L1: Complexity, Enforced Modularity, and client/server organization

Sam Madden and Dina Katabi

6.033 Spring 2014
http://web.mit.edu/6.033
http://web.mit.edu/6.033

- Schedule has all assignments
  - Every meeting has preparation/assignment

- On-line registration form to sign up for section and tutorial times
  - We will post sections assignment this evening

<table>
<thead>
<tr>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Feb 3</strong></td>
<td><strong>Reg day</strong></td>
<td><strong>Feb 4</strong></td>
<td><strong>Feb 5</strong></td>
<td><strong>Feb 6</strong></td>
</tr>
<tr>
<td><strong>Feb 10</strong></td>
<td><strong>LEC 2: Naming</strong></td>
<td><strong>REC 1: Worse is Better</strong></td>
<td><strong>LEC 1: Enforced Modularity and</strong></td>
<td><strong>REC 2: Therac-25</strong></td>
</tr>
<tr>
<td>Preparation:</td>
<td>Preparation:</td>
<td>Preparation:</td>
<td>Preparation:</td>
<td></td>
</tr>
<tr>
<td>Book sections 2.2, and 3.1</td>
<td>Read Worse is Better</td>
<td>Client/server Organization</td>
<td>Therac-25 paper</td>
<td></td>
</tr>
<tr>
<td>DUE: Hands-on DNS</td>
<td>Assigned: Hands-on DNS</td>
<td>Preparation:</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Feb 11</strong></td>
<td><strong>REC 3: DNS</strong></td>
<td><strong>Feb 12</strong></td>
<td><strong>Feb 13</strong></td>
<td><strong>Feb 14</strong></td>
</tr>
<tr>
<td>Preparation:</td>
<td>Preparation:</td>
<td><strong>LEC 3: Operating systems</strong></td>
<td><strong>REC 4: UNIX</strong></td>
<td><strong>TUT 2: Design project 1</strong></td>
</tr>
<tr>
<td>Book section 4.4</td>
<td>Case study: The Internet Domain Name System (DNS)</td>
<td>Preparation:</td>
<td>Preparation:</td>
<td>(run by TAs)</td>
</tr>
<tr>
<td>DUE: Hands-on DNS</td>
<td>Assigned: Hands-on UNIX</td>
<td>Book sections 5.1, 5.3, and 5.4</td>
<td>Unix paper</td>
<td></td>
</tr>
<tr>
<td><strong>Feb 13</strong></td>
<td><strong>Feb 14</strong></td>
<td><strong>Feb 15</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Feb 14</strong></td>
<td><strong>REC 4: UNIX</strong></td>
<td><strong>Feb 15</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Preparation:</td>
<td><strong>TUT 2: Design project 1</strong></td>
<td><strong>REC 3: Operating systems</strong></td>
<td><strong>REC 4: UNIX</strong></td>
<td></td>
</tr>
<tr>
<td>Book section 2.5</td>
<td>Case study: UNIX File System Layering and Naming</td>
<td>Preparation:</td>
<td>Preparation:</td>
<td></td>
</tr>
<tr>
<td>DUE: Memo #1</td>
<td>Assigned: Memo #1</td>
<td>Assignment:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What is a system?

System = *Interacting set of components with a specified behavior at the interface with its environment*

Examples: Web, Linux

6.033: study and design of systems, their components, and internals
6.033 Approach

- **Lectures/book**: big ideas and examples
- **Hands-ons**: play with successful systems
- **Recitations**: papers describing successful systems
- **Design projects**: you practice designing and writing
  - Design: choose problem, tradeoffs, structure
  - Writing: explain core ideas concisely
- **Exams**: focus on reasoning about system design
Why is building systems hard?
Example Complex System: Linux Kernel

- 1975 Unix kernel: 10,500 lines of code
- 2008 Linux 2.6.24 line counts:
  - 85,000 processes
  - 430,000 sound drivers
  - 490,000 network protocols
  - 710,000 file systems
  - 1,000,000 different CPU architectures
  - 4,000,000 drivers
  - 7,800,000 Total

- More examples: http://www.informationisbeautiful.net/
Emergent Property Example: Ethernet

- All computers share single cable
- Goal is reliable delivery
- Listen while sending to detect collisions
Emergent Property Example: Ethernet

- All computers share single cable
- Goal is reliable delivery
- Listen while sending to detect collisions
Does Collision Detection Work?

What if A finishes sending before data from B arrives? Can this happen?

1 km at 60% speed of light = 5 microseconds
Original Ethernet Spec: 3 Mbit / sec
- A can send 15 bits before bit 1 arrives at B
- A must keep sending for 2 * 5 microseconds
  (To detect collision when first bit from B arrives)
- Minimum packet size is 5 * 2 * 3 = 30 bits
Default header is 5 bytes (40 bits), so no problem!
3 Mbit/s ➔ 10 Mbit/s

- First Ethernet standard: 10 Mbit/s, 2.5 km wire
  - Must send for 2*12.5 μseconds = 250 bits @ 10 Mb/s
  - Header was 14 bytes
    ➔ Needed to pad packets to at least 250 bits (32 bytes)

Emergent property: Minimum packet size!
A computer system scaling example
Scaling the Internet

- Size routing tables (for shortest paths): O(n²)
  - Hierarchical routing on network numbers
  - Address: 16 bit network # and 16 bit host #
- Limited networks (2¹⁶)

=> Network Address Translators and IPv6
Example: No Small Changes

Phone network features
• Call Forwarding
• Call Number Delivery Blocking
• Automatic Call Back
• Itemized Billing

CNDB  ACB + IB
A  B

• A calls B, B is busy
• Once B is done, B calls A
• A’s number on appears on B’s bill
How can we mitigate the complexity of building systems?
Enforcing Modularity with Client/Server
Remote Procedure Call

- Stubs make client/server look like procedure calls!
- Stubs can be automatically generated

Web Client

```python
def main:
    html = load(URL)
    render(html)

def loadStubClient:
    msg ← URL
    send request
    wait for reply
    html ← reply
    return html
```

Web Server

```python
def serverLoad(URL):
    ...
    return html

def loadStubServer:
    wait for request
    URL ← request
    html = serverLoad(URL)
    reply ← html
    send reply
```

Web Client

Stub

Request

Reply

Web Server

Stub
RPC != PC

Load("view.html?bieberAlbum") ➞ HTML
Load("buy.html?bieberAlbum&ccNo=xxx")
Challenge 1: network loses requests

- Approach: Retry after time out
- Doesn’t work for buy.html
**Soln: Filter Duplicate Request**

- **What if server fails?**

**Diagram:**
- Client
  - Retry
  - Load("buy.html")
  - time
- Server
  - Saved Responses
    - Client | UID | Reply
  - Replay result from table – don’t reprocess order!
Challenge 2: server fails

• “Unknown” outcome for load(“buy.html”)
  Did the server process the request or not?

• Removing “unknown” outcome requires heavy-duty techniques
  Topic for April

• Practical solution: Expose that RPC != PC
  RPC caller must handle “serverFailed” exception
Summary so far

• Complexity makes building systems difficult
• Modularity and abstraction bound complexity
• Can enforce modularity through client/server
  • Remote procedure call simplifies C/S
  • Unfortunately, RPC != PC
• Failures will be a central challenge in 6.033
• No algorithm for successful system design
Example 6.033 systems

- Therac-25
  bad design, at many levels. detailed post-mortem
- UNIX
- The Internet
- MapReduce
- Relational Databases
Class plan

• **Client/server**: Naming

• **Operating systems**:  
  • Enforced modularity within a machine

• **Networks**:  
  • Enforced modularity between machines

• **Reliability and transactions**:  
  • Handling hardware failures

• **Security**: handling malicious failures