Preliminaries

CS 450 : Operating Systems

Michael Saelee <lee@iit.edu>

IIT College of Science
ILLINOIS INSTITUTE OF TECHNOLOGY

Michael (Sae) Lee

Email: saelee@iit.edu

Office: SB 226A

Hours: MW, 11:30-12:30

Agenda

- prerequisites
- website, textbooks & other class resources
- assignments, exams & grading
- class overview

§Prerequisites

Operating system abstractions:

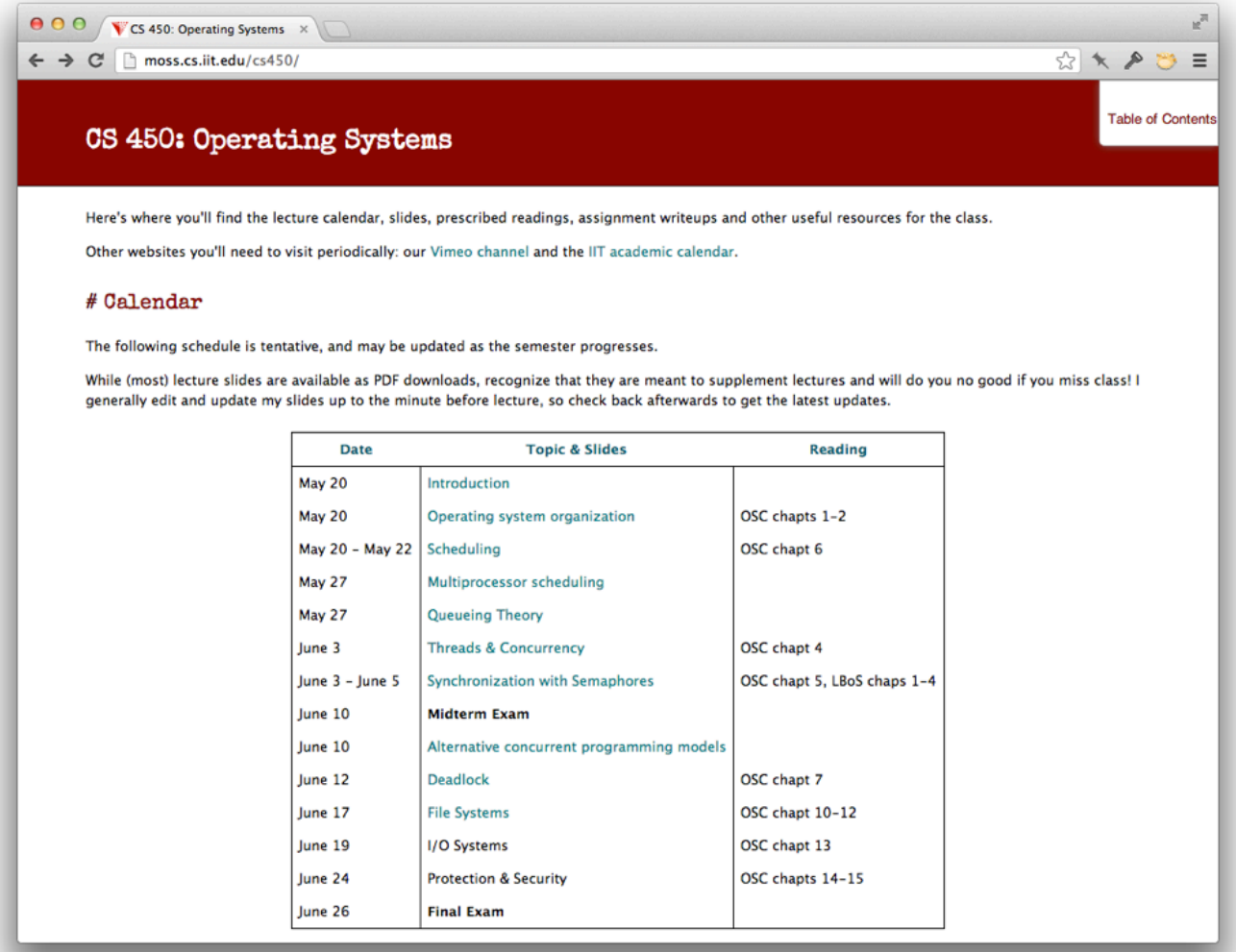
- the process
- concurrency & exceptional control flow
- memory hierarchy (caching)
- virtual memory
- files and I/O structures (e.g., file descr.)

System level APIs (i.e., syscalls) for:

- process management
- exceptional control flow
- input/output
- interprocess communication

- C
- Some form of assembly (e.g., x86, ARM)
- Stack usage (in procedure call/return)

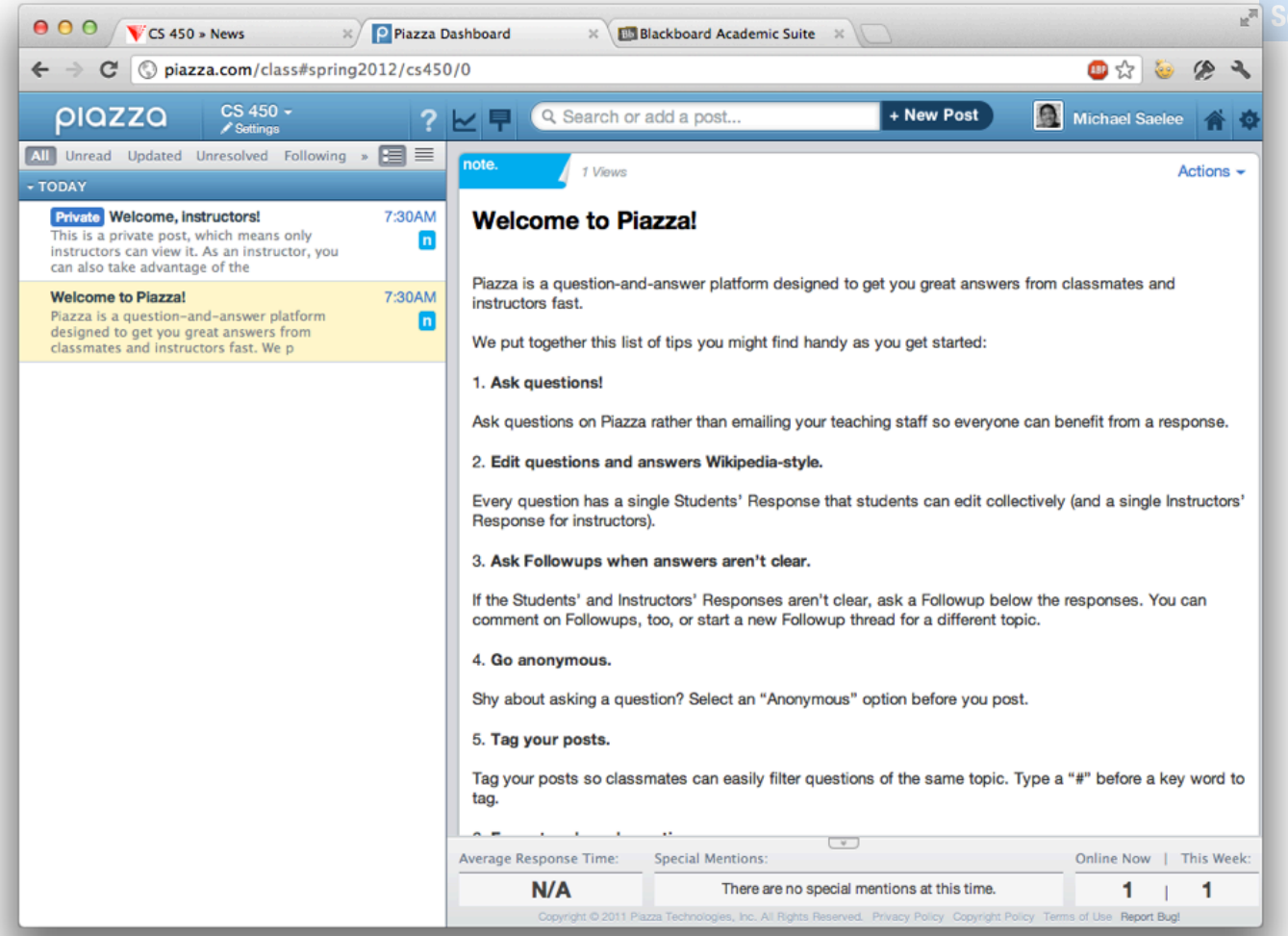
§Class resources



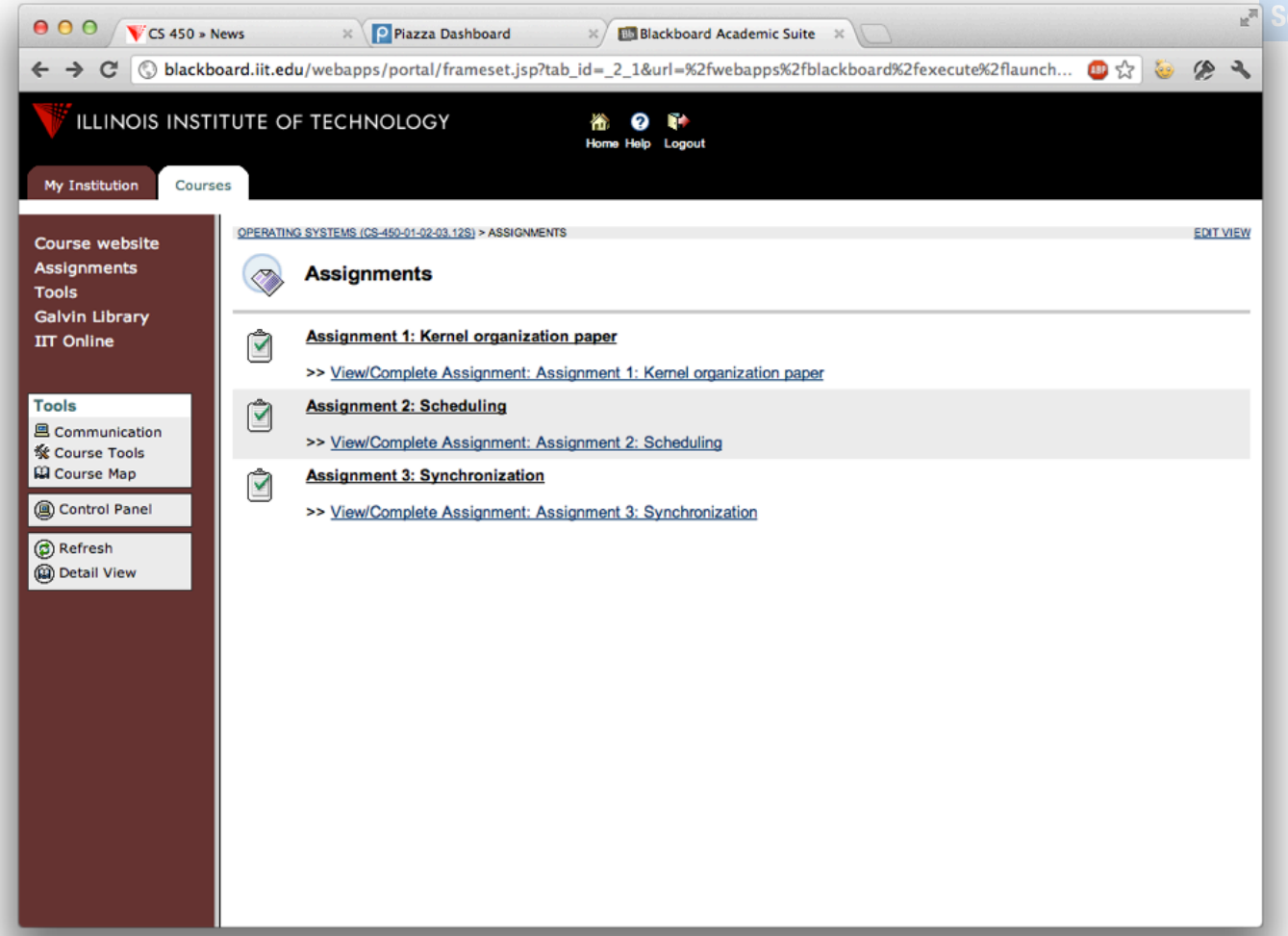
Date	Topic & Slides	Reading
May 20	Introduction	
May 20	Operating system organization	OSC chapts 1–2
May 20 – May 22	Scheduling	OSC chapt 6
May 27	Multiprocessor scheduling	
May 27	Queueing Theory	
June 3	Threads & Concurrency	OSC chapt 4
June 3 – June 5	Synchronization with Semaphores	OSC chapt 5, LBoS chaps 1–4
June 10	**Midterm Exam**	
June 10	Alternative concurrent programming models	
June 12	Deadlock	OSC chapt 7
June 17	File Systems	OSC chapt 10–12
June 19	I/O Systems	OSC chapt 13
June 24	Protection & Security	OSC chapts 14–15
June 26	**Final Exam**	

Class website: http://moss.cs.iit.edu/cs450

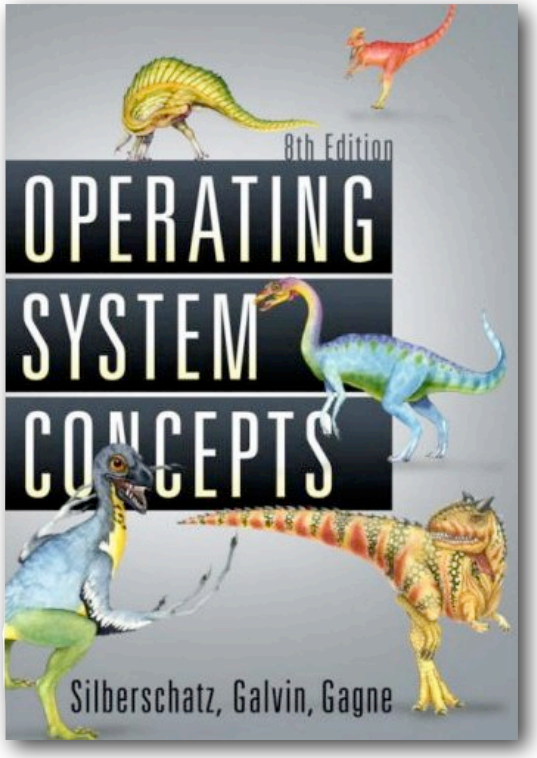
(not yet updated for Fall 2013!)



Class Q/A forum: http://piazza.com



Blackboard: http://blackboard.iit.edu



Required (Printed) Textbook: Operating System Concepts (OSC)

§Assignment, Exams & Grading

6-8 assignments — 50% of grade:

- written paper
- quantitative analysis
- machine problem

two exams (midterm & final) @ 25% each:

- final is *not* comprehensive
- *no* curving, scores normalized to 75%
  - score ≥ 50% on both exams to pass

A: ≥ 90%

B: 80-89%

C: 70-79%

D: 60-69%

E: < 60%

§Class Overview

prereqs = you should already know what services are *provided* by OSes, along with:

- how to invoke them (syscalls)
- how to use them efficiently
- how they are (conceptually) implemented

you should be familiar with details of:

- exceptional control flow
- file system structures (FDs, OFDs, etc.)
- virtual memory management constructs (e.g., page tables, TLB, etc.)

lingering questions:

- how are processes scheduled?
- how to leverage concurrency?
- how is the file system implemented (and how does I/O work, in general)?
- how are protection/security enforced?

primary topics:

1. scheduling and process management
2. concurrency and synchronization
3. storage management
4. protection and security

plenty of breadth/depth:

- queueing theory
- different approaches to concurrent programming (e.g., message passing)
- file system implementation

the debate: theory vs. implementation

- OSes are too big a topic for both
- theory first – (hopefully) broad application
- but it'd be nice to see some working OS code, too ...

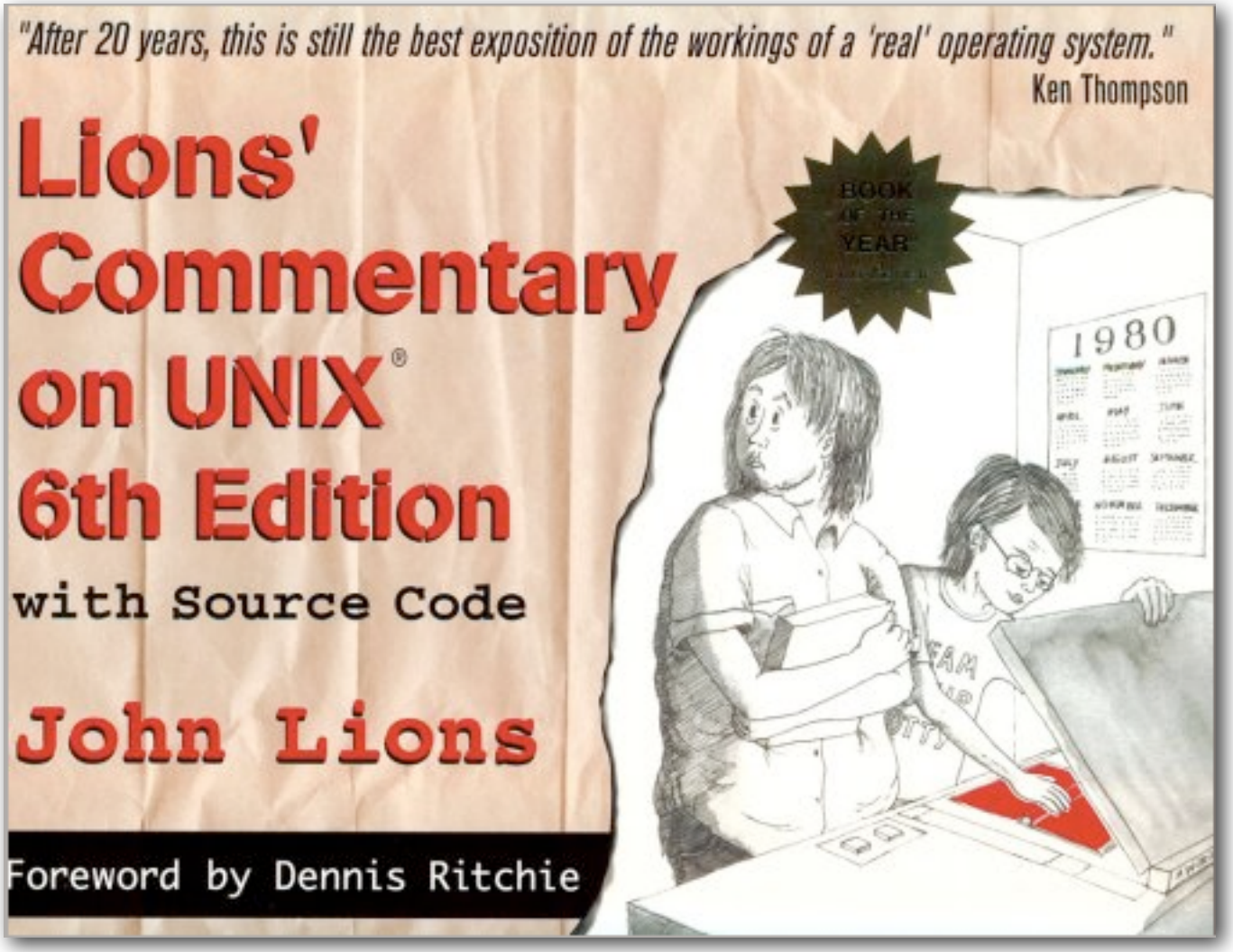
… the best way to prepare [to be a programmer] is to write programs, and to study great programs that other people have written.

In my case, I went to the garbage cans at the Computer Science Center and fished out listings of their operating system.

- Bill Gates

Those who do not understand Unix are condemned to reinvent it, poorly

- Henry Spencer


"After 20 years, this is still the best exposition of the workings of a 'real' operating system."
Ken Thompson
Lions' Commentary on UNIX® 6th Edition
with Source Code
John Lions
Foreword by Dennis Ritchie
BOOK OF THE YEAR
1980

Sep 1 09:28 1988 unix/malloc.c Page 1

```
2500 #
2501 /*
2502  */
2503
2504 /*
2505  * Structure of the coremap and swapmap
2506  * arrays. Consists of non-zero count
2507  * and base address of that many
2508  * contiguous units.
2509  * (The coremap unit is 64 bytes,
2510  * the swapmap unit is 512 bytes)
2511  * The addresses are increasing and
2512  * the list is terminated with the
2513  * first zero count.
2514  */
2515 struct map
2516 {
2517     char *m_size;
2518     char *m_addr;
2519 };
2520 /* -------------------------       */
2521
2522 /*
2523  * Allocate size units from the given
2524  * map. Return the base of the allocated
2525  * space.
2526  * Algorithm is first fit.
2527  */
2528 malloc(mp, size)
2529 struct map *mp;
2530 {
2531     register int a;
2532     register struct map *bp;
2533
2534     for (bp = mp; bp->m_size; bp++) {
2535             if (bp->m_size >= size) {
2536                     a = bp->m_addr;
2537                     bp->m_addr =+ size;
2538                     if ((bp->m_size =- size) == 0)
2539                         do {
2540                             bp++;
2541                             (bp-1)->m_addr = bp->m_addr;
2542                         } while((bp-1)->m_size = bp->m_size);
2543                     return(a);
2544             }
2545     }
2546     return(0);
2547 }
2548 /*----------------------    */
2549
```


Sheet 25

Sep 1 09:28 1988 unix/malloc.c Page 2

```
2550 /*
2551  * Free the previously allocated space aa
2552  * of size units into the specified map.
2553  * Sort aa into map and combine on
2554  * one or both ends if possible.
2555  */
2556 mfree(mp, size, aa)
2557 struct map *mp;
2558 {
2559     register struct map *bp;
2560     register int t;
2561     register int a;
2562
2563     a = aa;
2564     for (bp = mp; bp->m_addr<=a && bp->m_size!=0; bp++);
2565     if (bp>mp && (bp-1)->m_addr+(bp-1)->m_size == a) {
2566             (bp-1)->m_size =+ size;
2567             if (a+size == bp->m_addr) {
2568                     (bp-1)->m_size =+ bp->m_size;
2569                     while (bp->m_size) {
2570                             bp++;
2571                             (bp-1)->m_addr = bp->m_addr;
2572                             (bp-1)->m_size = bp->m_size;
2573                     }
2574             }
2575     } else {
2576             if (a+size == bp->m_addr && bp->m_size) {
2577                     bp->m_addr =- size;
2578                     bp->m_size =+ size;
2579             } else if(size) do {
2580                     t = bp->m_addr;
2581                     bp->m_addr = a;
2582                     a = t;
2583                     t = bp->m_size;
2584                     bp->m_size = size;
2585                     bp++;
2586             } while (size = t);
2587     }
2588 }
2589 /*----------------------  */
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
```

Reproduced under license from the Western Electric Company, NY
Copyright, J. Lions, 1976

Sheet 25

< 10,000 lines of code

- compared to > 10,000,000 for modern kernels (e.g., Linux)
- multi-user
- preemptively multitasked
- a UNIX

But: antiquated architecture (PDP/11) and language (pre-ANSI C)

- inconvenient to simulate and tweak

Enter xv6 — x86-based "clone" of v6:

- similarly small codebase
- machine problems will ask you to read through and make modifications to it