SOSP’15 History Day

Phil Gibbons
15-712 F15
Lecture 25

Today’s Reminders

• Today’s Office Hours will be 5-5:30 pm
  – Shortened due to conflict with CSD Faculty meeting

• No class Friday & Monday
  – Focus on projects & prep for midterm

• Midterm 2 is next Wednesday
  – Covers assigned readings from 10/28 to 11/23 + Anderson94 paper (9 papers in all)
  – Since fewer papers, a bit more in depth on each paper

• Project Presentations 12/15 & 12/16
  – Send email if want to meet r.e. projects before presentations
  – Final project report due midnight 12/16

SOSP History Day
October 4, 2015 — Monterey, California, USA

Introduction
Overview of the Day
Janak Mathews
SLIDES

The Founding of the SOSP Conferences
Jack O’Hare
SLIDES

9am - 10:30am
Perspectives on OS Foundations
Pam Dresner
ABSTRACT — SLIDES

Perspectives on Protection and Security
Harit Lampous
SLIDES

Perspectives on System Languages and Abstraction
Bence Lalik
SLIDES

11am - 12:00pm

Evaluation of File and Memory Management
Mahadev Satyanarayanan (Says)
ABSTRACT — SLIDES

Evaluation of Fault Tolerance
Ken Birman
ABSTRACT — SLIDES

1pm - 2:30pm
Virtualization
Andrew Warfield
PDF — SLIDES

Past and Future of Hardware and Architecture
Dave Patterson
ABSTRACT — SLIDES

Parallel Computing and the OS
Finn Kaunz
SLIDES

3:00pm - 4pm
The Network and the OS
Dave Clark
SLIDES

The Rise of Cloud Computing Systems
Jeff Dean
ABSTRACT — SLIDES

4pm - 5pm: Panel session
Is achieving security a hopeless quest?
Maria Schmitt, Mark Miller, David Mazières, Yuexuan Zhou
Note Concerning the Slides

• These slides were obtained by copying and pasting from the respective speakers’ original slides, and hence are copyrighted by those speakers.

• See http://www.sigops.org/sosp/sosp15/history/ for the complete set of slides for all speakers.

Perspectives on OS Foundations
Peter Denning

Number of new OS’s per decade (Wikipedia):

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9</td>
<td>40</td>
<td>50</td>
<td>96</td>
<td>73</td>
<td>245</td>
<td>320</td>
</tr>
</tbody>
</table>

Eras of Operating Systems

<table>
<thead>
<tr>
<th>Year</th>
<th>Batch</th>
<th>Interactive</th>
<th>Distributed</th>
<th>Cloud-Mobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>1950</td>
<td>One job at a time</td>
<td>Many jobs sharing</td>
<td>Personalized immersive world managing work (desktop)</td>
<td>Personalized immersive world managing life and social relations</td>
</tr>
<tr>
<td>1960</td>
<td>Batch</td>
<td>Interactive</td>
<td>Distributed</td>
<td>Cloud-Mobile</td>
</tr>
<tr>
<td>1970</td>
<td>Batch</td>
<td>Interactive</td>
<td>Distributed</td>
<td>Cloud-Mobile</td>
</tr>
<tr>
<td>1980</td>
<td>Batch</td>
<td>Interactive</td>
<td>Distributed</td>
<td>Cloud-Mobile</td>
</tr>
<tr>
<td>1990</td>
<td>Batch</td>
<td>Interactive</td>
<td>Distributed</td>
<td>Cloud-Mobile</td>
</tr>
<tr>
<td>2000</td>
<td>Batch</td>
<td>Interactive</td>
<td>Distributed</td>
<td>Cloud-Mobile</td>
</tr>
<tr>
<td>2010</td>
<td>Batch</td>
<td>Interactive</td>
<td>Distributed</td>
<td>Cloud-Mobile</td>
</tr>
</tbody>
</table>

“OS integrates with”:
- Protocol software
- IPC, RPC
- Daemon processes
- Client-server, X-windows
- Hyperlink, URL
- Browser
- Search
Eight programming support objectives added by 1965 seeded the research

Hierarchical file systems
Fault tolerant structures
Interrupt systems
Automated overlays (virtual memory)
Modular programming
Interactive programming
Controlled information sharing
Hierarchical file systems

Two Cosmic Principles Revealed in Memory Management

Locality
Location independent addressing

Emerged from virtual memory research
Originally seeking to automate overlays

Locality Principle
Initial intuitions confirmed
All computations display locality (empirical)
All computations must display locality (theory)
Harnessing locality always pays off
  caching
  parallelizing
  performance
  no-thrashing

Location Independent Addressing
Key insights:
  • Paging (U Manchester 1949)
  • Virtual v. real address (ca 1959 Kilburn and Fotheringham)
  • Segmentation in universal hierarchical address space (Dennis 1965)
From which flowed:
  • Dynamic mapping virtual to real via page table
  • Demand paging
  • Replacement algorithms
  • MMU and TLB mapping architecture
  • Hierarchical naming systems
Huge benefits:
- Location independence
- Logical partitioning (address space isolation)
- Artificial contiguity
- Relocation
- Distributed naming authorities

Evolved into global, all-time unique addresses for digital objects anywhere in a system.
Hierarchical Internet URLs and domain names.

Now Internet is a huge virtual address space of capabilities, URLs, and DOIs with mapping via DNS and handle-servers.

Perspectives on Security
Butler Lampson

How did we get here?
- In the beginning, security was by physical isolation (1950-1963)
  - Easy: You bring your data, control the machine, take everything away
  - Still do this today with VMs and crypto (+ enclaves if VM host is untrusted)
- Timesharing brought the basic dilemma of security: (1963-1982)
  - Isolation vs. sharing
    - Hard: Each user wants a private machine, isolated from others
      but users want to share data, programs and resources
- Since then, things have steadily gotten worse (1982-2015)
  - Less isolation, more sharing, no central management
  - More valuable stuff in the computers
  - Continued misguided search for perfection (following the NSA’s lead)

Wisdom
- If you want security, you must be prepared for inconvenience.
  —General B.W. Chidlaw, 12 December 1954
- When it comes to security, a change is unlikely to be an improvement.
  —Doug McIlroy, 1968
- The price of reliability is the pursuit of the utmost simplicity.
  It is a price which the very rich find most hard to pay.
  —Tony Hoare, 1980 (cf. Matthew 19:24)
- But who will watch the watchers? She’ll begin with them and buy their silence.
  —Juvenal, sixth satire, ~100

What we know how to do
- Secure something simple very well
- Protect complexity by isolation and sanitization
- Stage security theatre

What we don’t know how to do
- Make something complex secure
- Make something big secure if it’s not isolated
- Keep something secure when it changes
- Get users to make judgments about security
- Understand privacy—fortunately not an SOSP topic
**Themes**

- **Goals:** Secrecy (confidentiality), integrity, availability  
  (CIA: Ware 1970)
- **Gold standard:** Authentication, authorization, auditing  
  (S&S 1975)
- **Principals:** People, machines, programs, ...  
  (Dennis 1966, DEC 1991)
- **Groups/roles:** make policy manageable  
  (Multics 1968, NIST 1992)

**Oppositions**

- **Winner** vs. **Loser**  
  *(in deployment, not good vs. bad)*
- Convenience vs. Security
- Sharing vs. Isolation
- Bug fixes vs. Correctness
- Policy/mechanisms vs. Assurance
- Access control vs. Information flow

**Timeline**

<table>
<thead>
<tr>
<th>Year</th>
<th>Themes</th>
<th>Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>1960s</td>
<td><strong>Timesharing:</strong> ACLs; access control matrix; VMs; passwords; capabilities; domains; gates</td>
<td>CTSS; Multics; CP/CMS; Cal TSS; Adept-50; Plessey 250</td>
</tr>
<tr>
<td>1970s</td>
<td><strong>TS:</strong> LANs; Internet (e.g. security); public key; multi-level sec.; ADTs; objects; least privilege; Trojans; isolation by crypto; amplification; undetectability</td>
<td>Unix; VMS; VM/370; IBM RACF; Clu; Hydra; Cambridge CAP</td>
</tr>
<tr>
<td>1980s</td>
<td><strong>Workstations:</strong> client/server; Orange Book; global authentication; Clark and Wilson</td>
<td>A1 VMS; SecureID; Morris worm; IX</td>
</tr>
<tr>
<td>1990s</td>
<td><strong>PCs:</strong> Web; sandboxes; Java security; crypto export; decentralized information flow; Common Criteria; biometrics; RBAC; BAN; SEI; SET</td>
<td>Browsers; SSL; NT; Linux; PGP; Taos</td>
</tr>
<tr>
<td>2000s</td>
<td><strong>Web:</strong> JavaScript, buffer overflows; DDoS</td>
<td>TPM; LSM; SELinux; seL4; HiStar</td>
</tr>
<tr>
<td>2010s</td>
<td><strong>Web:</strong> big data, enclaves, homomorphic crypto</td>
<td>Singularity; CryptDB, Ironclad</td>
</tr>
</tbody>
</table>

**Does it actually work? Assurance (Correctness)**

- Keep it simple—Trusted Computing Base (TCB)  
  *(Rushby 1981)*
  - One way is a security kernel: apps are not in the TCB. Works for sharing hardware
- Ideally, you **verify**; prove that a system satisfies its security spec
  - This means that every behavior of the system is allowed by the spec
  - Not the same as proving that it does everything in the manual
  - Today in seL4, Ironclad, ... First tried in Gypsy  
    *(late 1970s)*
  - What if the spec is wrong? Keep it simple
- Usually verifying is too hard, so you **certify** instead
  - Through some “independent” agency. Alas, process trumps substance
    - First by DoD for Orange Book, later international Common Criteria  
      *(1985, 1999)*
- Or you can verify some properties: isolation, memory/type safety
- Or you can apply band aids

**Band-aids for Bugs (Defense in Depth)**

- No guarantees, but at least the bad guy has to work harder  
  *(DEC 1988)*
  - Firewalls to keep intruders out, look for suspicious traffic  
  *(~1990)*
  - **Signature** hacks to detect malware  
  *(Phrack 1996)*
  - **Intrusion detection** hacks to look for anomalous behavior  
  *(SRI 1986)*
  - **Control Flow Integrity** to block jumps not in the normal flow  
  *(MSR 2005)*
  - **Taint tracking** to keep unsanitized input away from execution  
  *(CMU 2005)*
  - **Process to enforce use of the tools**  
  *(MS SDL 2004)*

- “I don’t have to outrun the bear; I just have to outrun you.”
- These are not bad things, but they are hacks
What has worked? What hasn’t?

Worked – gotten wide adoption
- VMs
- SSL
- Passwords
- Safe languages
- Firewalls
- Process—SDL

Failed
- “Secure systems”
- Capabilities (except short term)
- Metrics for security
- MLS/Orange book
- User education
- Intrusion detection

Why don’t we have “real” security?

- A. People don’t buy it
  - Danger is small, so it’s OK to buy features instead
  - Security is expensive
    - Configuring security is a lot of work
    - Secure systems do less because they’re older
  - Security is a pain
    - It stops you from doing things
    - Users have to authenticate themselves
- B. Systems are complicated, so they have bugs
  - Especially the configuration

What next?

- Lower aspirations. In the real world, good security is a bank vault
  - Hardly any computer systems have anything like this
  - We only know how to make simple things secure
- Access control doesn’t work—40 years of experience says so
  - Basic problem: its job is to say “No”
    - This stops people from doing their work, and then they relax the access control
    - Usually too much, but no one notices until there’s a disaster
- Retroactive security: focus on things that actually happened
  - Rather than all the many things that might happen
- Real world security is retroactive
  - Burglars are stopped by fear of jail, not by locks
  - The financial system’s security depends on *undo*, not on *vaults*

Memory and File Systems
M. Satyanarayanan

Four Drivers of Progress

- The quest for **scale** from early 1950s
- The quest for **speed** from early 1950s
- The quest for **transparency** from early-1960s
- The quest for **robustness** (both system and human errors) from mid- to late-1960s

Complex Interactions
Cost of Memory & Storage
(Source: John C. MacCallum http://jcmil.com)

Memory Prices ($ / MB)
- 13 orders of magnitude since 1955

Naming and Addressability
Consistently too few bits in addressing (12-bit, 16-bit, 18-bit, 32-bit, ...)
- re-learned in DOS/Win1.1 (memory extenders); hopefully 64 bits will last us a while

Semantic addressing
- hierarchical name spaces, SQL, search engines

Content Addressable Storage (aka deduplication)
- Venti (late 1980s), LBFS (early 2000s), many others since,
- continuing concerns regarding collisions (Vai Henson)

Capability-based
- short term (seconds, minutes, hours lifetime)
  - can be viewed as a form of caching expensive/verbose access checks
- long term (infinite life)
  - Hydra on C.mmp (mid 1970s) pushed this concept to the limit
  - Intel IA64 432 (3 papers in SOSP 1981)

Transparency
- "Indistinguishable from original abstraction"
  - no application changes: programs behave as expected
  - no unpleasant surprises for users: good user experience
  - importance increases as hardware to human cost ratio shifts

Hugely important in industry, less important in academic research

Achieved by interposing new functionality at widely-used interfaces
- memory abstraction (hardware caches)
- POSIX distributed file systems
- x86 virtual machines

A Brief History of Caching
Demand paging was first known use of caching idea (1961)

Dynamic Storage Allocation in the Atlas Computer,
Including an Automatic Use of a Backing Store
John K. Furth
Harvard Univ., Cambridge, Mass.

Hardware caches (1968)
- "Structural Aspects of the System/360 Model 30L, Part II: The Caches."

Distributed file systems (1983)
- AFIS, NFS, Sprite, Codas"
The Importance of Demand Fetch

Assumes ability to detect read operations
- ability to detect cache misses
- ability to interpose cache logic
- result is total transparency

In a file system this requires OS support
- distributed file systems (e.g. AFS, Coda, ...)
- Fuse interface

Systems like Dropbox cannot do this
- lack of OS support simplifies implementation
- improves portability of code across OSes
- Dropbox needs complete replicas everywhere
  (aka "sync solution")

Without OS intercept

1. Even viewing one small file requires whole replica
2. Every update has to be propagated everywhere

Coping With Human Error

Use of separate address spaces (threads vs. processes)

Easy retrospection of file systems by users
- periodic read-only snapshots (AFS)
- Apple Time Machine, Elephant File System, ...

Why is memory distinct from file system?

Single level stores have been proposed in the past
- but separation offers enhanced robustness
- well-formed open / read / write / close unlikely to be accidental
- contrast with wild memory write

Are Classic File Systems Dead?

Hot Topic Today

The death watch has begun

Hierarchical file systems are dead

Every Page is Page One

Ken Thompson made radical changes in creating Unix
- why was the Unix file system so conventional and hierarchical?
- mere sentiment? lack of imagination?

"The Architecture of Complexity"

"Empirically, a large proportion of the complex systems we observe in nature exhibit hierarchic structure. On theoretical grounds we could expect complex systems to be hierarchies in a world in which complexity had to evolve from simplicity. In their dynamics, hierarchies have a property, near-decomposability, that greatly simplifies their behavior."
How Hierarchy Helps

Hierarchical file systems conflate search and access
- well-matched to limitations of human cognition,
- locality is an emergent property (temporal and spatial)
- locality is precious performance-wise for direct human exploration of data

Retrospective use of old unstructured data (e.g., decades later) →
- even the features for indexing may be unclear
- manual exploration may be necessary

Need for manual exploration (even if rare) →
- hierarchical file systems will not disappear
- but the hierarchical nature may remain deeply buried

Fault Tolerance

Ken Birman

Too many seminal concepts
- Process pairs, primary-backup
- 2PC and 3PC, Quorums
- Atomic Transactions
- State machine replication
- RAID storage solutions
- Checkpoints, Message Logging
- Byzantine Agreement
- Gossip protocols
- Virtual synchrony model
- Paxos
- Zookeeper

Theory ...

Skepticism
- Consensus
- 0W: consensus
- FLP + oracle
- CATOCS
- CAP

Principles from the theory side...
- FLP: Protocols strong enough to solve asynchronous consensus cannot guarantee liveness (progress under all conditions).
- If running a highly available database with network partition, conflicting transactions induce inconsistencies (CAP theorem).
- Need 3f+1 replicas to overcome Byzantine faults
Principles from the systems side...

- Make core elements as simple as possible
  - *Pare down, optimize the critical path*
  - Captures something fundamental about systems.

- Generalized End-to-End argument:
  - Let the application layer pick its own models.
  - Limit core systems to fast, flexible building blocks.

---

Tensions

- Why aren’t *existing* OS mechanisms adequate?
- Is fault-tolerance / consistency too complex or costly?
- Do the needed mechanisms enable or impose models?

---

Existing core OS support: Inadequate

- IP multicast just doesn’t work...
  - Amazon AWS disables IPMC and tunnels over TCP

- TCP is the main option, but it has some issues:
  - No support for reliable transfer to multiple receivers
  - Uncoordinated model for breaking connections on failure
  - Byte stream model is mismatched to RDMA

---

... Real systems informed by sound theory

- Isis: Widely adopted during the 1995-2005 period
  - French ATC system, US Navy AEGIS, NYSE...

- Paxos: Very wide uptake 2005-now
  - Locking, file replication, HA databases...
  - Clean methodology and theory appeal to designers
  - Corfu is the purest Paxos solution: robust logging
... Cloud rebellion: “Just say no!”

- State Machine Replication, Paxos, ACID transactions
- Chubby, Zookeeper, Corfu
- Primary + Warm backup... Chain Replication
- Dynamo: Eventual consistency (BASE), NoSQL KVS

Is consistency just too costly?

- CAP: Two of {Consistency, Availability, Partition-Tolerance}
  - Widely cited by systems that cache or replicate data
  - Relaxed consistency eliminates blocking on the critical path
  - CAP theorem: proved for a WAN partition of an H/A database
- BASE (eBay, Amazon)
  - Start with a transactional design, but then weaken atomicity
  - Eventually sense inconsistencies and repair them

... but does CAP+BASE work?

- CAP folk theorem: “don’t even try to achieve consistency.”
- CAP + BASE are successful for a reason:
  - In the applications that dominate today’s cloud, stale cache reads have negative utility but don’t cause safety violations.
  - In effect a redefinition, not a rejection, of consistency

Parallelism and Operating Systems

Frans Kaashoek

Three types of parallelism in operating systems

1. User parallelism
   - Users working concurrently with computer
2. I/O concurrency
   - Overlap computation with I/O to keep a processor busy
3. Multiprocessors parallelism
   - Exploit several processors to speedup tasks

The first two may involve only 1 processor
The Rise of Cloud Computing Systems

Jeff Dean

- network rewiring (rolling ~5% of machines down over 2-day span)
- 20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
- 5 racks go wonky (40-80 machines see 50% packetloss)
- 8 network maintenances (4 might cause ~30-min random connectivity losses)
- 12 router reloads (takes out DNS and external vips for a couple minutes)
- 3 router failures (have to immediately pull traffic for an hour)
- dozens of minor 30-second blips for DNS
- 1000 individual machine failures
- thousands of hard drive failures
- slow disks, bad memory, misconfigured machines, flaky machines, etc.

Reliability Must Come From Software
A Series of Steps, All With Common Theme:

Provide Higher-Level View Than “Large Collection of Individual Machines”

Self-manage and self-repair as much as possible

Google File System (Ghemawat, Gobioff, & Leung, SOSP'03)

- Centralized master manages metadata
- 1000s of clients read/write directly to/from 1000s of disk serving processes
- Files chunks of 64 MB, each replicated on 3 different servers
- High fault tolerance + automatic recovery, high availability

Successful design pattern:

Centralized master for metadata/control, with thousands of workers and thousands of clients

Many Applications Need To Update Structured State With Low-Latency and Large Scale

Desires:

- Spread across many machines, grow and shrink automatically
- Handle machine failures quickly and transparently
- Often prefer low latency and high performance over consistency
Distributed Semi-Structured Storage Systems

- BigTable [Google: Chang et al. OSDI 2006]
  - higher-level storage system built on top of distributed file system (GFS)
  - data model: rows, columns, timestamps
  - no cross-row consistency guarantees
  - state managed in small pieces (tablets)
  - recovery fast (10s or 100s of machines each recover state of one tablet)

- Dynamo [Amazon: DeCandia et al., 2007]
  - versioning + app-assisted conflict resolution

- Spanner [Google: Corbett et al., 2012]
  - wide-area distribution, supports both strong and weak consistency

Successful design pattern:

Give each machine hundreds or thousands of units of work or state

Helps with:
- dynamic capacity sizing
- load balancing
- faster failure recovery