Modularity and Separation of Concerns

Noah Mendelsohn
Tufts University
Email: noah@cs.tufts.edu
Web: http://www.cs.tufts.edu/~noah
Goals

- Explore the benefits of modularity and separation of concerns
- Explore some of the *limits and drawbacks* of modular systems
Abstracting the Hard Disk
What’s a hard disk?

- Now: http://www.youtube.com/watch?v=3owqvmMf6No
- Then: http://www.youtube.com/watch?v=CUeXy80zMBg&t=19s
What’s a hard disk?

Typical Characteristics

- Fixed sized data blocks (512bytes -> 4K bytes)
- Seek time: 3ms – 15ms (depends on drive and distance)
- Rotational delay: ~5ms for commodity drives
- Transfer rate from platter: 100MBytes/sec
How does our software show us the disk?

- **Filesystem**
  - Names: `/home/noah/myfile.txt`
  - Files can grow and shrink dynamically
  - Geometry and timing hidden
  - Free space managed transparently
  - Sharing and security
  - Buffering and optimization
  - May span multiple drives

- **Relational database**
  - Collections of tables: rows + columns
  - Access via query language
How is the disk used in Unix / Linux?

- **Application**
  - **Filesystem**
    - **Unix Kernel**
    - **Files/Dirs**
      - security, etc
    - **Buffered block r/w**: hides timing
  - **Block Device Driver**
    - In-memory Block Cache
    - Sector
    - **Direct read/write of filesystem “blocks”** (hides sector size and device geometry)
  - **Raw Device Driver**
    - Sector
    - Access by cylinder/track/sector
How is the disk used in Unix / Linux (over-simplified)

Application ─ Filesystem ─ Raw Device Driver ─ Sector

Unix Kernel

Files/Dirs
security, etc

Direct read/write of filesystem “blocks” (hides sector size and device geometry)

Access by cylinder/track/sector
How is the disk used in Unix / Linux?

Application  

Files/Dirs, security, etc

Unix Kernel

Files/Dirs security, etc

In-memory Block Cache

Sector

Buffered block r/w: hides timing

Direct read/write of filesystem "blocks" (hides sector size and device geometry)

Access by cylinder/track/sector
Things to note

- Each layer provides clean abstraction for next

- Code replaceable by layer
  - New filesystem on same block driver
  - New raw driver supports new device (different manufacturer, SSD, USB key, digital camera, etc.)
  - Cached block space supports (nearly) same interface as uncached

- Reuse!
  - All devices supported by common buffer management and filesystem
  - Common APIs at all levels above device
Network Layering Revisited
# Architecture of the Internet Protocols

<table>
<thead>
<tr>
<th>Layer</th>
<th>Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Program</td>
<td>Use the network for some purpose</td>
<td>Firefox, Apache Server, Your program</td>
</tr>
<tr>
<td>Application Layer</td>
<td>Protocols with application-specific semantics</td>
<td>HTTP (Web)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SMTP (E-mail)</td>
</tr>
<tr>
<td>Transport Layer</td>
<td>User-level connection &amp; datagram</td>
<td>TCP/UDP</td>
</tr>
<tr>
<td>Internet Layer</td>
<td>Unreliable, multi-hop packet delivery</td>
<td>IP Packet Routing</td>
</tr>
<tr>
<td>Link Layer</td>
<td>Send an IP Packet over Hardware</td>
<td>Ethernet, Wi-fi, Dial-up</td>
</tr>
</tbody>
</table>

We can replace link layer and still use upper layers!
Compare the following RFC’s


Please note that RFC 1149 support has been demonstrated: [http://www.blug.linux.no/rfc119/](http://www.blug.linux.no/rfc1149/)
# Architecture of the Internet Protocols

<table>
<thead>
<tr>
<th>Layer</th>
<th>Purpose</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Program</td>
<td>Use the network for some purpose</td>
<td>Firefox, Apache Server, Your program</td>
</tr>
<tr>
<td>Application Layer</td>
<td>Protocols with application-specific semantics</td>
<td>HTTP (Web) SMTP (E-mail)</td>
</tr>
<tr>
<td>Transport Layer</td>
<td>User-level connection &amp; datagram</td>
<td>TCP/UDP</td>
</tr>
<tr>
<td>Internet Layer</td>
<td>Unreliable, multi-hop packet delivery</td>
<td>IP Packet Routing</td>
</tr>
<tr>
<td>Link Layer</td>
<td>Send an IP Packet over Hardware</td>
<td>Ethernet, Wi-fi, Dial-up</td>
</tr>
</tbody>
</table>

Implementations are often layered to match the architecture!
Overview of Layering/Modularity Issues
Some terms

- Separation of concerns
- Modularity
- Layering
- Encapsulation
- Information hiding
- Abstraction
- Reuse
Separation of concerns – HTTP

HTTP Status Codes Evolve Orthogonally from Rest of Protocol

HTTP/1.1 200 OK
Date: Tue, 28 Aug 2007 01:49:33 GMT
Server: Apache
Transfer-Encoding: chunked
Content-Type: text/html

<html>
<head>
<title>Demo #1</title>
</head>
<body>
<h1>A very simple Web page</h1>
</body>
</html>
Separation of concerns – HTTP

Media type registrations shared with E-mail (MIME) infrastructure

HTTP/1.1 200 OK
Date: Tue, 28 Aug 2007 01:49:33 GMT
Server: Apache
Transfer-Encoding: chunked
Content-Type: text/html

<html>
<head>
<title>Demo #1</title>
</head>
<body>
<h1>A very simple Web page</h1>
</body>
</html>
Separation of concerns – HTTP

Unicode, HTML and other specifications modular and shareable with other systems
Separation of concerns – HTTP

HTTP/1.1 200 OK
Date: Tue, 28 Aug 2007 01:49:33 GMT
Server: Apache
Transfer-Encoding: chunked
Content-Type: text/html

<html>
<head>
<title>Demo #1</title>
</head>
<body>
<h1>A very simple Web page</h1>
</body>
</html>

The HTML for the page.
Why modularity and encapsulation?

- Sharing and re-use
- Layers can evolve separately
- Synergies:
  - Photoshop and GIMP help everyone who uses JPEG
  - Including Web use of image/jpeg media type
- Reasoning about systems: correctness proofs, etc.
- Hiding complexity
- Progressive disclosure of complexity
- Making complex functions economical
Noah’s Theory of Simplification Choke Points

Very complex telephone switching system

Nationwide cable & fiber network & ESS Switches
Noah’s Theory of Simplification Choke Points

Wonderfully simple choke point interface

Nationwide cable & fiber network & ESS Switches

RJ-11 Jack & Touch Tones = Talk to anyone in the world using simple touch tone pad.

hook up devices
Noah’s Theory of Simplification Choke Points

Very complex modulation and signalling standard

Group 3 Fax Protocols

Nationwide cable & fiber network & ESS Switches

RJ-11 Jack & Touch Tones = Talk to anyone in the world using simple touch tone pad.. hook up devices
Noah’s theory...

Nationwide cable & fiber network & ESS Switches

Group 3 Fax Protocols

Drop in paper, dial #, paper delivered

RJ-11 Jack & Touch Tones = Talk to anyone in the world using simple touch tone pad.. hook up devices
Example: the Web Stack

- **Internet dynamic routing, ARP, etc.**
  - Drop in packet, probably gets there

- **Distributed DNS resolution**
  - a) name->ip addr
  - b) UDP Packet to named addr

- **TCP w/flow control, etc.**
  - Deliver stream to Named Destination

- **URIs, Hyperlinks, HTTP Get, Media typed streams, HTML**
  - Click to Browse, worldwide

Each layer hides significant complexity behind simple interface.
Layering and Performance
Layering can help performance

- Wrap highly tuned implementations in easy-to-use interfaces!
- Make those implementations easy to reuse
- This is a big, big deal!
- But...
Layering can *hurt* performance

- Layering can hurt!
- Layering can keep you from getting at details that need to be tuned

**Examples:**
- Disk errors
- TCP/IP performance
- Compiler optimizations
Layering and disk performance

- Many disks and device drivers automatically forward data to a spare cylinder when a sector goes bad ... spares are usually at inside or outside of disk

- But...the filesystem may put critical directory there, unaware access will be amazingly slow

- Thanks to Forest Baskett, who gave me this example in about 1980
Layering and TCP/IP Performance

- Hard to share buffers and get alignment right across TCP/IP software layers in the OS
- *Layered implementations can lead to data copying*
- *Studies show that TCP/IP implementations need to share buffers and optimizations across the device, IP, and TCP layers*
- The highest performing remote file systems share buffers between network and filesystem code
- Watson & Mamrak: “a common mistake is to take a layered design as a requirement for a correspondingly layered implementation.” *ACM Transactions on Computer Systems (TOCS), Volume 5 Issue 2, May 1987*
Layering and compiler optimizations

- Compiler front ends tend to respect language layering
- *Compiler code generators need to optimize across layers*

This code doesn’t compute anything useful, but it’s interesting to see how it would be optimized:

```c
int myArray[20];
For (i=0; i<19 && (myArray[i]/2 < 50); i++)
  myArray[i] += myArray[i+1]/2;
```

A good compiler will remember pointer to myArray[i] or even value myArray[i]/2 from previous loop iteration.
Abstractions Leak!
Leaky abstractions

- When you abstract something...you lose something
- Sometimes the details you lose show through
- These leaky details can cause big trouble!

See “The Law of Leaky Abstractions”
A posting by Joel Spolsky
http://www.joelonsoftware.com/articles/LeakyAbstractions.html

By the way, Joel is the person behind StackOverflow and other “Stack” sites
Leaky example: CPU memory

- CPU memory reads faster when locality is good
- Cache-aligned loads/stores faster
- Multi-core: memory access in one core can slow the other.
- Etc.
Leaky example: Filesystem performance

- Sequential access faster than random
  - Causes seeks
Summary
Summary

- Separation of concerns is one of the key principles of CS
- Proper layering and modularization of your designs and code will bring tremendous benefits
- But…beware of “leaky” abstractions, performance concerns, etc.