

**15-410**

*“...This is a **transformative** class...”*

Review  
May 4, 2007

**Dave Eckhardt**

**Bruce Maggs**

# Synchronization

**Time is running out**

# Synchronization

## Time is running out

- <http://www.cmu.edu/fce>

# Synchronization

## Homework 2 due tonight

- Solutions will be released “immediately”
- ⇒ No late days

## Book report due tonight

## Exam Thursday

- <http://www.cmu.edu/hub/current-finals.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 them 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...

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

## Examples

- Plan 9
- Lock-free programming
- Transactions
- Freeblock scheduling
- OS Structure

# 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

# Scheduling

## Round-Robin

### Things people do

- Multi-level feedback queues

### Dubious

- Priority

### “Priority Inheritance”

# Disk scheduling

**Spinning platter/waving arm model**

**Seek time vs. rotational latency**

**FCFS, SSTF, SCAN, LOOK, C-SCAN, C-LOOK, SPTF,  
WSPTF**

**Fairness, mean response time, variance, starvation**

# Disk Array Overview

## Historical practices

- Striping, mirroring

## The reliability problem

- More disks  $\Rightarrow$  *frequent* array failures
- *Cannot* tolerate  $1/N$  reliability

## Parity, ECC, why parity is enough

- Erasure channels
  - Good terminology to display at parties

# Disk Array Overview

## RAID “levels” (really: flavors)

- Understand RAID 0, 1, 4 vs. 5
- What they're good for, why



# 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**

# 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)

# Marshalling

**Values must cross the network**

**Machine formats differ**

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

**“The pointer problem”**

# Distributed File Systems

## Client-side redirection

- Which files are remote? How to operate remotely?

## Communication

- RPC, XDR
- Flow of RPC operations to access a file
  - Mount, lookup, access
- Batching (what, why)
- Cache vs. consistency vs. locks

# 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**