ECE595: Introduction to Operating Systems

Y. Charlie Hu

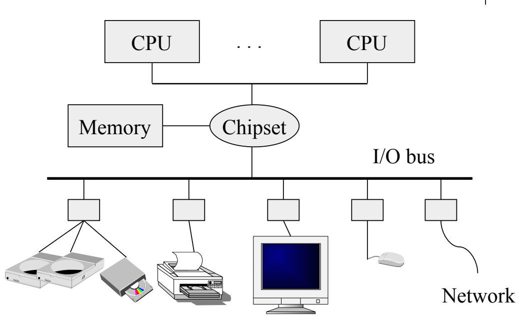
8/21/2018

About Me (www.ece.purdue.edu/~ychu)

- Ph.D in CS, 1997, Harvard
- Joined Purdue 2002
- Research interests:
  - Operating systems
    - Virtual memory
    - File system
  - Networking (Internet, Datacenter, Wireless)
  - Distributed systems
    - Peer-to-Peer
    - Cloud computing
  - Mobile (Smartphone Energy Debugging)
- Teach ECE673 (Distributed Systems) in Springs

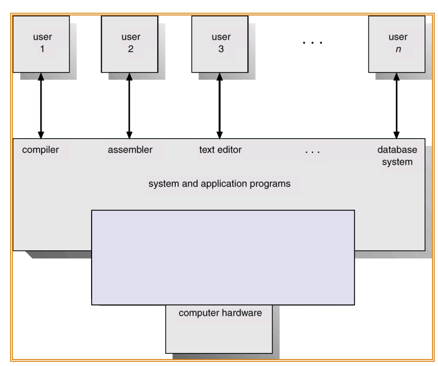
2

A Typical Computer from a Hardware Point of View



3

Computer System Components



4

Missing piece of the "puzzle"

main.c
math.c

compiler

main.o
math.o

linker

a.out

OS

Load a.out to mem
Manage resources
to run a.out

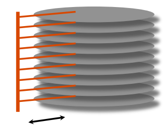
arch

Instruction
execution

5

Example: programming hard drive

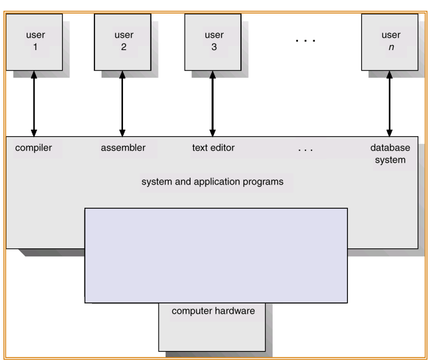
- Physical reality
  - Block oriented (e.g. 512 bytes)
  - Physical sector numbers
  - No protection among users of the system
  - Data might be corrupted if machine crashes
  - Programming:
    - Loading values into special device registers



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

6

Computer System Components



7

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, sectors 3-4."

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

8

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
- Multiprogramming systems provided increased utilization (throughput)
  - multiple runable jobs loaded in memory
  - overlap I/O with computation
  - benefit from asynchronous I/O devices (interrupt, DMA, …)
  - 1st instance where the OS must schedule resources
    - CPU scheduling
    - Memory management
    - Protection

Brief History of Cmputer Systems (2)

- 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 definition based on pragmatics)

What is an OS?

Resource manager (answer1)
- Allocation
- Reclamation
- Protection

13

What is an OS?

Resource manager
- Allocation
- Reclamation
- Protection

Finite resources
Competing demands

Examples:
- CPU
- Memory
- Disk
- Network

14

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)

15

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

16

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

21

Hardware is evolving (cont) …

- New architectures
  - Multiprocessors
  - 32-bit vs. 64-bit
  - Multi-core

22

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?*

23

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)
  - …

24

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?

25

There is no magic in OS design

This is Engineering

- Imperfection
- Tradeoffs (perf/func)
- 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

26

Break / Assessment Quiz (7 min)

- Purposes
  - To assess your background knowledge, so
    - you know what you need to brush up
    - I know how to better structure lecture material
  - A glimpse of the midterm/final exam format

27

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

28

Expect (some) pain

Somewhat fast pace

Lots of programming projects

Some difficult (abstract) concepts

29

Mechanics – Course Staff

Y. Charlie Hu, ychu@ecn, MSEE 232
Office hours: Wedn 1:30-3:30pm and by appt.

No TA. ☹

Volunteering Helper with programming projects:
Jiayi Meng, Ph.D student, ece595@purdue.edu

30

Mechanics – General Info

- Course home page: engineering.purdue.edu/~ece595
  - login/password
- Mailing list: fall-2018-ece-59500-005@lists.purdue.edu
  - Class announcements
  - Q&A of general interests
- “TA” email: ece595@ecn.purdue.edu

31

Mechanics – Q & A

- Questions of general interests → mailing list
- Other questions → ece595@ecn
- Announcements → mailing list

32

Mechanics – Textbook

Operating System Concepts
Silberschatz, Galvin, and Gagne, 9^{th} (8^{th}, 7^{th}, 6^{th}) edition

Explains concepts very well

Some papers – will be available from class web page

33

Mechanics – Lecture Notes

If available, will be provided on the web

Not necessary self-contained, complete, or coherent

Not a substitute for attending classes

34

Mechanics - Projects

- 4 Programming projects
  - **Use DLXOS, simulated on Linux machines**
  - Build key components of a mini-OS (DLXOS)
- 1^{st} not graded (DLXOS tutorial)
- 3~4 weeks each
  - Explained in designated Friday lectures
  - due: *Sunday midnights, no extensions*
- Work in pairs (optional to work on your own)
  - Be decent to each other!

35

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

36

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: close-book
  - Oct 4, Thursday 7-9pm (before October break)
- Final: close-book
- Short answers, some proof, some design (derivation), at most 1 programming problem

Mechanics – Grading

- Programming projects (50%: 15%-15%-20%)
- Midterm exam (25%)
- Final exam (25%)
- Bonus pop quizzes (up to 10%)

Academic Integrity

- Programming assignments
  - Ask instructor/helper 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

ABET outcomes

- By Oct 4: document on mapping of questions in labs/exams to ABET outcomes
- By Oct 4: Remediation homework

42

Homework

- Reading assignment:
  - OSC, Chapters 1-2, by Thursday
- Find a lab partner and email ece595@ecn
  - No later than Sep 3

43