ECE469: Operating Systems Engineering

Yiying Zhang

1/10/2017

About This Course

- ECE 469 - Operating Systems Engineering
  - Undergraduate-level operating systems
  - Basic OS concepts and mechanisms + hands-on projects
- Prerequisite:
  - ECE368 (Data Structures)
  - ECE437 (Introduction to Digital Computer Design and Prototyping)
  - Programming proficiency in C is absolutely required

About Me (https://engineering.purdue.edu/~yiying/)

Vincent C. Rideout (MS 1940, Caltech)

-- Gerald Estrin (PhD 1951, University of Wisconsin)

---- David Martin (PhD 1966, University of California at Los Angeles)

------ David Patterson (PhD 1976, University of California at Los Angeles)

-------- Remzi and Andrea Arpaci-Dusseau (PhD 1999, University of California Berkeley)

---------- Yiying Zhang (PhD 2013, University of Wisconsin)

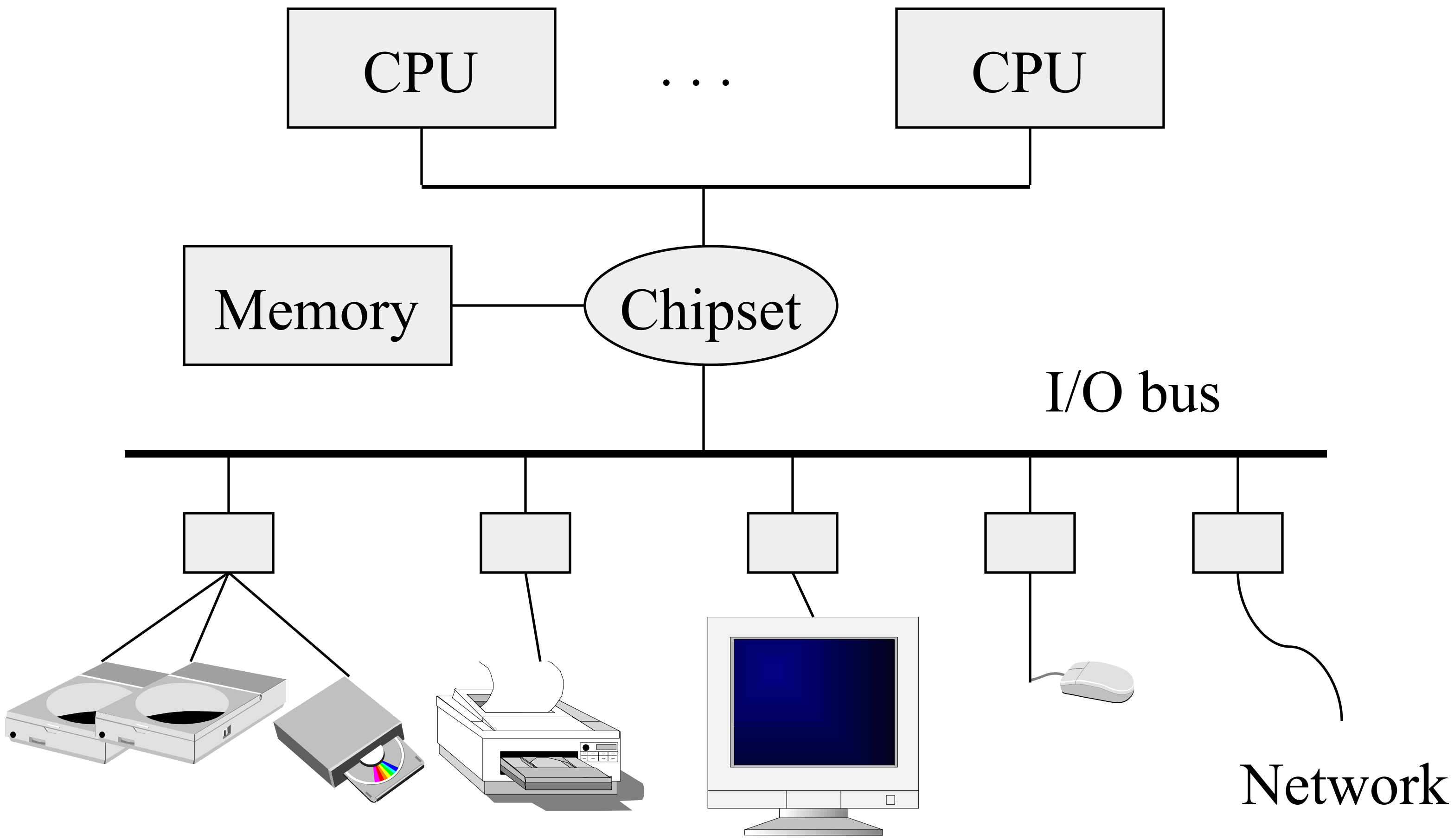
- Research interests:
  - Operating systems
    - Kernel, file systems, memory systems, etc.
  - Storage Systems
  - Distributed systems
  - Datacenter Networking

About WukLab

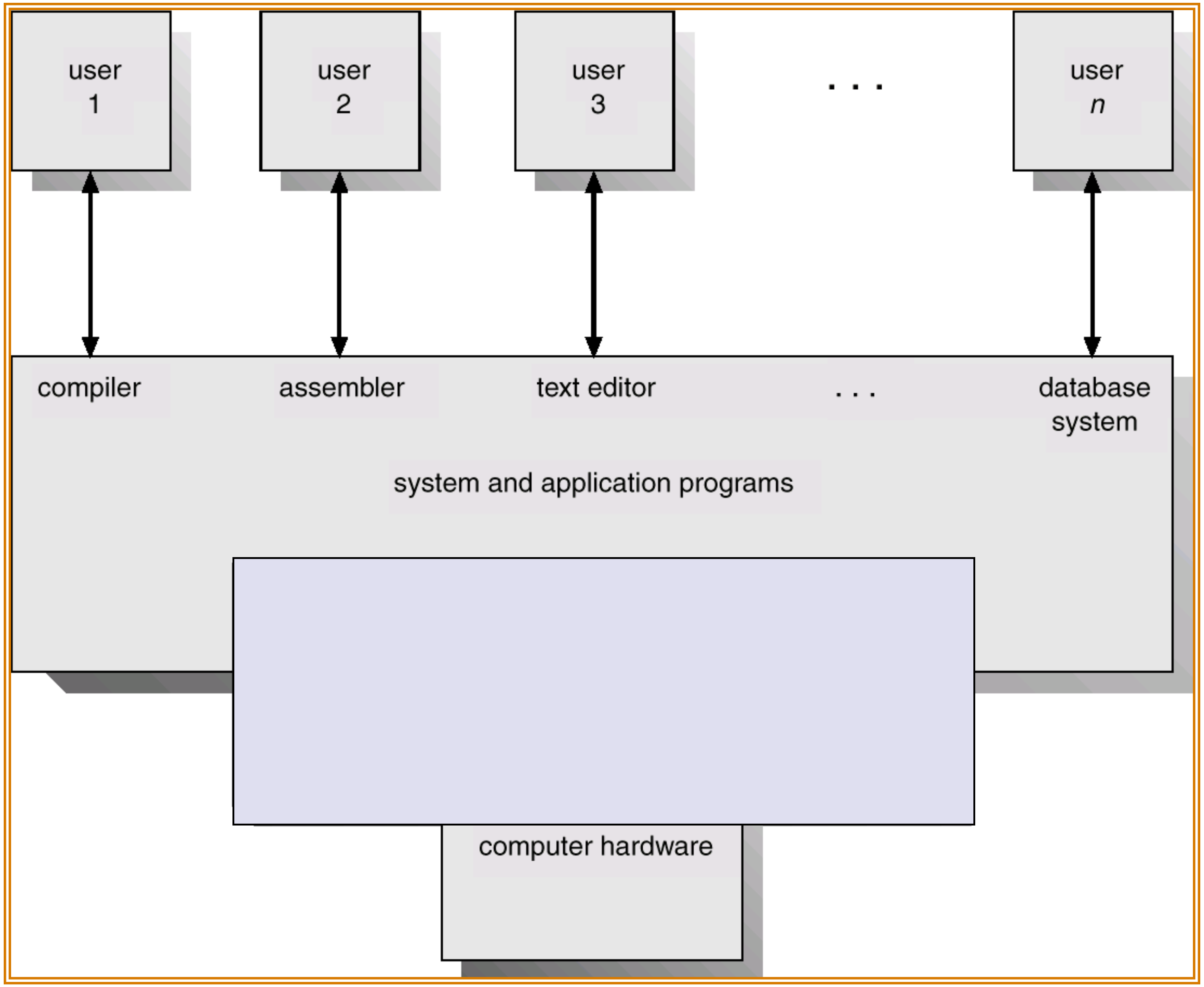
- Research focus: datacenter systems
  - Operating systems
  - Distributed systems
  - Datacenter networking
  - Security
  - Computer architecture
  - Big data
  - We are building a brand new OS from scratch now!

- Welcome undergraduate research participation! Just come talk to me.

A Typical Computer from a Hardware Point of View

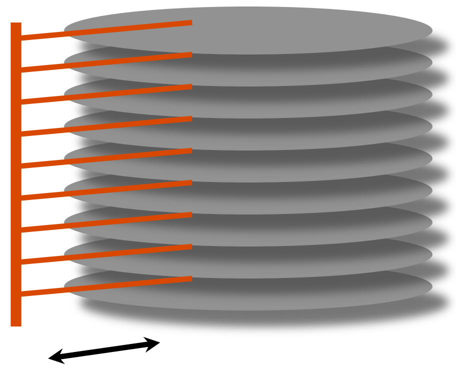


Computer System Components



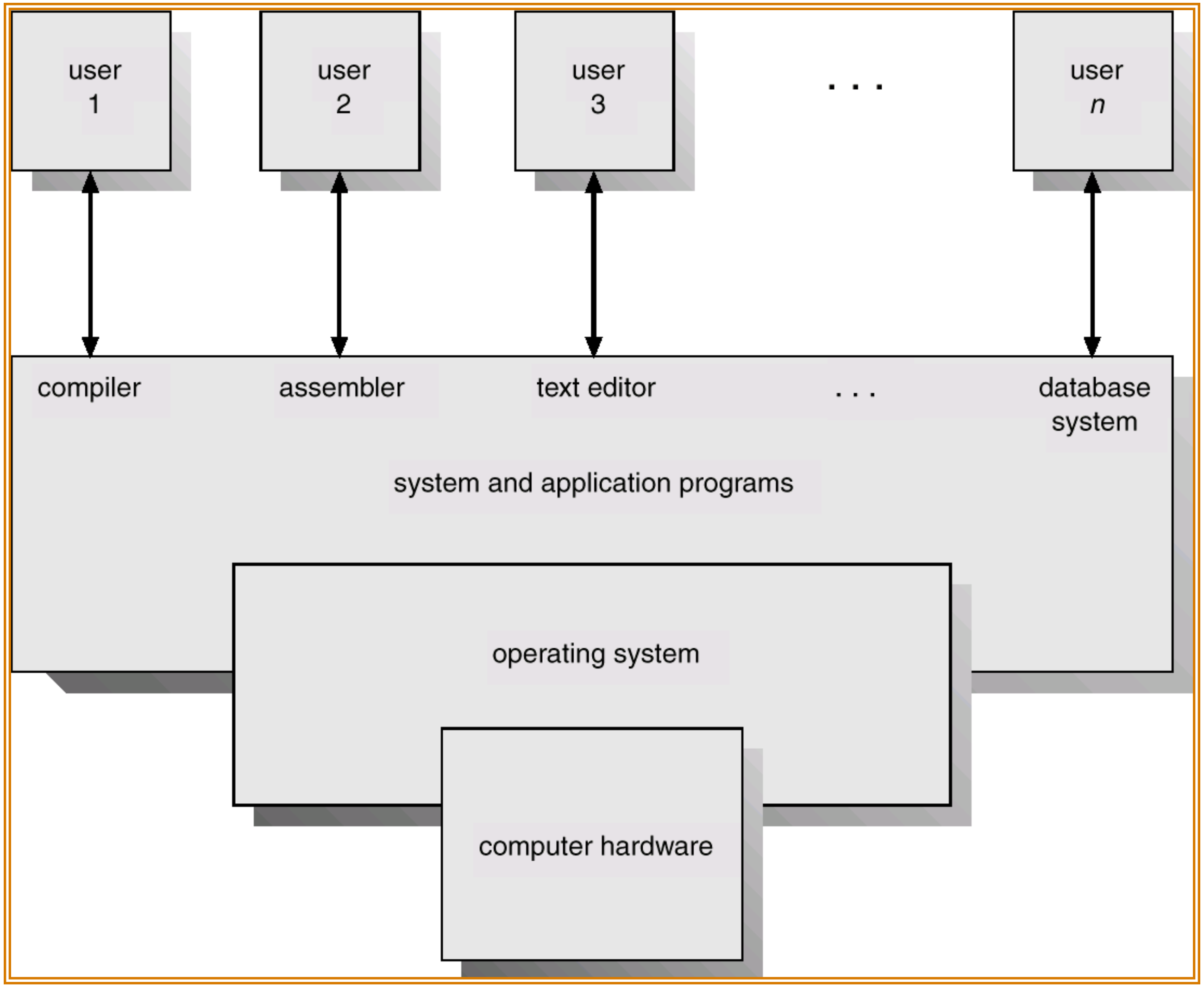
Example: programming hard drive

- Physical reality
  - Block oriented (e.g. 512 bytes)
  - Physical sector numbers
  - No protection among users of the system
  - Data might be corrupted if machine crashes
  - Programming:
    - Loading values into special device registers



"I will save my lab1 solution on platter 5, track 8739, sector 3-4."

Computer System Components



Example: programming hard drive

- Physical reality
  - Block oriented
  - Physical sector numbers
  - No protection among users of the system
  - Data might be corrupted if machine crashes
  - Programming:
    - Loading values into special device registers

- File system abstraction
  - Byte oriented
  - Named files
  - Users protected from each other
  - Robust to machine failures
  - Programming
    - open/read/write/close

"I will save my lab1 solution on platter 5, track 8739, sector 3-4."

"My lab1 solution is in ~yiying/lab1/process.c."

Brief History of Computer Systems (1)

- In the beginning, 1 user/program at a time
- Simple batch systems were 1st real OS
  - Spooling and buffering allowed jobs to be read ahead of time
  - Advantages:
    - Computer does all the work without manual intervention.
    - Could increase performance as a new job gets started as soon as the previous job finishes without any manual intervention.
  - Disadvantages:
    - Difficult to debug program.
    - A job could enter an infinite loop.
    - Due to lack of protection schemes, one batch job can affect pending jobs.

Brief History of Computer Systems (2)

- Multiprogramming systems provided increased utilization (throughput)
  - Multiple runable jobs loaded in memory
  - Overlap I/O with computation
  - 1st instance where the OS must schedule resources
    - CPU scheduling
    - Memory management
    - Protection
  - Advantages
    - Feels that many programs are allotted CPU almost simultaneously.
    - High and efficient CPU utilization.
    - Benefit from asynchronous I/O devices (interrupt, DMA, …)
  - Disadvantages
    - CPU scheduling is required.
    - To accommodate many jobs in memory, memory management is required.

Brief History of Cmputer Systems (3)

- Timesharing systems support interactive use
  - Logical extension of multiprogramming
  - Permits interactive work
    - Each user feels he/she has the entire machine
  - Optimize response time by frequent time-slicing multiple jobs
  - More complex than multiprogramming OS
    - In addition to CPU scheduling, memory management, protection
    - Virtual memory to allow part of the job be in memory
    - File system (needed by interactive use)
    - Job communication, synchronization
    - Handling deadlocks
  - Most systems today are timesharing (focus of this class)

What is an OS?

“Code” that *sits between*:

- programs & hardware
- different programs
- different users

But what does it do/achieve?

What is an OS?

- Resource manager
- Extended (abstract) machine

Makes computers *efficient* and *easy* to use

- (will have a 3rd def based on pragmatics)

What is an OS?

Resource manager (answer1)

- Allocation
- Reclamation
- Protection

What is an OS?

Resource manager

- Allocation
- Reclamation
- Protection

Finite resources

Competing demands

Examples:

- CPU
- Memory
- Disk
- Network

What is an OS?

Resource manager

- Allocation
- Reclamation
- Protection

"The OS giveth
The OS taketh away"

Implied at termination

Involuntary at run time

Cooperative (yield cpu)

What is an OS?

Resource manager

- Allocation
- Reclamation
- Protection

"You can't hurt me
I can't hurt you"

Implies some degree of safety & security

What is an OS?

Extended (abstract) machine (answer 2)

- Much more ideal environment than the hardware
  - Ease to use
  - Fair (well-behaved)
  - Supporting backward-compatibility
  - Reliable
  - Secure
- Illusion of infinite, private (reliable, secure) resources
  - Single processor → many separate processors
  - Single memory → many separate, larger memories

Separating Policies from Mechanisms

A fundamental design principle in Computer Science

Mechanism – tool/implementation to achieve some effect

Policy – decisions on what effect should be achieved

Example – CPU scheduling:

- All users treated equally
- All program instances treated equally
- Preferred users treated better

Separation leads to flexibility!

Is there a perfect OS?
(resource manager, abstract machine)

Efficiency

Fairness

Portability

Interfaces

Security

Robustness

- Conflicting goals
  - Fairness vs efficiency
  - Efficiency vs portablity
  - …
- Furthermore, …

Hardware is evolving…

- 60’s-70’s - Mainframes
  - Rise of IBM
- 70’s - 80’s – Minicomputers
  - Rise of Digital Equipment
- 80’s - 90’s – PCs
  - Rise of Intel, Microsoft
- 90’s - 00’s – handheld/portable systems (laptops)
- 2007 - today -- mobile systems (smartphones), Internet of Things
  - Rise of iPhone, Android

Implications on OS Design Goals: Historical Comparison

	Mainframe	Mini	Micro/ Mobile
System $/ worker	10:1 – 100:1	10:1 – 1:1	1:10-1:100
Performance goal	System utilization	Overall cost	Worker productivity
Functionality goal	Maximize utilization	Features	Ease of Use

Hardware is evolving (cont) …

- (once) New architectures
  - Multiprocessors
  - 32-bit vs. 64-bit
  - Multi-core
- New memory, storage, network devices

May You Live in Interesting Times…

- Processor speed doubles in 18 months
  - Number of cores per chip doubles in 24 months
- Disk capacity doubles every 12 months
- Global bandwidth doubles every 6 months

→ Performance/cost "sweet spot" constantly decaying

** Does human productivity ever double?*

Applications are also evolving…

- New applications
  - Scientific computing
  - Computer games
  - Java
  - WWW (web servers, browsers)
  - Networked games
  - Peer-to-peer
  - Web 2.0 (search, youtube, social network, …)
  - Mobile apps (> 2.8 million iPhone, Android apps)
  - Big data
  - Machine learning, deep learning
  - Virtual reality
  - …

Implications to OS Design

- Constant evolution of hardware and applications continuously reshape
  - OS design goals (performance vs. functionality)
  - OS design performance/cost tradeoffs

- Any magic bullet to good OS design?

There is no magic in OS design

This is Engineering

- Imperfection
- Tradeoffs (perf/func/security)
- Different Goals
- Constraints
  - hardware, cost, time, power
- Optimizations

Nothing's Permanent

- High rate of change
  - Hardware
  - Applications
- Cost / benefit analyses
- One good news:
  - Inertia of a few design principles

About this course…

Principles of OS design

- Some theory
- Some rational
- Lots of practice

Goals

- Understand OS design decisions
- Last piece of the “puzzle”
- Basis for future learning

To achieve the goals:

- Learn concepts in class
- Get hands “dirty” in labs

Major issues in OS

- Concurrency – how to create and control parallel activities
- Sharing – how to share resources among users
- Naming – how to name resources
- Protection – how to protect user/program from each other
- Security – how to restrict the flow of information
- Performance – why is it so slow
- Reliability – how to handle failures
- Extensibility – how to add new features
- Communication – how & with whom can we communicate
- Persistence – how to make data last longer than programs
- Accounting – who pays the bills & how to control resource usage

Topics we'll cover

- Process management
- Memory management
- I/O management
- Protection and Security
- Intro to distributed systems
- A touch of advanced topics if have time

Expect (some) pain

Somewhat fast pace

Lots of programming projects

Some difficult (abstract) concepts

Mechanics – Course Staff

Instructor:

Yiying Zhang, yiying@purdue.edu, EE 330

Office hours: Thur 10:15am-noon and by appt.

TAs:

Sumukh Hallymysore Ravindra, shallymy@purdue.edu

Sanghyun Cho, cho303@purdue.edu

TA office hours and location: check course website

Mechanics – General Info

- Course home page: engineering.purdue.edu/~ee469
  - Login: ee46917:students17
- Announcement, grading, and course discussion via Blackboard

Mechanics – Q & A

- Questions of general interests → Blackboard forum
- Other questions → TAs (esp. grading-, project-related) and instructor
- Announcements → Blackboard (with email notice)

Mechanics – Textbook

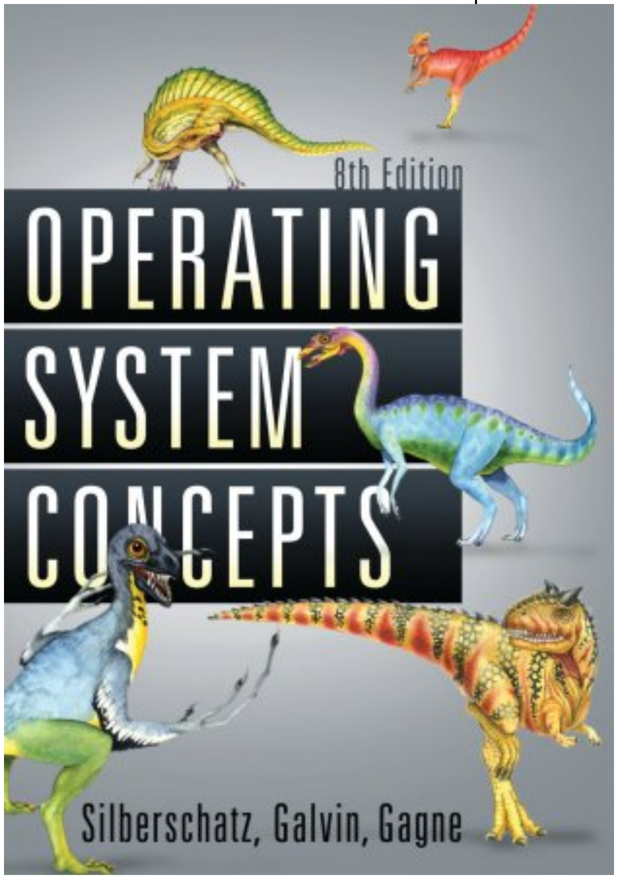
Operating System Concepts

Silberschatz, Galvin, and Gagne,

8th (7th, 6th) edition

AKA the Dinosaur Book

- Explains concepts very well

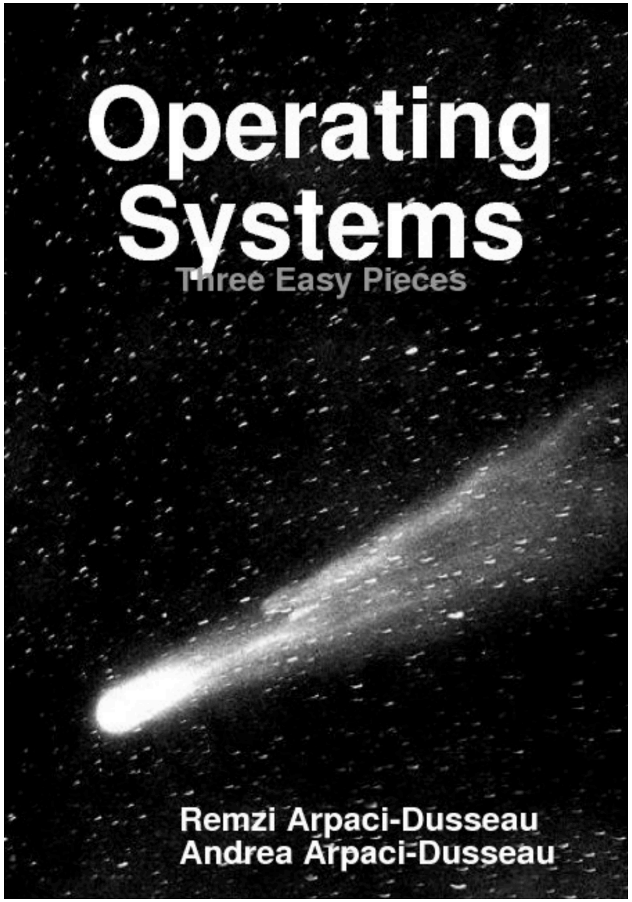


Mechanics - Additional Reading

Some papers – will be available from class web page

Optional additional textbook:
Operating Systems: Three Easy Pieces, by Remzi and Andrea Arpaci-Dusseau
http://www.ostep.org

Free online book,
easy to understand and follow,
useful for interview preparation too



Mechanics – Lecture Notes

If available, will be provided on the web

Not necessary self-contained, complete, or coherent

Not a substitute for attending classes

Mechanics - Projects

- 5 lab projects
  - **Use DLXOS, simulated on Linux machines**
  - Build parts of a mini-OS (DLXOS)
- 1st not graded (DLXOS tutorial)
- 3 weeks each (excl. spring break)
  - explained in the corresponding first week's lab
  - due: usually *Sunday midnights*
  - no extensions
- Work in pairs (optional to work on your own)
  - Team members from the same lab highly preferred
  - Be decent to each other!

Pairing Programming

From the McDowell et al. article (CACM 2006):

- Sitting shoulder to shoulder at 1 computer
  - one member of the pair is the "designated driver", actively creating code and controlling the keyboard and mouse.
  - The "non-driver" constantly reviews the keyed data in order to identify tactical and strategic deficiencies, including erroneous syntax and logic, misspellings, and implementations that don't map to the design

The McDowell et al. Study (CACM 2006) (UC Santa Cruz)

	Female		Male		All	
	Pair	Solo	Pair	Solo	Pair	Solo
% that persisted in the course and took the final	88.1	79.5	91.7	81.5	90.8	80.4
% of students taking the final that passed the class with C or better	74.2	74.2	81.3	79.5	79.6	78.2
% of passers that took the 2nd programming course within 1 year	61.1	50.0	81.2	66.1	76.7	62.2
% of passers that took the 2nd course within 1 year—restricted to those indicating a planned CS related major at start of intro course	73.8	55.6	88.0	69.4	84.9	66.7
% of those taking the 2nd course that passed it on the first attempt	68.3	44.4	64.6	37.5	65.5	40.0
% of passers still at UCSC 1 year later that declared a CS major	46.3	11.1	59.5	41.1	56.9	33.8
% of passers still at UCSC 1 year later that declared a CS major—restricted to those indicating a planned CS related major at start of intro course	59.5	22.2	74.0	47.2	70.8	42.2

Shaded numbers indicate statistically significant differences.

Mechanics - Exams

- Midterm
  - before Spring break, mostly in evening
- Final
  - Non-cumulative
- Multiple choices, True or False, short answers, some design (derivation), very few programming problems

Mechanics – Grading

- Programming projects (50%; 12.5% each)
- Midterm exam (25%)
- Final exam (25%)
- Bonus pop quizzes (see me about health/school absences)

- Late policy:
  - No extension if total machine down time < 3 days per lab (you have 3 weeks per assignment)

Academic Integrity

- Programming assignments
  - Ask TAs / instructor for clarification
  - Each team must write their own solution
  - No discussion of or sharing of specific code or written answers is allowed
  - Any sources used outside of textbook/handouts/ lectures must be explicitly acknowledged
  - Your responsibility to protect your files from
    - e-copying using UNIX file protection
    - public access, including disposal

Academic Integrity Policy

- Cheating
  - **The first case of cheating on an assignment will result in zero for that whole assignment & reporting to university administration for disciplinary action**
  - **The second case will result in an immediate F grade for the course**

ABET outcomes

- By early March: document on mapping of questions in labs/exams to ABET outcomes
- By early March: Remediation homework

Questions?

- Reading assignment:
  - Dinosaur Chapters 1-2, by Thursday
  - Alternative: Comet Chapters 1-2
- Find a lab partner and email the TAs ASAP
  - No later than Jan 26 (lab2 starts)
- Start lab 1 soon