Types of Servers

Connectionless (UDP) verse Connection-Oriented (TCP)

Iterative verses Concurrent

Stateless verse stateful
Iterative verses Concurrent Server

**Iterative**

Single process

Handles requests one at a time

Good for low volume & requests that are answered quickly
Iterative verses Concurrent Server

Concurrent

Handle multiple requests concurrently

Normally uses thread/processes

Needed for high volume & complex requests

Harder to implement than iterative

Must deal with currency
require 'socket'

class DateServer
  def initialize(port)
    @port = port
  end

  def run()
    server = TCPServer.new( @port)
    puts("start " + @port.to_s)
    while (session = server.accept)
      Thread.new(session) do |connection|
        process_request_on(connection)
        connection.close
      end
    end
  end
end

def process_request_on(socket)
  request = canonical_form( socket.gets("\n") )
  now = Time.now
  answer = case request
    when 'time'
      now.strftime("%X")
    when 'date'
      now.strftime("%x")
    else
      "Invalid request"
  end
  socket.send(answer + "\n",0)
end

def canonical_form(string)
  string.lstrip.rstrip.downcase
end
One can implement a concurrent server using one thread/process

while (true) {
    check if any new connects (non-block accept)
    if new connection accept
    process a little on each current request
}
Stateless verses Stateful Servers

State information

Information maintained by server about ongoing interactions with clients

Consumes server resources

How long does one maintain the state?
Modes of Operation

Stateful servers sometimes have different modes of operation

Each mode has a set of legal commands

In Login mode only the commands password & username are acceptable

After successful login client-server connection in transaction mode

In transaction mode command X, Y Z are legal

These modes are also called server states or just states
Protocol

Requirements for a "good protocol"

Well defined

Complete

Parsable

Extendable

Available protocol document
Assignment 2 Protocol

Client commands
- `count url;
- `reset url;

Server Responses
- `n ISO-8601Date;
- `reset url;
- `Invalid command;

<table>
<thead>
<tr>
<th>Client Request</th>
<th>Server Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>count /foo</td>
<td>1 2006-2-2</td>
</tr>
<tr>
<td>count /foo ;</td>
<td>2 2006-2-2</td>
</tr>
<tr>
<td>count /bar/foo ;</td>
<td>1 2006-2-2</td>
</tr>
</tbody>
</table>
Every bit of data sent in either direction has to have its place in the protocol description.

Protocol is a Language

Common formal description:
   BNF and Augmented BNF

Format of the description language needs to be part of the protocol document.

Examples are important
The protocol must cover all possible situations.

Garbage data
Old client or server (different protocol versions)
Illegal requests
Boundary conditions
Etc.
Both clients and servers are computer programs.

A computer program's IQ is generally 0.

**Design goals**

Distinct information packets or messages

- Allow parsing independent of semantics

Consistency

- Allow for code reuse

Flexibility
Allow parsing independent of semantics

Client commands A
count url;
reset url;

Client commands B
count url;
reset url^;

How does the server parse each set of commands?
Different groups may write clients and servers at different times.

Central registry for Internet protocols

Self regulating:
  RFC - Request For Comment
  IETF - Internet Engineering Task Force

Official:
  ISO
  ANSI
Protocol Types

Typical **synchronous**

Client sends request to server
Server responds with a reply

HTTP, POP, SMTP, Gopher, XMODEM

Typical **asynchronous**

Client and server both send information to each other concurrently.

TELNET, RLOGIN, ZMODEM

A hybrid protocol is also possible
Protocol Design Issues

Protocol design is difficult!
Learn from examples

Some issues

Protocol extendibility and versioning

Byte order used for sending values

ASCII vs. Binary protocol

Synchronous vs. Asynchronous

State

Timeouts