Mobile Operating Systems and Course Conclusion

Sam Kumar

CS 162: Operating Systems and System Programming

Lecture 28

https://inst.eecs.berkeley.edu/~cs162/su20

Read: None
Recall: More Robust Agreement...

• Is there protocol for distributed consensus that can make progress without depending on any *particular node* being available?

• Yes. **Paxos** and **Raft** are consensus algorithms that can make progress as long as a *majority* of nodes are available.
Recall: Byzantine Generals’ Problem

• Byzantine General’s Problem (n players):
  • One General and n-1 Lieutenants
  • Some number of these (f) want chaos

• Want the following Integrity Constraints to apply:
  • IC1: All loyal lieutenants obey the same order
  • IC2: If the commanding general is loyal, then all loyal lieutenants obey the order he sends
Recall: CAP Theorem

• Originally proposed by Eric Brewer (Berkeley)

1. **Consistency** – changes appear to everyone in same order (linearizability)
2. **Availability** – *any* node in the system can process requests
3. **Partition Tolerance** – system continues to work even when one part of network can't communicate with the other

• Impossible to achieve all 3 at the same time (pick two)
Recall: Performance Isolation (cgroups)

- Idea: provide greater performance isolation between cgroups than between processes...
Docker

Features on top of OS cgroups:

• Images
  • Describes the starting state of a Docker container (like a snapshot that can be restarted)

• Union file system
  • Image describes file system as a sequence of layers, each with some files
  • Overall file system is the union of the layers
  • Layers can be reused across images/containers

• Tools and observability

• Container lifecycle management
Recall: Advantages of Microservices

• Services can be developed independently
  • Only RPC interface matters, not implementation
  • Can even use third party applications

• Horizontal scalability
  • Each microservice can be scaled separately from the rest

• Fault tolerance
  • Failure of one replica of a service should not affect other replicas
  • Failure of an entire service should (ideally) not bring down the entire pipeline
Secure Channels

1. Each party encrypts everything that’s sent across the network with recipient’s public key (recipient decrypts with secret key)
2. Each party signs everything that’s sent across the network with their secret key (recipient verifies with sender’s public key)

Can be accelerated by agreeing on symmetric keys this way, and then using symmetric keys for the actual data...
Recall: Digital Certificate

• How do you know $K_A$ is amazon.com’s public key?

• Because some authority that you trust (e.g., Verisign) has signed a statement that $K_A$ is amazon.com’s public key

• That statement, together with the signature, is called a **certificate**

• amazon.com presents this certificate for incoming connections, so that the user knows they’re really speaking to amazon.com
Recall: Security in Computer Systems

• Security is needed at **all** levels

• From Networking (secure channels)...

• ... all the way down to Architecture
Recall: Linux Virtual Memory Map

32-Bit Virtual Address Space

- Kernel Addresses: 0x00000000 - 0xFFFFFFFF
- User Addresses: 0xC0000000 - 0x0000000000000000

64-Bit Virtual Address Space

- Kernel Addresses: 0xFFFFFFFFFFFFFFFF - 0xCCCCCCCCCCCCCCCC00000000
- User Addresses: 0x0000000000000000 - 0xFFFF800000000000

“Canonical Hole”

3GB Total

1GB Physical

128TiB

896MB Physical

32-Bit Virtual Address Space

128TiB

64 TiB Physical

0x00007FFFFFFF

0x0000000000000000
Recall: User $\rightarrow$ Kernel

Process Virtual Address Space

Kernel space

Pages:
- 0xffffffff
- 0xc0000000

Processor registers:
- ip
- sp

CPL: 3 - user

Stack:
- argv
- stack

Heap:
- user data

User code:
- user code

Page Table:
- u/s

Physical Memory

8/12/2020
Recall: User → Kernel

Process Virtual Address Space

Processor registers

CPL: 0 - sys

Kernel

user code

user data

heap

argv

stack

Page Table

Physical Memory
Meltdown Flaw (2018, Intel x86, IBM, ARM)

- Exploit speculative execution to observe contents of kernel memory

```
1: // Set up side channel (array flushed from cache)
2: uchar array[256 * 4096];
3: flush(array); // Make sure array flushed from cache

4: // Set up signal handler to catch and ignore SIGSEGV
5: uchar result = *(uchar *)kernel_address;
6: uchar dummy = array[result * 4096];

7: // scan through 256 array slots to determine which loaded

- Value detected by the fact that one cache line is loaded
  - As we scan through the array, that cache line will be faster to access
- Fix: better hardware (hopefully coming soon)
```
Patch: Kernel Page Table Isolation

• Requires TLB flushes (potentially high overhead on some workloads)!
• Can use PCIDs if hardware supports them
• Of course, a much better fix would be hardware without this vulnerability...
Recall: Making Address Translation Fast

- Cache results of recent translations
  - Separate from memory cache
  - Cache PTEs using Virtual Page Number as the key

![Diagram of processor, MMU, and cache connections]

- Processor (core)
- MMU
- Cache(s)
- Physical Memory

\[
\begin{align*}
V_{Pg} M_1 &: \langle \text{Phs_Frame } #_1, V, \ldots \rangle \\
V_{Pg} M_2 &: \langle \text{Phs_Frame } #_2, V, \ldots \rangle \\
V_{Pg} M_k &: \langle \text{Phs_Frame } #_k, V, \ldots \rangle
\end{align*}
\]
Recall: The Big Picture

- CPU
- TLB
- Cache
- Memory

MMU traverses page table on miss

Page Fault trap on translation failure
Virtual Caches?

- Why not cache data for virtual addresses instead of physical addresses?
  - Cache end-to-end memory lookup

- Potential disadvantages:
  - Need to also cache permission bits (for permission checks)
  - Need to flush cache where we previously flushed the TLB
  - Hard to deal with aliased pages
    - Either put antialiasing support in hardware...
    - Or make the OS ensure that aliased pages always conflict in the cache...
    - Or a combination of both? (simpler antialiasing support because the OS cooperates)
A Closer Look at a Memory Access

Can we look data up in the cache first?

Virtual Address: V page no. offset

TLB Lookup:
- V Access Rights PA

P page no. offset

Physical Address:
- tag index offset

Cache Way #1:
- V Tag Data

Cache Way #2:
- V Tag Data

Cmp
Recall: Set-Associative Cache

- **N-way set associative**: N entries per Cache Index
  - N direct mapped caches operates in parallel

Example: Two-way set associative cache
- Cache Index selects a “set” from the cache
- Two tags in the set are compared to input in parallel
- Data is selected based on the tag result
Overlapping TLB Lookup and Cache Lookup

• Main idea:
  • Offset in virtual address covers the “cache index” and “cache offset”
  • Can determine cache index before translating the address
  • Thus, can select the cached byte(s) in parallel to perform address translation

<table>
<thead>
<tr>
<th>virtual address</th>
<th>Virtual Page #</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>physical address</td>
<td>tag / page #</td>
<td>index</td>
</tr>
</tbody>
</table>
Overlapping TLB Lookup and Cache Lookup

• Here’s how this might work for a 4 KiB, direct-mapped cache
Overlapping TLB Lookup and Cache Lookup

• When does this work?
  • Cache index/offset fit inside page offset
  • Each way of the cache is no bigger than a page
  • Direct-mapped cache cannot exceed page size!

• E.g., Intel Core i7-7820HQ (the processor in my laptop)
  • L1 i/d caches: 32 KiB, 8-way, 64 B blocks → 64 sets, 6-bit index/6-bit offset
  • Can overlap TLB and cache lookups in the L1 caches!

• What if you want a large cache?
  • Increase page size
  • Increase cache associativity
  • Look up in multiple cache indices (!)
What About Larger Caches?

• Increase page size (so offset has more bits)
• Increase associativity (so cache index has fewer bits)
• Look up in multiple cache indices, pick with TLB lookup finished (!)

• Virtual Indexed, Physical Tagged (VIPT) cache
  • Cache index lookup can happen in parallel
  • May need to include additional tag bits based on extent of overlap
Announcements

• Project 3 code due on Friday

• Final Exam on Friday

• Homework 6 (optional) due on Sunday
Mobile Operating Systems
Role of Laptop/Desktop Operating System

- User decides which applications to run and starts them
- User starts an application before attempting to interact with it
- User switches between applications explicitly (e.g., multiple windows open on a screen)

- Operating system takes a passive role
  - Performs service for the application when it issues a system call
  - Allows it to communicate with other running processes
  - Keeps background processes/services running (e.g., systemd services)
Android: A More Cooperative Ecosystem

“The mobile-app experience differs from its desktop counterpart in that a user's interaction with the app doesn't always begin in the same place. Instead, the user journey often begins non-deterministically.”


• Example: social media app allows you to compose an email
• Example: social media app allows you to take a pictures
• Apps request services from and build on the functionality in others...
Components of an Android App

- Not just exec `main()` and syscall ...
- Lots of entry points (call backs)
- Process started “whenever needed”
Security Model in Laptop/Desktop OS

• Potentially multiple users
  • They don’t trust each other
  • Give permission to files, etc. based on which user/group they are

• Applications are trusted with users’ privileges
  • OS protection prevents one application from overwriting others, but does not protect I/O resources (e.g., files)
  • If a user runs an application, it (by default) inherits all of that users’ permissions
Mobile OS: Want Stronger Isolation!

• Users may install malicious apps
  • Want to maintain system stability (shared goal with traditional OS)
  • Want to protect users’ data—requires *sandboxing* applications

• Even legitimate applications do questionable things
  • Huge incentive to track users and see their data
Attempting to Circumvent Permission...

• 2019 study: many apps (even legitimate ones) attempt to circumvent the permission system, or use SDKs that do

• They use side channels or covert channels
  • E.g., “companies getting the MAC address of the connected WiFi base stations from the ARP cache”
  • E.g., “third-party libraries provided by ... Baidu and Salmonads ... independently make use of the SD card as a covert channel, so that when an app can read the phone’s IMEI, it stores it for other apps that cannot”

So what do we want in a Mobile OS?

- Greater integration
  - To support a more cooperative ecosystem
  - Where apps providing service to each other is the common case

- Greater isolation
  - To have a stronger sandbox
  - Restrict apps from interacting in more complex ways

- *Aren’t these goals somewhat contradictory???
Isolation and Integration in Android

- **Rough equivalence: App = User = Process**

- Each Android App is assigned a distinct Linux UID (user ID)
  - Separate permission per app
  - By default, only the app can access its own files
  - Stronger “sandbox” for each app, stronger isolation between apps

- Operating system provides richer forms of IPC
  - Ability to send messages to *apps*, even if they aren’t running
Android Architecture

• Linux kernel (modified) provides proc/mem management, filesys, and drivers

• App is made of Components. Four types:
  • Activities: interact with user
  • Services: runs in the background, no UI
  • Content Providers: provides controlled data access
  • Broadcast Receivers: react to system events

• “Binder” IPC is the key

• Any app can start another app’s component
Extending the OS without new syscalls?

- Drivers – add modules to kernel that communicate with user process through file descriptor operations.
When Does a Component Run?

• Whenever it needs to...

“When the system starts a component, it starts the process for that app if it's not already running and instantiates the classes needed for the component.”

—https://developer.android.com/guide/components/fundamentals

• How does an App activate a component in another App?
• Deliver a message to the system that conveys intent to start the particular component
  • The system intervenes in all App-to-App interactions
  • IPC !!!
Intents: An Active Message

• A message that activates a component
  • Activities & Services: it describes the action to perform
  • Broadcast Receivers: simply an announcement

• How is the recipient of the intent identified?
  • What ”binds” the two participants?
  • Explicit Intent specifies the component name
  • Implicit Intent specifies the action, can be invoked on any App that can perform that action

• Content Providers are not activated by intents
  • They are a more passive entity
  • Activated by a request from a “Content Resolver”
    • Pulled, not pushed
Binder

- AIDL (Android Interface Definition Language) defines interface with method of remote services
  - Parser generate Java class (Client Proxy and Service Stub)
- Java API wrapper
  - Access to Binder
Android: Major Changes to Linux

- Binder – Android Specific IPC / RPC mechanism
  - THE key to the entire application framework
- ashmem – Anonymous (or Android) Shared Memory
  - driver-based augment mmap (replace shmat) for small memory & extensive process sharing
- PMEM – process memory allocator
  - Large physically contiguous memory blocks mapped into user space
- logger – system logging facility
- wakelocks
  - User-controlled means of controlling whether system enters low-power states and what wakes it up
- Out-of-memory handling
  - Kill processes under low memory conditions
- Paranoid networking – restrict I/O to certain proc’s
- Alarm timers, Timed gpio from user space
- yaffs2 flash file system, ...
Course Conclusion
Recall: Most Transformative Artifact of Human Civilization...

Map of the Internet in 1999
Recall: Running Systems at Internet Scale
Recall: Across Incredible Diversity

Bell’s Law: New computer class every 10 years

- Mainframe
- Mini
- Workstation
- PC
- Laptop
- PDA
- Cell
- Mote!

Number crunching, Data Storage, Massive Inet Services, ML, ...

Productivity, Interactive

Streaming from/to the physical world

The Internet of Things!
Recall: And Range of Timescales

Jeff Dean: “Numbers Everyone Should Know”

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5</td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5</td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>7</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>25</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100</td>
</tr>
<tr>
<td>Compress 1K bytes with Zippy</td>
<td>3,000</td>
</tr>
<tr>
<td>Send 2K bytes over 1 Gbps network</td>
<td>20,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
<td>500,000</td>
</tr>
<tr>
<td>Disk seek</td>
<td>10,000,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>20,000,000</td>
</tr>
<tr>
<td>Send packet CA-&gt;Netherlands-&gt;CA</td>
<td>150,000,000</td>
</tr>
</tbody>
</table>
Operating Systems at the heart of it all...

• Make the incredible advance in the underlying technology available to a rapidly evolving body of applications
  • Provide **consistent abstractions** to applications, even on different hardware
  • Manage **sharing of resources** among multiple applications

• The key building blocks:
  • Processes
  • Threads, Concurrency, Scheduling, Coordination
  • Address Spaces
  • Protection, Isolation, Sharing, Security
  • Communication, Protocols
  • Persistent storage, transactions, consistency, resilience
  • Interfaces to all devices
It’s been almost 8 weeks since then!

• 6 Homeworks
• 3 Projects
• ~7,500 autograder builds
• ~1,800 slides

What do you now understand about operating systems?
Recall: Syllabus, in a Nutshell

• OS Concepts and Abstractions (2 weeks)
  • Process, I/O, Networks, etc.
• Concurrency (2 weeks)
  • Threads, scheduling, synchronization, scalability, fairness
• Address Spaces (1 week)
  • Virtual memory, paging, address translation, protection, sharing
• File Systems (1 week)
  • I/O devices, files, storage, naming, caching, performance, transactions
• Distributed Systems (1 week)
  • Protocols, RPC, NFS, DHTs, Consistency, Scalability, Fault Tolerance
• Special Topics (1 week)
Recall: CS 162 is a class about...

• The key systems abstractions that have emerged over time,
  • Processes, Threads, Events, Address Spaces, File Systems, Sockets, Transactions, Key-Value Stores, etc.

• The tradeoffs surrounding their design,

• Their efficient implementation,
  • Including the hardware support that makes them possible and practical

• And how to use them effectively.
You have a better understanding of how the software on your computer REALLY works...

AN x64 PROCESSOR IS SCREAMING ALONG AT BILLIONS OF CYCLES PER SECOND TO RUN THE XNU KERNEL, WHICH IS FRANTICALLY WORKING THROUGH ALL THE POSIX-SPECIFIED ABSTRACTION TO CREATE THE DARWIN SYSTEM UNDERLYING OS X, WHICH IN TURN IS STRAINING ITSELF TO RUN FIREFOX AND ITS GECKO RENDERER, WHICH CREATES A FLASH OBJECT WHICH /></pre> <pre>BECAUSE I WANTED TO SEE A CAT JUMP INTO A BOX AND FALL OVER. 

I AM A GOD.
And you got a taste of what it’s like to actually build systems...

Operating **System**

What makes something a **system**?

- Multiple interrelated parts
  - Each potentially interacts with the others
- Robustness requires an **engineering mindset**
  - Meticulous error handling, defending against malicious careless users
  - Treating the computer as a concrete machine, with all of its limitations and possible failure cases

**System programming is an important part of this class!**
Theme: Management of Computer Resources

• Ties the class together
  • Managing CPU = Synchronization/Scheduling
  • Managing Memory = Address Spaces
  • Managing Storage = File System
  • Managing Multiple Machines = Distributed Systems
Example: What’s in a search query?

- Complex interaction of multiple components in multiple administrative domains
  - Systems, services, protocols, ...

- DNS request
- DNS Servers
- Internet
- Datacenter
  - Load balancer
  - Search Index
  - Page store
  - Ad Server
  - Create result page
What are some directions the field is moving in?

• Kernels in high-level languages

• Datacenter networking and management

• Support for high-performance I/O (especially networking)

• IoT: Bringing intelligence into the physical world

• Specializing resource management for important tasks (e.g., machine learning)...

• And many other directions (this list is nowhere near complete...)

8/12/2020
Where to Go Next?

• Architecture (CS 152)

• Security (CS 161)

• Networking (CS 168)

• Databases (CS 186)

• Distributed Systems
  • No class in the department at the undergrad level
  • CS 262A at the graduate level
Why take CS 162? Why learn about OS?

• Some of you will design and build parts of operating systems

• Many of you will create systems that use OS concepts
  • Whether you build hardware or software
  • The concepts and design patterns appear at many levels

• All of you will write programs that use OS abstractions
  • The better you understand their design and implementation, the better use you’ll make of them