Server) |
| makes call | | delivers answer |

# Client/Server Example

telnet cheetah.cecs.csulb.edu

`telnet` is a client program

This client is directed to request service from `cheetah`

`telnetd` is a server program

This server is running on cheetah waiting for service requests

What is produced: a login session.

`telnet` delivers user commands to `telnetd` on cheetah

`telnetd` executes them (on cheetah)
and returns the answer to the client

# Typical Client Actions

1. Open a connection to a server.
2. Send requests to the server and receive replies.
3. Close the connection.

Analogy (Phone)

1. Make a phone call.
2. Ask questions, get replies.
3. Hang up.

Analogy (Mail)

1. Write letter.
2. Mail letter.
3. Get reply.

# Typical Server Actions

1. Await a connection.
2. Receive requests from a client and send replies.
3. Close the connection after client does.

Analogy (Phone)

1. Wait in office.
2. Answer phone.
3. Listen to questions, give replies.
4. Hang up after the customer does.

Analogy (Mail)

1. Wait for mail.
2. Read each letter.
3. Give reply.

## Connection-Oriented vs. Connectionless Models

Connection-Oriented Model (TCP)
*Transmission Control Protocol*
Analogy: Phone

A connection must be established.

Connection is interactive.

Once connection is made, reliable delivery

Note: TCP guarantees delivery of information in the order sent or an error will be signaled.

Connectionless Model (UDP)
*User Datagram Protocol*
Analogy: Mail

Messages are sent.

A message is not interactive.

Delivery not confirmed, not guaranteed.

Note: With UDP it is up to the letter writers to provide confirmation, and use a series of messages to achieve interaction.

# Client/Server Problems

1. memory and programs don't survive a crash (reboot)
2. messages get lost
3. messages get duplicated
4. messages don't always arrive in the same order they were sent

A server that relies on the history of the interaction often behaves incorrectly if any of the above occur.

A server that relies on the history of an interaction, must ensure that the history is accurate.

Conclusion: models that depend the history of an interaction (states) are harder to program.

Note: 2,3,4 will not happen with TCP

# The Stateless Model

Principle: server doesn't have a memory

Consequence: each message received by the server must be self-contained

Principle: client doesn't rely on server's remembering.

Consequence Each message a client sends must contain a self-contained command.

Do not depend on a previous command
(get-record(6) is ok, but get-*next*-record isn't)

Stateless type problems:

Did the request get through?

How do you know?

Due to confusion a command could be done twice!