Course overview

- Everything you need to know will be on the course web page:
  
  http://www.cs.washington.edu/education/courses/451/CurrentQtr

Today’s agenda

- Administrivia
  - course overview
    - course staff
    - general structure
    - your to-do list
  - overloading
- OS overview
  - functional
    - resource mgmt, major issues
  - historical
    - batch systems, multiprogramming, time shared OS’s
    - PCs, networked computers

- But to tide you over for the next hour …
  - course staff
    - Brian Bershad (bershad@cs.washington.edu)
    - Kararyzna Wilamowska (kasiaw@cs.washington.edu)
    - Ilya Vladimirovich Maykov (uvmaykov@cs.washington.edu)
  - general structure
    - read the text prior to class
    - class will supplement rather than regurgitate the text
    - sections will focus on the project
    - we really want to encourage discussion, both in class and in section
– your to-do list …
  • please read the entire course web thoroughly, today
  • please get yourself on the cse451 email list, today, and check your email daily
  • homework 1 (reading + problems) is posted on the web now; due Friday
  • project 1 is posted on the web now and will be discussed in section on Thursday; due a week from Friday

Overloading

• If you’re going to drop this course
  – please do it soon!
• If you want to get into this course
  – plan for the worst case
  – but, make sure you’ve filed a petition with the advisors

Required Background

• Processor fundamentals
• Compiler fundamentals
• Programming in C

What is an Operating System?

• An operating system (OS) is:
  – a software layer to abstract away and manage details of hardware resources
  – a set of utilities to simplify application development

  – “all the code you didn’t write” in order to implement your application
What is Windows?

Application

DOS

What is Windows?

Application

DOS

Windows

What is .NET?

Application

Internet

What is .NET?

Application

Internet

.NET

Device independence
XML
Identity & security
Asynchrony
Extensibility

eBay
FedEx
Bank

Windows

Browser
TCP/IP
COM
Installer
Printing

Microsoft

Excel

Word

VISA

eBay

FedEx

Extensibility
Asynchrony

XML

Device independence
Identity & security
The major OS issues

- **structure**: how is the OS organized?
- **sharing**: how are resources shared across users?
- **naming**: how are resources named (by users or programs)?
- **security**: how is the integrity of the OS and its resources ensured?
  - **protection**: how is one user/program protected from another?
- **performance**: how do we make it all go fast?
- **reliability**: what happens if something goes wrong (either with hardware or with a program)?
- **extensibility**: can we add new features?
- **communication**: how do programs exchange information, including across a network?

Why bother with an OS?

- **Application benefits**
  - programming **simplicity**
    - see high-level abstractions (files) instead of low-level hardware details (device registers)
    - abstractions are **reusable** across many programs
  - **portability** (across machine configurations or architectures)
    - device independence: 3Com card or Intel card?
- **User benefits**
  - **safety**
    - program “sees” own virtual machine, thinks it owns computer
    - OS protects programs from each other
    - OS fairly multiplexes resources across programs
  - **efficiency** (cost and speed)
    - share one computer across many users
    - concurrent execution of multiple programs

The OS and hardware

- An OS **mediates** programs’ access to hardware resources
  - Computation (CPU)
  - Volatile storage (memory) and persistent storage (disk, etc.)
  - Network communications (TCP/IP stacks, ethernet cards, etc.)
  - Input/output devices (keyboard, display, sound card, etc.)
- The OS **abstracts** hardware into logical resources and well-defined interfaces to those resources
  - processes (CPU, memory)
  - files (disk)
    - programs (sequences of instructions)
  - sockets (network)

More OS issues...

- **concurrency**: how are parallel activities (computation and I/O) created and controlled?
- **scale**: what happens as demands or resources increase?
- **persistence**: how do you make data last longer than program executions?
- **distribution**: how do multiple computers interact with each other?
- **accounting**: how do we keep track of resource usage, and perhaps charge for it?
- **manageability**: how do we adapt a system to a particular environment
  - “management is whatever you need to do that you can’t write an algorithm for.”
Inflection Point #0: 19th Century

- **Machines can do things faster than people**

Business Number Crunching
Technology: 1880-1920

**Adders**
+ (and sometimes -)
mechanical devices
slow, clunky

**Calculators**
+*/
mechanical devices
slow, clunky, $\$$

**Tabulators**
electronic and mechanical
popular in the insurance industry and
w/employers
CTR

Automatic Calculators:
“Bigger Must be Better”

The Bush Differential Analyzer

- Built in 1931
- Analog and Decimal
  - poor accuracy ~ 1%
- Big & Slow
  - all mechanical
  - 100 tons
- Only one of them

In the very beginning…
- There was no OS and there was no
  program.
- Just levers, computation, and output
  (think Big Calculator)
- Problem: Really Hard to Use

Iowa in 1937

In the treatment of many mathematical
problems one requires the solution of
systems of linear simultaneous
algebraic equations.

- Curve fitting.
- Method of least squares.
- Vibration problems including
  the vibrational Raman effect.
- Electrical circuit analysis.
- Analysis of elastic structures.
- Approximate solution of many
  problems of elasticity.
- Approximate solution of
  problems of quantum
  mechanics.
- Perturbation theories of
  mechanics, astronomy and the
  quantum theory.

John Vincent Atanasoff
(A math & physics guy)
John Atanasoff Invents the First
*Electronic* Calculator

- 3kbits of RAM
- 30 calculations/minute

Key Technical Ideas

- Binary rather than decimal
- Vacuum tubes for logic rather than storage
- Inexpensive rotating capacitors for (cheap) temporary storage
- Punch cards for input/output

John Berry (Grad Student) With the
ABC Computer

What Happened to Atanasoff?

- In 1940, asks for 5K from Iowa Research Council to build a commercial-grade machine
  - he’s turned down
- He meets John Mauchly
  - physicist from U Penn
- But forgets to have him sign the NDA
- WWII starts
  - By 1942, Atanasoff resigns from Iowa State to join the war effort
The First “Killer” App

- Projectile trajectories
- 5 days per trajectory using a mechanical calculator
- 30 minutes using BDA
  - not accurate enough
  - and still takes one month/table
- John Mauchly seized the opportunity

The ENIAC

- Started in 1942
- Completed in 1945
- Key stats
  - $500K
  - 30 tons
  - 1800 square feet
  - 19000 vacuum tubes
  - 175 Kw/power
  - 5000 ops/second

Followed 2 years later by EDVAC.
- Stored Program

Electronic Numerical Integrator and Computer

IPs Leading to EDVAC

- IP #1: Application of Dynamic RAM at any cost (Vacuum tubes)
  - Computers can now execute code against data
  - Could now have a “PROGRAM” that could be loaded quickly and run.
    - programs were loaded in their entirety into memory, and executed
    - OS was just a library of code that you linked into your program
  - By using the Hollerith Card (Old Technology Reapplied) (~1952) computers can run a BATCH of programs
    - Led to “Batch Processing Languages” which told OS about the programs
    - OS loaded the next job into memory from the card reader
    - job gets executed
    - output is printed, including a dump of memory (why?)
    - repeat...
- Problem: card readers and line printers were very slow
  - so CPU (most $$ part of system) was idle much of the time (wastes $$)

1947-1952: The UNIVAC

- E&M’s first commercial computer
  - used mag tape, courtesy of the Nazis
    - WWII was a Big Inflection Point
- $1M dev costs
  - Sold to Census Bureau for $150K
- The company was undercapitalized and needed cash to build
  - No problem finding customers.
  - They couldn’t deliver.
The UNIVAC Invented Spooling

- **IP#3:** Tapes, which are faster than mechanical readers and printers
  - Spool (Simultaneous Peripheral Operation On-Line)
    - while one job is executing, spool next job from card reader onto tape
      - slow card reader I/O is overlapped with CPU
  - Problem: Tapes are sequential → can only run the “next” program.

IBM Invents The Disk in 1956

- **IP#4:** Disks, which allow Random Access to Storage
  - Run multiple programs at “once” by swapping image to disk
    - OS must choose which to run next when current ends, or performs I/O (eg, fetch data from disk)
    - Job scheduling. EARLY POLICY/Mechanism Separation Example
  - Problem: CPU still idle during I/O

Timesharing by the mid-60s

- **IP#5:** Big, Cheap Memory, to keep multiple programs in core at the same time
  - Again, to increase system utilization
  - keeps multiple runnable jobs loaded in memory at once
  - overlaps I/O of a job with computing of another
  - while one job waits for I/O completion, OS runs instructions from another job
  - to benefit, need asynchronous I/O devices
    - need some way to know when devices are done
      - interrupts
      - polling
  - goal: optimize system throughput
    - perhaps at the cost of response time…
    - Yielded the invention of the “PROCESS” – a program which is executing in memory
  - Problem: Computer is NOT interactive

- **IP#6:** Even cheaper memory, modems, and inexpensive CRTs, allowing multiple users to “interact” with the computer at the same time
  - To support interactive use, create a timesharing OS:
    - multiple terminals into one machine
    - each user has illusion of entire machine to him/herself
    - optimize response time, perhaps at the cost of throughput
  - Timeslicing
    - divide CPU equally among the users
    - if job is truly interactive (e.g. editor), then can jump between programs and users faster than users can generate load
    - permits users to interactively view, edit, debug running programs (why does this matter?)
  - MIT Multics system (mid-1960’s) was the first large timeshared system (based on the CTSS for the IBM 7094 from 1961)
    - nearly all OS concepts can be traced back to Multics
    - UNIX is just a simpler MULTICS
      - Core of Mac OS X, Linux, etc.
  - Problem: The computer ran faster at night
The Multics System

Last one decommissioned in 2000

1968: Honeywell 645
1975: Honeywell 6180

Personal and Distributed Computing by the 1970s

• IP#7: Plummeting cost of silicon and networking allows everyone to have their own computer
  – Memory was under a penny/bit
  – “Share everything but the time”
  – distributed systems using geographically distributed resources
    • workstations on a LAN
    • servers across the Internet
  – OS supports communications between jobs
    • interprocess communication
    • networking stacks
  – OS supports sharing of distributed resources (hardware, software)
    • load balancing, authentication and access control, …
  – speedup isn’t the issue
    • access to diversity of resources is goal
• Problem: there’s never a time that the machine runs faster

XEROX ALTO -- 1972

First personal workstation
First wide deployment of:
  • Bit-map graphics
  • Mouse
  • WYSIWYG editing
Hosted the invention of:
  • Local-area networking
  • Laser printing
  • All of modern client / server distributed computing

< $50K

Parallel Systems by the 80’s

• IP#8: High Speed Interconnects allow multiple processors to cooperate on a single program
  – Some applications can be written as multiple parallel threads or processes
  – can speed up the execution by running multiple threads/processes simultaneously on multiple CPUs
  – need OS and language primitives for dividing program into multiple parallel activities
  – need OS primitives for fast communication between activities
    • degree of speedup dictated by communication/computation ratio
  – many flavors of parallel computers
    • SMPS (symmetric multi-processors)
    • MPPs (massively parallel processors)
    • NOWs (networks of workstations)
    • computational grid (SETI@home)
Internet by the 90s

- IP#9: WEB, HTTP, TCP at Scale
  - TCP+HTTP+Web Browser has changed the way that the world looks at computing
  - BUT, there have been relatively few Operating System Advances to support this
    - The internet was functional by the 70s
  - Mostly, it’s been a time of leveraging the last 50 years to deploy MASSIVELY SCALABLE SYSTEMS

Ubiquitous Computing in the 21st Century

- IP#10: Massive miniaturization and integration of computers + Wireless
  - Ubiquitous computing
    - cheap processors embedded everywhere and anywhere
    - how many are on your body now? in your car?
    - cell phones, PDAs, network computers, ...
  - Typically very constrained hardware resources
    - (relatively) slow processors
    - very small amount of memory (e.g. 8 MB)
    - no disk
  - OS researchers are working in this area today to understand what we’ll be using tomorrow.

Embedded OS

- Pervasive computing
  - cheap processors embedded everywhere
  - how many are on your body now? in your car?
  - cell phones, PDAs, network computers, ...
- Typically very constrained hardware resources
  - slow processors
  - very small amount of memory (e.g. 8 MB)
  - no disk
  - typically only one dedicated application

Progression of concepts and form factors
Multiple trends at work

• “Ontogeny recapitulates phylogeny”
  – Ernst Haeckel (1834-1919)
    • The evolution of any given OS tends to follow the evolution of
      OS technologies in general.
• “Those who cannot remember the past are
  condemned to repeat it”
  – George Santayana (1863-1952)
• But new problems arise, and old problems re-define
  themselves
  – The evolution of PCs recapitulated the evolution of
    minicomputers, which had recapitulated the evolution of
    mainframes
  – But the ubiquity of PCs re-defined the issues in protection
    and security

Protection and security as an example

• Protection is a very old OS concept
  – PhD theses still collecting dust from the 60s and 70s.
• What have we learned to protect?
  – OS & Devices from my program
    • ~1960: User/Supervisor Mode
    • ~1962: Atlas (Virtual Memory)
    • ~1970: Multics
    • ~1978: Unix
    • ~1995: NT
  – OS from OS
    • ~1965: CP/370 & Virtual Machines
    • ~1971: Hydra & Capabilities/Objects
    • ~1985: Mach & Microkernels
    • ~1995: NT & Microkernels
    • ~2000: VMWARE & Virtual Machines
  – access by intruding individuals
    • ~1970: Multics

What are we not so good at (yet)?

• Unsolved, or poorly solved
  – access by intruding programs
  – denial of service
  – distributed denial of service
  – spoofing
  – spam
  – worms
  – viruses
  – stuff you download and run knowingly (bugs, trojan horses)
  – stuff you download and run unknowingly (cookies, spyware)

Some Principles

1. It is a system
  • Good analogies found in real life systems
2. Technology changes everything
  • Look for the inflection points!
    • Stemming from, or leading to, the "killer app"
    • Generally, someone's trying to MAKE money or SAVE
    money by doing something differently
3. Priorities should not be mistaken for stupidity.
  • Choose 2 of [Soon, Good, Cheap.]
4. Things often left for “later”
  • Testability, Usability, Manageability, Security.
Rules - Continued

5. Code and Data are a matter of perspective.
6. MECHANISM and POLICY are generally separable
7. Simple solutions are generally better than complex ones.
   • “Simple and good enough” almost always more attractive than optimal.
   • Always consider the “Do nothing” option.
   • Static solutions are generally simpler than dynamic ones
8. In general, the future is unknowable, except in retrospect
   • It’s a good bet though that it looks like the past
9. Laziness in the presence of uncertainty is generally rewarded
   • And punished in its absence
10. Often, all it takes is the the right level of indirection.
11. Space and time trade off
12. 2x rarely matters, 10x almost always does.
13. Scalability is hard, but rewarded in time.
14. Performance is easier than most other things

CSE 451

• In this class we will learn:
  – what are the major components of most OS’s?
  – how are the components structured?
  – what are the most important (common?) interfaces?
  – what policies are typically used in an OS?
  – what algorithms are used to implement policies?
• Philosophy
  – you may not ever build an OS
  – but as a computer scientist or computer engineer you need to understand the foundations
  – most importantly, operating systems exemplify the sorts of engineering design tradeoffs that you’ll need to make throughout your careers – compromises among and within cost, performance, functionality, complexity, schedule …