15-410

"...This is a transformative class..."

Review
Dec. 9, 2016

Dave Eckhardt

Dave O'Hallaron

Synchronization

Time is running out

Synchronization

Time is running out

- https://cmu.SmartEvals.com

Synchronization

Time is running out

- https://cmu.SmartEvals.com
- By the way, the 605 students are responding at a higher rate so far than the 410 students! But the contest isn't over yet!

Synchronization

The end is nigh

Synchronization

The end is nigh

- But you might consider 15-412!
  - http://www.cs.cmu.edu/~412
  - Questions? Eckhardt
- Consider for S'17...
  - 15-418 (Parallel Architecture)
  - 15-440 (Distributed)
  - 18-845 (apply via Dave O'Hallaron)
  - 15-721 (Database Systems) – the hard-core version
- F'17...
  - 15-411 (Compilers) is pretty transformative too
  - 15-418 (Parallel Architecture)
  - 15-712 (“Grad OS/DS”)
  - 15/18-487 (Intro to Computer Security)
  - 15/18-746 (Advanced Storage Systems)
  - We also have an intro database class, 15-415

Synchronization

Homework 2 due tonight

- Solutions will be released “immediately”
- ⇒ No late days
- ~ 1/4 have already turned in all or part (good)
- Don't forget to try storing files in your hand-in directory *before* the deadline!

Book report due tonight

Exam some time in the next month

- http://www.cmu.edu/hub/docs/final-exams.pdf

Synchronization

P3 grading guidance reminder

- Weights are approximate

~50% shell works (w/o horrible hacks)

~30% tests

- 10% basic
- 15% stress, trickiness, argument verification
- 5% threads

~10% structure/style

~10% concurrency (preemption, locking, synch)

Synchronization

Exam will be closed-book

- *But you may bring a 1-sided 8.5x11 sheet of notes*
  - WRITTEN BY YOU IN YOUR OWN HAND
  - WRITTEN BY YOU IN YOUR OWN HAND
- *Weakly* non-cumulative
  - Emphasis on new material, design questions
  - You will need to use some “old” knowledge
  - We didn't really test on “P2 knowledge” (nor P3)
  - Recall: VM was off-limits on mid-term (and you've done it)
- Mixture of fact/concept testing and *design*

Synchronization

About today's “review”

- **More “reminders” than “course outline”**
  - **Un-mentioned topic implies “text & lectures straightforward”**
- **Reading *some* of the textbook is advisable!**

Read Your Code

Re-read your P2

Re-read your P3

Go over feedback

Talk about code with your partner

- Schedule a time

You should understand "the hard parts"

- Focus on whichever part you know least well
  - (or fear the most)

“Concept” Lectures

We *could* ask a question about these topics...

- **...we would give you guidance/refresh your memory**

Examples

- **Windows device-driver architecture**
- **Lock-free programming**
  - **We are not likely to ask you to prove a lock-free splay treap**
  - **Though as an educated person in 2016, you *should* understand the motivation and basic approach**

Core “Phase I” concepts

Machine model

- Registers
  - “regular”
  - “special”
- Interrupt (vs. exception – how they differ, why)

Process model

- You should be a memory-map *expert*
  - Kernel space, user space, virtual memory
- Process vs. thread
- *Exactly* what goes on a stack, where it comes from...

Core “Phase I” concepts

Mutual exclusion

- mutex, cvar, what's inside, why

Concurrency

- Race-condition expert!
- Be able to explain one to your nephew
  - (the one you'll visit over break)

Deadlock

- Ingredients
- Various approaches to coping

Virtual Memory

The Game

- Maintain multiple illusions (aka “address spaces”)

Players

- High-level info (what uses which regions, COW/ZFOD)
- Mapping data structure (typically set by processor)
- TLB – cache of v-to-p translations from that data structure
  - “flush” - when, why, how?

Behavior of the Players

- Mappings are *sparse*
- This explains the ways they're implemented

Thread Scheduling

Round-Robin

Things people do

- Multi-level feedback queues

Be careful!

- Priority

Non-volatile Storage

Model for non-volatile storage

- Blocks as atomic unit of fetch/store
- *Much* slower than RAM
- Requests can complete in non-FIFO order

"Disk"

- Spinning platter/waving arm model
- C/H/S (obsolete – now: "logical block addressing", LBA)

"SSD"

- Various technologies exist
- "NAND flash"
  - Significant features
  - Challenges: Wear leveling, write amplification
    - » How to address them

File Systems

Data access model

- What it means for a file to be “open”

Cache issues

Naming

- Directory flavors, mounting

Core problem: block mapping

- Compare data structures to VM
- “Holes”

Architecture

- Layering to support multiple file system types, ...

IPC

Communicating process on one machine

Naming

- Name server?
- File system?

Message structure

- Sender id, priority, type
- Capabilities: memory region, IPC rights

Synchronization/queueing/blocking

IPC

Group receive

Copy/share/transfer

A Unix surprise

- **sendmsg()/recvmsg() pass file descriptors!**

RPC Overview

RPC = Remote Procedure Call

Extends IPC in two ways

- **IPC = Inter-**Process **Communication**
  - **OS-level: bytes, not objects**
- **IPC restricted to single machine**

Marshalling

Server location, call semantics

Marshalling

Values must cross the network

Machine formats differ

- Serialize/de-serialize
- Format/packing
- Type mismatch issues

"The pointer problem"

RPC Overview

Call semantics

- Asynch? Batch? Net/server failure?

Client flow, server flow

- Client stub routines, server dispatch skeleton

Java RMI

- (have some sense - obviously, we didn't make you use it)

Virtualization

Interposition

- Concepts and examples

Reasons to use virtualization

- Business or otherwise

What things are easy/hard to virtualize

Parallelism: Memory Hierarchy

TLB shoot-down

- What is it? Why do we need it?
- How does it work?
- Why does it get slower as we add processors?

Cache coherence

- Standard "MESI" invalidation protocol
  - Modified/Exclusive/Shared/Invalid
  - Key idea: *atomic* operation on a cache line requires exclusive ownership of it
- New sources of cache misses
  - "True sharing" vs. "False sharing"
  - How false sharing might occur in the kernel, how to avoid it

Parallelism: Memory Consistency

Sequential consistency

- What programmers intuitively expect...
- ...but which few machines actually provide

Weaker consistency models

- "Total Store Ordering", "Release consistency"

Data Races

Properly synchronized programs

- All synchronization explicitly identified
- All data accesses ordered through synchronization

Fence operations

- Explicit (`MFENCE`, etc.) or implicit (`XCHG`, etc.)
- Do *not* use "regular load/store" for synchronization

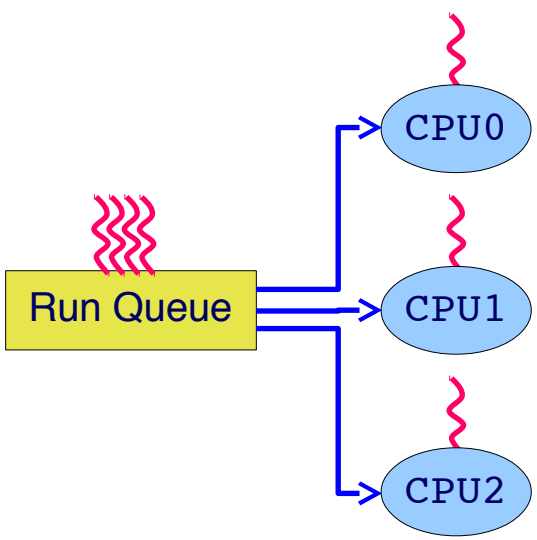
Parallelism: Multiprocessor Scheduling

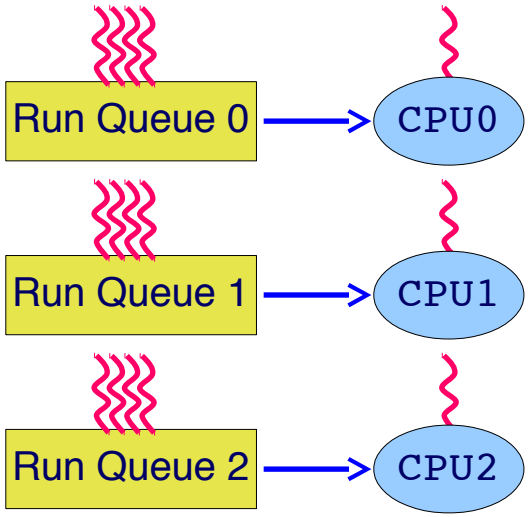
New scheduling goals

- *Load balancing* and *affinity*

Tradeoffs between *centralized* vs. *distributed* queues

- Distributed queues with *work stealing*





Parallelism: Synchronization Re-visited

Spinning vs. blocking

Coherence traffic **from a simple spin lock**

```
while (!xchg(&lock_available, 0)) continue;
```

"Test-and-test-and-set" lock

- How they work and why they help

Avoiding a lock-release traffic burst

- Backoff, "ticket locks"

Queueing locks

- Array-based, list-based

Transactional memory

- Motivation and programmer model

Coping With System Failures

Example

- Crash in the middle of a file rename()
  - File is lost?
  - Accidental extra hard link?

"fsck approach"

- Examine *all* metadata
  - Look for inconsistencies, guess what happened, do something
  - Costs time
  - "Guess what happened" has limits

General approach: *transactions*

Transactions

Basic Concept

- ACID properties
  - Atomicity – all or nothing
  - Durability – don't lose results from committed transactions
  - Isolation – concurrent transactions appear serialized
  - Consistency – (up to application logic)
- Programing model
  - Begin/Commit/Roll-back

Transactions

Write-Ahead Logging

- Why it works
- Ordering requirements
  - Record planned actions; Write commit record; “Write back” primary copy of data
- Crash-restart processing (undo/re-do)
  - What if you crash during recovery?

Transaction Examples

Transactional *Memory*

- "Durability" is different - "the truth" is in RAM
- Logging is for *undo* only (after a crash, RAM is blank)

Transactional Databases

- Maintain durable, consistent picture of *entire database*
  - Every value in every column of every row of every table
- New transactions can be added by any programmer
  - System must be robust in many ways

File Systems

- Small number of operations (I/O, rename, delete) simplifies design
- Many file systems cover only metadata (file existence, size, names), do not guarantee exactness for user data stored *inside* each file

Protection Overview

Protection vs. Security

- Inside vs. outside "the box"

Objects, operations, domains

Access control ***(least privilege)***

3 domain models

Domain switch (setuid example)

Multics ring architecture

Access Matrix

- Concept and real-world approaches

Security Overview

Goal / Threat / Response tuples

Malware

- Trojans, trapdoors
- Buffer overflow
- Viruses, worms

Password files, salt

- What is the threat, how does the technique help

Biometrics vs. cheating

Security Overview

"Understand cryptography"

- What *secure* hashing is good for
- One-time pad
- Symmetric (private-key) crypto
  - Small, "password-like" keys
- Asymmetric (public-key) crypto
  - Has private keys and public keys
  - And, in practice, symmetric session keys – know how/why
- The mysterious nonce
- Kerberos
  - Symmetric crypto
  - Central server avoids the n^2 key problem

Preparation Suggestions

Sleep well (***two*** nights)

Scan lecture notes

Read any skipped textbook sections

- **Well, the most-important ones, anyway**

Understand the code you turned in

- **Even what your partner wrote**
- **What are the hard issues, why?**

Preparation Suggestions

Prepare a sheet of notes

Read comp.risks & Effective Java

- Ok, after the exam will suffice

Don't panic!

- Budget time wisely during exam
  - (don't get bogged down on one question)

15-410 on One Slide

What a process/thread *really is*

- (the novel-length version, not the fairy tale)

Concurrency & synchronization

- Issues, mechanisms, *hazards*

How the pieces of hardware fit together

- ...to make a “system” which can run “programs”

A sense of “what's out there” beyond the kernel

Skills for non-small software artifacts

- Design, debugging, partnering
- Documenting, source control

Closing Thought

To understand a program you must become both the machine and the program.

-Alan Perlis