Principles of Computer Systems

Lorenzo Alvisi
Chao Xie

POX

- How do we construct systems that are
  - reliable
  - portable
  - efficient
  - secure

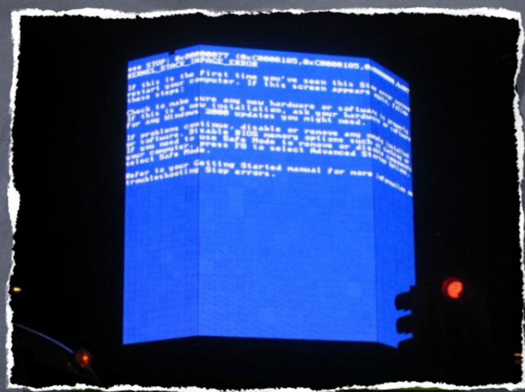
?

Meet the OS

- Software that manages a computer's resources
  - makes it easier to write the applications you want to write
  - makes you want to use the applications you wrote by running them efficiently

Why study Operating Systems?

- To learn how computers work
- To learn how to manage complexity through appropriate abstractions
  - infinite CPU, infinite memory, files, semaphores, etc.
- To learn about system design
  - performance vs. simplicity, HW vs. SW, etc.
- Because OSs are everywhere!



Where's the OS?

Las Vegas

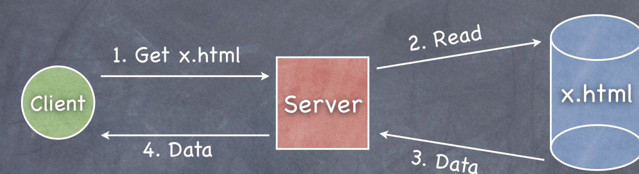
Where's the OS?

New York

Why study Operating Systems?

Because you are worth it!

Running a Web Server



- How does the OS
  - allow multiple applications to communicate with each other?
  - handle multiple concurrent requests?
  - support access to shared data (such as the cache)?
  - protect against malicious scripts?
  - enable different apps to share the data they have produced?
  - support consistent changes to complex data structures?
  - handle clients and servers of different speed?
  - transparently move to more powerful hardware?

Three steps to transmitting POX

1. How to approach problems
   - fundamental issues
     - coordination, abstraction
   - design space
   - case studies
- Goal: Forever mutate your brain (Mwahahahaahhaha!)
- Timescale: Big, long-term payoff

Three steps to transmitting POX

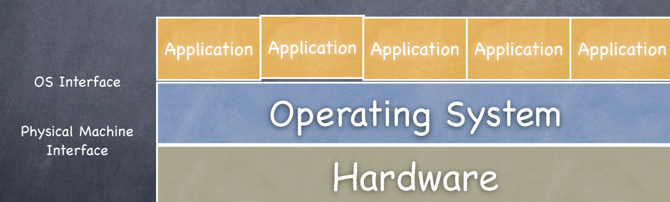
2. How to apply specific techniques
   - Time-tested solutions to hard problems
   - Hacking will not succeed
     - concurrent programming, transactions, etc
- Goal: Be a good engineer (Mwahahahaahhaha!)
- Timescale: Now — and in 20 years

Three steps to transmitting POX

3. How, in detail, current OSs work
   - FS, network stack, internal data structures, VM... of
     - MacOS, Linux, iOS, Windows
- Goal: Well...now in detail how current OSs work!
- Timescale: Better be now, because all will change tomorrow

What is an OS?

- An Operating System implements a virtual machine whose interface is more convenient* that the raw hardware interface



* easier to use, simpler to code, more reliable, more secure...

More than one hat

- Referee
- Illusionist
- Glue

More than one hat

- Referee
  - Manages shared resources such as CPU, memory, disks, networks, displays, cameras, etc.
- Illusionist
  - Look! Infinite memory! Your own private processor!
- Glue

More than one hat

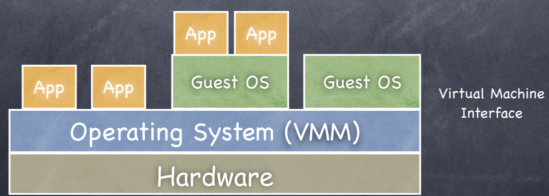
- Referee
  - Manages shared resources such as CPU, memory, disks, networks, displays, cameras, etc.
- Illusionist
  - Look! Infinite memory! Your own private processor!
- Glue
  - Offers a set of common services (e.g. U.I. routines)
  - Separates apps from I/O devices

OS as a referee

- Resource allocation
  - When multiple concurrent tasks, how does OS decide who gets how much?
- Isolation
  - A faulty app should not disrupt other apps or OS
    - OS must export less than full power of underlying hardware
- Communication
  - Apps need to coordinate and share state
    - Web site: select ads, cache recent data, fetch/merge data from disk, etc

OS as an illusionist

- Illusion of resources that are not physically present
  - Virtualization
    - processor, memory, screen space, disk, network
    - We can virtualize the entire computer!
      - ease of debugging, portability, isolation



OS as an illusionist

- Illusion of resources that are not physically present
  - Atomic operations
    - hardware guarantees atomicity at the word level
      - what happens during concurrent updates to complex data structures?
      - what if computer crashes during a block write?

OS as a glue

- Offers standard services to simplify app design and facilitate sharing
  - send/receive of byte streams
  - read/write files
  - pass messages
  - share memory
- Decouples hardware and app development
  - ...but database may need to be aware of specific disk drive

What makes a good OS?

- Reliability
  - OS does exactly what is designed to do
- Security
  - OS cannot be compromised by a malicious attacker
- Portability
  - OS does not change as hardware changes
- Performance
  - efficiency, overhead, fairness, latency, throughput, predictability
- Adoption
  - Are applications ported to the OS widely available?
  - Is hardware supported by the OS widely available?

Reliability

- The ability of a computer-related hardware or software component to consistently perform according to its specifications.
- In theory, a reliable product is totally free of technical errors (yeah, right)
- Availability: percentage of time system is useful
  - Depends on MTTF and MTTR

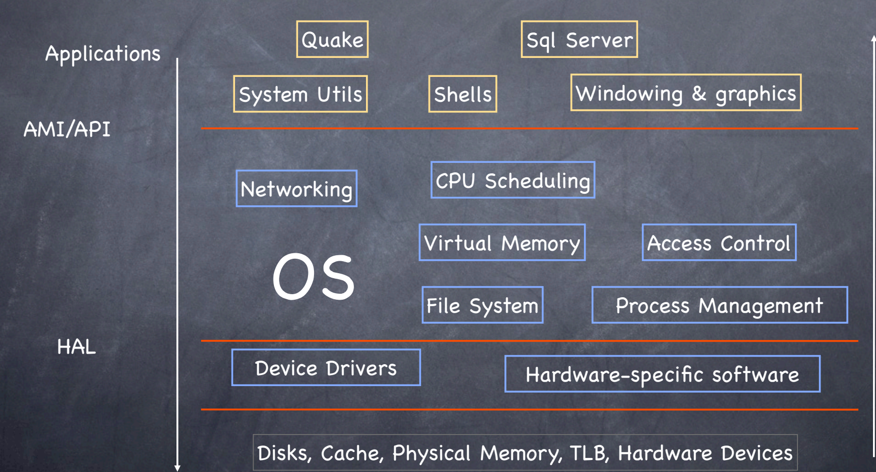
Security

- Includes privacy: data on the computer only accessible to authorized users
- Strong fault isolation helps, but not enough
  - Email gives no strong assurance of sender's identity
  - Security mechanisms should not prevent legitimate sharing!
- Enforcement mechanism
  - Ensures only permitted actions are allowed
- Security policy
  - Defines what is permitted

Portability

- OSs can live more than your cat!
  - must support applications not yet written
  - must run on hardware not yet developed
- Three interfaces
  - Abstract Machine Interface (AMI)
    - between OS and apps: API + memory access model + legally executable instructions
  - Application Programming Interface (API)
    - function calls provided to apps
  - Hardware Abstraction Layer (HAL)
    - abstracts hardware internally to the OS

Logical OS Structure

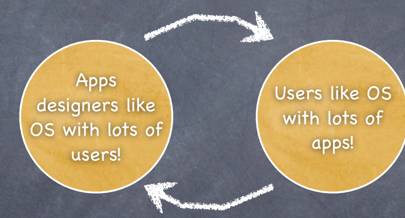


Performance

- Efficiency/Overhead
  - how much is lost by not running on bare hardware?
- Fairness
  - how are resources divided?
- Response time
  - how long does a task take to complete?
- Throughput
  - how many tasks complete per unit of time?
- Predictability
  - are performance metrics consistent over time?

Adoption

- Network effect



- Proprietary or Open?

A Short History of Operating Systems

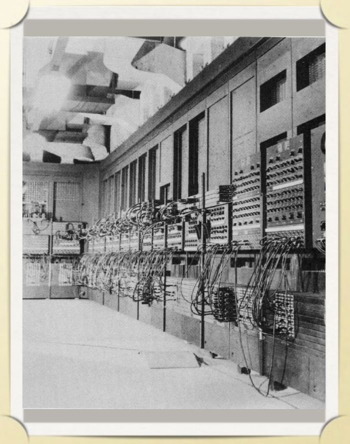
History of Operating Systems: Phases

- Phase 1: Hardware is expensive, humans are cheap
  - User at console: single-user systems
  - Batching systems
  - Multi-programming systems

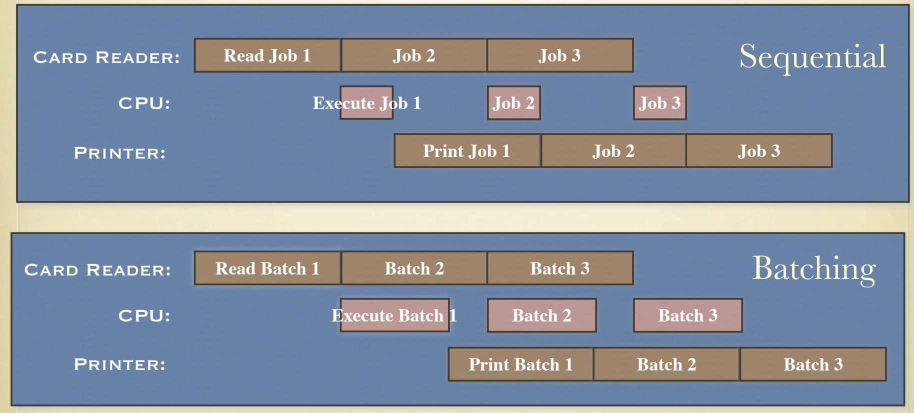
Hand programmed machines (1945-1955)

- Single user systems
- OS = loader + libraries of common subroutines
- Problem: low utilization of expensive components

$$\frac{\textit{time device busy}}{\textit{observation interval}} = \%\ \textit{utilization}$$

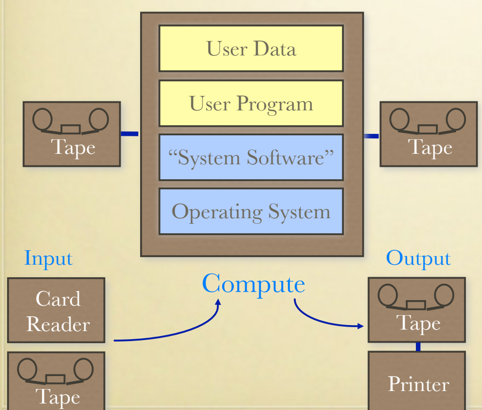


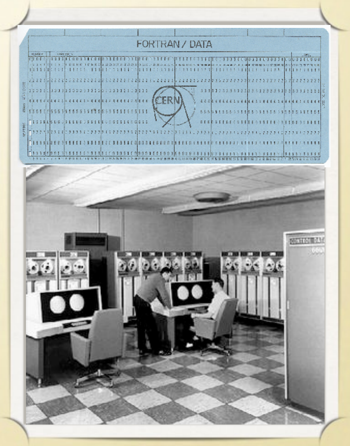
Batch/Off-line processing (1955-1965)



Batch Processing (1955-1965)

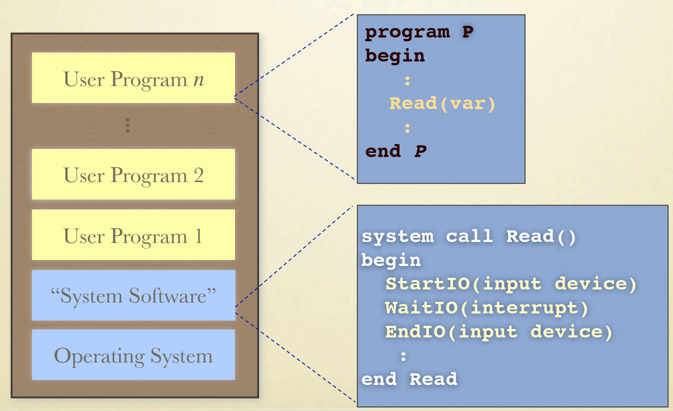
Operating system = loader + sequencer + output processor





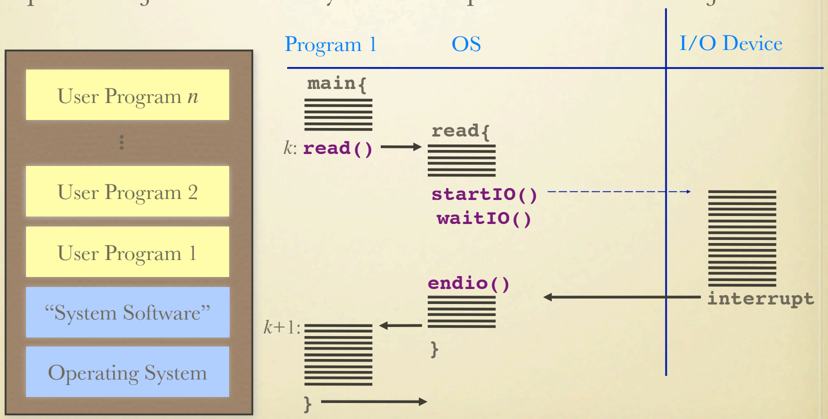
Multiprogramming (1965-1980)

Keep several jobs in memory and multiplex CPU between jobs



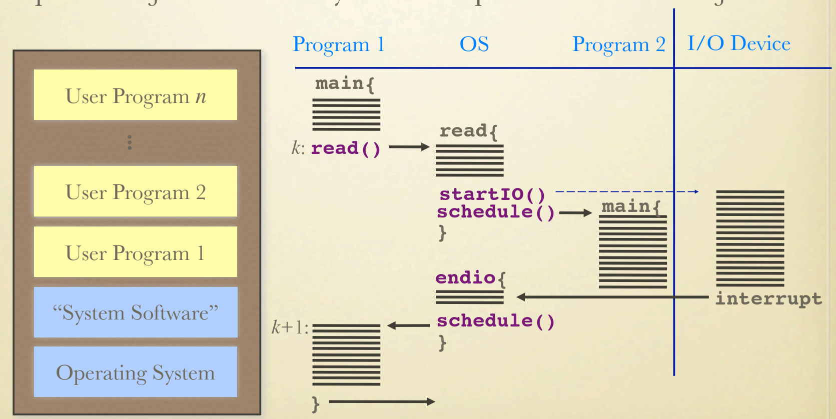
Multiprogramming (1965-1980)

Keep several jobs in memory and multiplex CPU between jobs



Multiprogramming (1965-1980)

Keep several jobs in memory and multiplex CPU between jobs

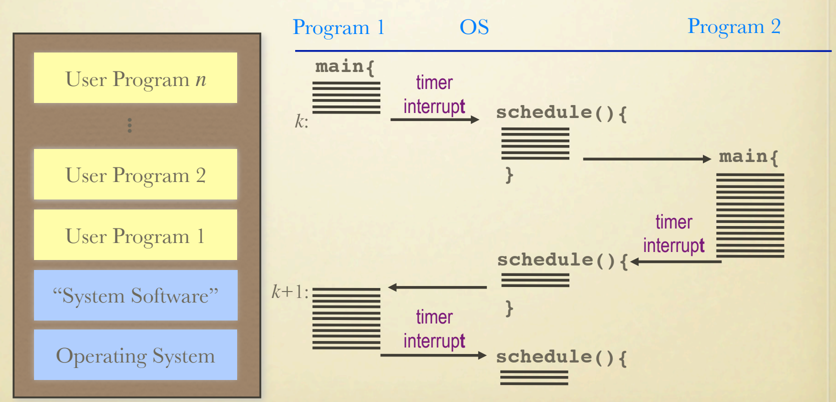


History of Operating Systems: Phases

- Phase 1: Hardware is expensive, humans are cheap
  - User at console: single-user systems
  - Batching systems
  - Multi-programming systems
- Phase 2: Hardware is cheap, humans are expensive
  - Time sharing: Users use cheap terminals and share servers

Timesharing (1970-)

A timer interrupt is used to multiplex CPU between jobs



History of Operating Systems: Phases

- Phase 1: Hardware is expensive, humans are cheap
  - User at console: single-user systems
  - Batching systems
  - Multi-programming systems
- Phase 2: Hardware is cheap, humans are expensive
  - Time sharing: Users use cheap terminals and share servers
- Phase 3: Hardware is very cheap, humans are very expensive
  - Personal computing: One system per user
  - Distributed computing: many systems per user
  - Ubiquitous computing: LOTS of systems per users

Operating Systems for PCs

Personal computing systems

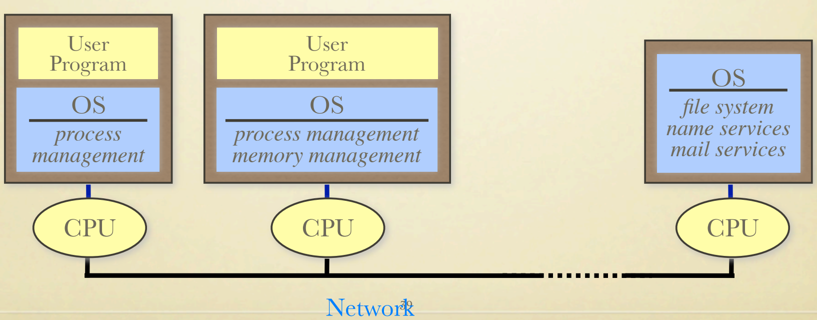
- Single user
- Utilization is no longer a concern
- Emphasis is on user interface and API
- Many services & features not present

Evolution

- Initially: OS as a simple service provider (simple libraries)
- Now: Multi-application systems with support for coordination

Distributed Operating Systems

- Abstraction: present a multi-processor system as a single processor one.
- New challenges in consistency, reliability, resource management, performance, etc.
- Examples: SANs, Oracle Parallel Server



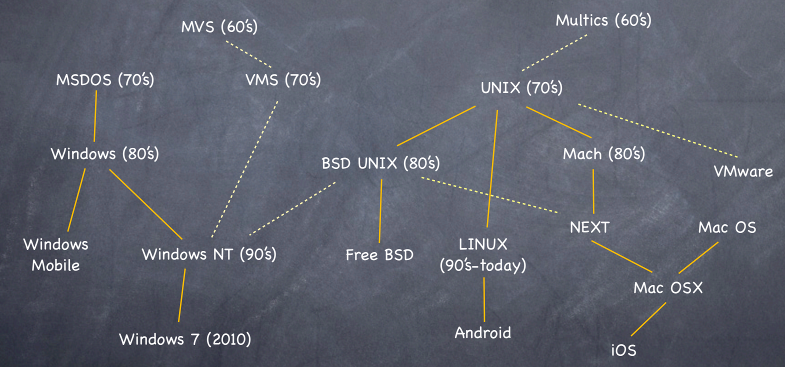
Ubiquitous Computing

- PDAs, cellular phones, sensors
- Challenges
  - Small memory size
  - Slow processor
  - Battery concerns
  - Scale
  - Security
  - Naming

History of Operating Systems: Phases

- Phase 1: Hardware is expensive, humans are cheap
  - User at console: single-user systems
  - Batching systems
  - Multi-programming systems
- Phase 2: Hardware is cheap, humans are expensive
  - Time sharing: Users use cheap terminals and share servers
- Phase 3: Hardware is very cheap, humans are very expensive
  - Personal computing: One system per user
  - Distributed computing: many systems per user
  - Ubiquitous computing: LOTS of systems per user
- Richer Services
  - Real-time operating systems

Genealogy of modern Operating Systems



Cambia, Todo Cambia

- Nothing wrong with batch systems
  - They were right for tradeoffs at the time
- But tradeoffs change...

	1981	1996	2011	Factor
MIPS	1	300	10000	10K
$/MIPS	$100K	$30	$0.50	200K
DRAM	128KB	128MB	10GB	100K
Disk	10MB	4GB	1TB	100K
Home Internet	9.6Kbps	256 Kbps	5Mbps	500
LAN Network	3Mbps (shared)	10 Mbps	1Gbps	300
# Users	100	100 Mb/s	<<1	100+