Goals for Today

• A couple of requested topics
  - Windows vs. Linux
  - Trusted Computing
• Peer-to-Peer Systems
  - OceanStore

Requests for Final topics

• Some topics people requested:
  - More about device drivers
  - Xbox/Playstation/gamecube/etc operating systems
  - Windows vs Linux
  - Trusted computing platforms
• About Device Drivers
  - Well, very complex topic.
  - Documentation associated with various operating systems
    » Many similarities, many differences
  - Good place to start:
    » Chapter 6 of “The design and Implementation of the 4.4 BSD Operating System” (on reserve for this class)
• Xbox vs Playstation etc
  - Well, most of these are custom OSs.
    » Original Xbox ran modified version of Window 2000
    » New Xbox 360 rumored to run modified version of original Xbox OS (i.e. a modified version of Windows 2000)
  - Most important property: Real Time scheduling
    » Ability to meet scheduling deadlines

Windows vs Linux

• Windows came from personal computer domain
  - Add-on to IBM PC providing a windowing user interface
    » Became “good at” doing graphical interfaces
  - Didn’t have protection until Windows NT
    » Multiple users supported (starting with Window NT), but can’t necessarily have multiple GUIs running at same time
  - Product differentiation model:
    » Purchase separate products to get email, web servers, file servers, compilers, debuggers...
• Linux came from long line of UNIX Mainframe OSs
  - Targeted at high-performance computation and I/O
    » High performance servers
    » GUI historically lacking compared to Windows
  - Protection model from beginning
    » Multiple users supported at core of OS
  - Full function Mainframe OS: email, web servers, file servers, ftp servers, compilers, debuggers, etc.
**Windows vs Linux**

- **Internal Structure is different**
  - Windows XP evolved from NT which was a microkernel
  - Core "executive" runs in protected mode
  - Many services run in user mode (Although Windowing runs inside kernel for performance)
  - Object-oriented design: communication by passing objects
  - Event registration model: many subsystems can ask for callbacks when events happen
  - Loadable modules for device drivers and system extension
- Linux Evolved from monolithic kernel
  - Many portions of kernel operate in same address space
  - Loadable modules for device drivers and system extension
  - Fewer layers ⇒ higher performance

- **Source Code development model**
  - Windows: closed code development
    - Must sign non-disclosure to get access to source code
    - "Cathedral" model of development: only Microsoft’s developers produce code for Windows
  - Linux: open development model
    - All distributions make source code available to analyze
    - "Bazaar" model of development: many on the net contribute to making Linux distribution

**Perceptions:**
- Windows has more bugs/is more vulnerable to viruses?
  - True? Hard to say for sure
  - More Windows systems ⇒ more interesting for hackers
- Linux simpler to manage?
  - True? Well, Windows has hidden info (e.g. registry)
  - Linux has all configuration available in clear text
- Microsoft is untrustworthy? Many distrust "the man"
  - Quick to adopt things like Digital Rights Management (DRM)
  - Quick to embrace new models of income such as software rental which counter traditional understanding of software
- Windows is slow?
  - This definitely seemed to be true with earlier versions
  - Less true now, but complexity may still get in the way

**Why choose one over other?**
- Which has greater diversity of graphical programs?
  - Probably Windows
- Which cheaper? Well, versions of Linux are "free"
- Which better for developing code and managing servers?
  - Probably Linux, although this is changing
  - OS API (e.g. system calls) definitely seem simpler

**Trusted Computing**

- **Problem:** Can't trust that software is correct
  - Viruses/Worms install themselves into kernel or system without users knowledge
  - Rootkit: software tools to conceal running processes, files or system data, which helps an intruder maintain access to a system without the user's knowledge
  - How do you know that software won’t leak private information or further compromise user's access?
- **A solution:** What if there were a secure way to validate all software running on system?
  - Idea: Compute a cryptographic hash of BIOS, Kernel, crucial programs, etc.
  - Then, if hashes don’t match, know have problem
- **Further extension:**
  - Secure attestation: ability to prove to a remote party that local machine is running correct software
  - Reason: allow remote user to avoid interacting with compromised system
- **Challenge:** How to do this in an unhackable way
  - Must have hardware components somewhere

**TCPA: Trusted Computing Platform Alliance**

- **Idea:** Add a Trusted Platform Module (TPM)
- Founded in 1999: Compaq, HP, IBM, Intel, Microsoft
- Currently more than 200 members
- **Changes to platform**
  - Extra: Trusted Platform Module (TPM)
  - Software changes: BIOS + OS
- **Main properties**
  - Secure bootstrap
  - Platform attestation
  - Protected storage
- **Microsoft version:**
  - Palladium
  - Note quite same: More extensive hardware/software system

**ATMEL TPM Chip**
(Used in IBM equipment)
Trusted Platform Module

<table>
<thead>
<tr>
<th>Functional Units</th>
<th>Non-volatile Memory</th>
<th>Volatile Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Num Generator</td>
<td>Endorsement Key (2048 Bits)</td>
<td>RSA Key Slot-0</td>
</tr>
<tr>
<td>SHA-1 Hash</td>
<td>Storage Root Key (2048 Bits)</td>
<td>RSA Key Slot-9</td>
</tr>
<tr>
<td>HMAC</td>
<td>Owner Auth Secret (160 Bits)</td>
<td>PCR-0…PCR-15</td>
</tr>
<tr>
<td>RSA Encrypt/ Decrypt</td>
<td>Key Handles</td>
<td>Key Handles</td>
</tr>
<tr>
<td>RSA Key Generation</td>
<td>Auth Session Handles</td>
<td>Auth Session Handles</td>
</tr>
</tbody>
</table>

- **Cryptographic operations**
  - Hashing: SHA-1, HMAC
  - Random number generator
  - Asymmetric key generation: RSA (512, 1024, 2048)
  - Asymmetric encryption/decryption: RSA
  - Symmetric encryption/decryption: DES, 3DES (AES)
- **Tamper resistant (hash and key) storage**

TCPA: PCR Reporting Value

- **Platform Configuration Registers (PCRO-16)**
  - Reset at boot time to well defined value
  - Only thing that software can do is give new measured value to TPM
    - TPM takes new value, concatenates with old value, then hashes result together for new PCR
- **Measuring involves hashing components of software**
- **Integrity reporting**: report the value of the PCR
  - Challenge-response protocol:
    - Challenger nonce Trusted Platform Agent
    - SignID(nonce, PCR, log), CID

TCPA: Secure bootstrap

- **Implications of TPM Philosophy?**
  - Could have great benefits
    - Prevent use of malicious software
    - Parts of OceanStore would benefit (mention later)
  - What does “trusted computing” really mean?
    - You are forced to trust hardware to be correct!
    - Could also mean that user is not trusted to install their own software
  - Many in the security community have talked about potential abuses
    - These are only theoretical, but very possible
  - Software fixing
    - What if companies prevent user from accessing their websites with non-Microsoft browser?
    - Possible to encrypt data and only decrypt if software still matches ⇒ Could prevent display of .doc files except on Microsoft versions of software
  - Digital Rights Management (DRM):
    - Prevent playing of music/video except on accepted players
    - Selling of CDs that only play 3 times?
Administrivia

- Final Exam
  - 12:30 - 3:30, December 17th
  - 220 Hearst Gym
  - Bring 2 sheets of notes, double-sided
- Project 4
  - Due date moved to Friday, 12/9
- Midterm II
  - Still Grading!

Peer-to-Peer: Fully equivalent components

- Peer-to-Peer has many interacting components
  - View system as a set of equivalent nodes
    » “All nodes are created equal”
  - Any structure on system must be self-organizing
    » Not based on physical characteristics, location, or ownership

Is Peer-to-peer new?

- Certainly doesn’t seem like it
  - What about Usenet? News groups first truly decentralized system
  - DNS? Handles huge number of clients
  - Basic IP? Vastly decentralized, many equivalent routers
- One view: P2P is a reverting to the old internet
  - Remember? (Perhaps you don’t)
  - Once upon a time, all members on the internet were trusted.
    » Every machine had an IP address.
    » Every machine was a client and server.
    » Many machines were routers and/or were equivalent
- But: peer-to-peer seems to mean something else
  - More about the scale (total number) of directly interacting components
  - Also, has a “bad reputation” (stealing music)

Research Community View of Peer-to-Peer

- Old View:
  - A bunch of flakey high-school students stealing music
- New View:
  - A philosophy of systems design at extreme scale
  - Probabilistic design when it is appropriate
  - New techniques aimed at unreliable components
  - A rethinking (and recasting) of distributed algorithms
  - Use of Physical, Biological, and Game-Theoretic techniques to achieve guarantees
**Why the hype??**

- **File Sharing: Napster (+Gnutella, KaZaa, etc)**
  - Is this peer-to-peer? Hard to say.
  - Suddenly people could contribute to active global network
    » High coolness factor
  - Served a high-demand niche: online jukebox
- **Anonymity/Privacy/Anarchy: FreeNet, Publis, etc**
  - Libertarian dream of freedom from the man
    » (ISPs? Other 3-letter agencies)
  - Extremely valid concern of Censorship/Privacy
  - In search of copyright violators, RIAA challenging rights to privacy
- **Computing: The Grid**
  - Scavenge numerous free cycles of the world to do work
    - Seti@Home most visible version of this
- **Management: Businesses**
  - Businesses have discovered extreme distributed computing
  - Does P2P mean “self-configuring” from equivalent resources?
  - Bound up in “Autonomic Computing Initiative”?

---

**Utility-based Infrastructure**

- Data service provided by storage federation
- Cross-administrative domain
- Contractual Quality of Service (“someone to sue”)

**OceanStore**

- **Everyone’s Data, One Big Utility**

  - The data is just out there

- **How many files in the OceanStore?**
  - Assume 10^{10} people in world
  - Say 10,000 files/person (very conservative?)
  - So 10^{14} files in OceanStore!

  - If 1 gig files (ok, a stretch), get 1 mole of bytes!
    (or a Yotta-Byte if you are a computer person)

  Truly impressive number of elements…
  … but small relative to physical constants

  Aside: SIMS school: 1.5 Exabytes/year (1.5\times10^{18})
Key Observation: Want Automatic Maintenance

- Can’t possibly manage billions of servers by hand!
- System should automatically:
  - Adapt to failure
  - Exclude malicious elements
  - Repair itself
  - Incorporate new elements
- System should be secure and private
  - Encryption, authentication
- System should preserve data over the long term (accessible for 1000 years):
  - Geographic distribution of information
  - New servers added from time to time
  - Old servers removed from time to time
  - Everything just works

Example: Secure Object Storage

- Security: Access and Content controlled by client
  - Privacy through data encryption
  - Optional use of cryptographic hardware for revocation
  - Authenticity through hashing and active integrity checking
- Flexible self-management and optimization:
  - Performance and durability
  - Efficient sharing

OceanStore Assumptions

- Untrusted Infrastructure:
  - The OceanStore is comprised of untrusted components
  - Individual hardware has finite lifetimes
  - All data encrypted within the infrastructure
- Mostly Well-Connected:
  - Data producers and consumers are connected to a high-bandwidth network most of the time
  - Exploit multicast for quicker consistency when possible
- Promiscuous Caching:
  - Data may be cached anywhere, anytime
- Responsible Party:
  - Some organization (i.e. service provider) guarantees that your data is consistent and durable
  - Not trusted with content of data, merely its integrity

Peer-to-Peer for Data Location
Peer-to-Peer in OceanStore: DOLR
(Decentralized Object Location and Routing)

Stability under extreme circumstances
(May 2003: 1.5 TB over 4 hours)
DOLR Model generalizes to many simultaneous apps

Object Location with Tapestry DOLR

Peek at OceanStore
Mechanisms
OceanStore Data Model

- **Versioned Objects**
  - Every update generates a new version
  - Can always go back in time (Time Travel)

- **Each Version is Read-Only**
  - Can have permanent name
  - Much easier to repair

- **An Object is a signed mapping between permanent name and latest version**
  - Write access control/integrity involves managing these mappings

Comet Analogy

OceanStore API: Universal Conflict Resolution

- **Active Data:** “Floating Replicas”
  - Per object virtual server
  - Interaction with other replicas for consistency
  - May appear and disappear like bubbles

- **Archival Data:** OceanStore’s Stable Store
  - m-of-n coding: Like hologram
    - Data coded into n fragments, any m of which are sufficient to reconstruct (e.g. m=16, n=64)
    - Coding overhead is proportional to n*m (e.g. 4)
    - Other parameter, rate, is 1/overhead
  - Fragments are cryptographically self-verifying

  - Most data in the OceanStore is archival!
The Path of an OceanStore Update

- Second-Tier Caches
- Inner-Ring Servers
- Clients

Self-Organizing Soft-State Replication

- Simple algorithms for placing replicas on nodes in the interior
  - Intuition: locality properties of Tapestry help select positions for replicas
  - Tapestry helps associate parents and children to build multicast tree
- Preliminary results encouraging
- Current Investigations:
  - Game Theory
  - Thermodynamics

Archival Dissemination of Fragments

- Exploit law of large numbers for durability!
- 6 month repair, FBLPY:
  - Replication: 0.03
  - Fragmentation: 10-35
Extreme Durability?

- Exploiting Infrastructure for Repair
  - DOLR permits efficient heartbeat mechanism to notice:
    » Servers going away for a while
    » Or, going away forever!
  - Continuous sweep through data also possible
  - Erasure Code provides Flexibility in Timing
- Data transferred from physical medium to physical medium
  - No “tapes decaying in basement”
  - Information becomes fully Virtualized
- Thermodynamic Analogy: Use of Energy (supplied by servers) to Suppress Entropy

Differing Degrees of Responsibility

- Inner-ring provides quality of service
  - Handles of live data and write access control
  - Focus utility resources on this vital service
  - Compromised servers must be detected quickly
- Caching service can be provided by anyone
  - Data encrypted and self-verifying
  - Pay for service “Caching Kiosks”?
- Archival Storage and Repair
  - Read-only data: easier to authenticate and repair
  - Tradeoff redundancy for responsiveness
  - Could be provided by different companies!

Peer-to-peer Goal: Stable, large-scale systems

- State of the art:
  - Chips: $10^8$ transistors, 8 layers of metal
  - Internet: $10^9$ hosts, terabytes of bisection bandwidth
  - Societies: $10^8$ to $10^9$ people, 6-degrees of separation
- Complexity is a liability!
  - More components $\Rightarrow$ Higher failure rate
  - Chip verification $> 50\%$ of design team
  - Large societies unstable (especially when centralized)
  - Small, simple, perfect components combine to generate complex emergent behavior!
- Can complexity be a useful thing?
  - Redundancy and interaction can yield stable behavior
  - Better figure out new ways to design things...
Exploiting Numbers: Thermodynamic Analogy

- Large Systems have a variety of latent order
  - Connections between elements
  - Mathematical structure (erasure coding, etc)
  - Distributions peaked about some desired behavior
- Permits "Stability through Statistics"
  - Exploit the behavior of aggregates (redundancy)
- Subject to Entropy
  - Servers fail, attacks happen, system changes
- Requires continuous repair
  - Apply energy (i.e. through servers) to reduce entropy

Exploiting Numbers: The Biological Inspiration

- Biological Systems are built from (extremely) faulty components, yet:
  - They operate with a variety of component failures
    ⇒ Redundancy of function and representation
  - They have stable behavior ⇒ Negative feedback
  - They are self-tuning ⇒ Optimization of common case
- Introspective (Autonomic) Computing:
  - Components for performing
  - Components for monitoring and model building
  - Components for continuous adaptation

What does this really mean?

- Redundancy, Redundancy, Redundancy:
  - Many components that are roughly equivalent
  - System stabilized by consulting multiple elements
  - Voting/signature checking to exclude bad elements
  - Averaged behavior/Median behavior/First Arriving

- Passive Stabilization
  - Elements interact to self-correct each other
  - Constant resource shuffling

- Active Stabilization
  - Reevaluate and Restore good properties on wider scale
    ⇒ System-wide property validation
  - Negative feedback/chaotic attractor

- Observation and Monitoring
  - Aggregate external information to find hidden order
  - Use “to tune functional behavior and recognize dysfunctional behavior.”

Problems?

- Most people don’t know how to think about this
  - Requires new way of thinking
  - Some domains closer to thermodynamic realm than others:
    - peer-to-peer networks fit well
- Stability?
  - Positive feedback/oscillation easy to get accidentally
- Cost?
  - Power, bandwidth, storage, ...
- Correctness?
  - System behavior achieved as aggregate behavior
  - Need to design around fixed point or chaotic attractor behavior (How does one think about this)?
  - Strong properties harder to guarantee
- Bad case could be quite bad!
  - Poorly designed ⇒ Fragile to directed attacks
  - Redundancy below threshold ⇒ failure rate increases drastically
Conclusions

- **Windows vs Linux:**
  - Graphics vs Server?
  - Cathedral vs Bazaar
  - Controlled vs Free
- **Trusted Computing**
  - Hardware to allow software attestation, secure storage
- **Peer to Peer**
  - A philosophy of systems design at extreme scale
  - Probabilistic design when it is appropriate
  - New techniques aimed at unreliable components
  - A rethinking (and recasting) of distributed algorithms
  - Use of Physical, Biological, and Game-Theoretic techniques to achieve guarantees
- **Let’s give a hand to the TAs!**
  - Clap, clap, clap, clap
- **Good Bye!**
  - You guys have been great!