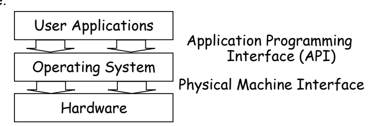
Operating Systems:
Basic Concepts and History

Don Porter
Portions courtesy Emmett Witchel

Introduction to Operating Systems

- An operating system is the interface between the user and the architecture.



- **OS as juggler:** providing the illusion of a dedicated machine with infinite memory and CPU.
- **OS as government:** protecting users from each other, allocating resources efficiently and fairly, and providing secure and safe communication
- **OS as complex system:** keeping OS design and implementation as simple as possible is the key to getting the OS to work

What is an Operating System?

- Any code that runs with the hardware kernel bit set
  - An abstract virtual machine
  - A set of abstractions that simplify application design
    - Files instead of "bytes on a disk"
- Core OS services, written by "pros"
  - Processes, process scheduling
  - Address spaces
  - Device control
  - ~30% of Linux source code. Basis of stability and security
- Device drivers written by "whoever"
  - Software run in kernel to manages a particular vendor's hardware
    - E.g. Homer Simpson doll with USB
  - ~70% of Linux source code
  - OS is extensible
  - Drivers are the biggest source of OS instability

What is an Operating System?

- For any OS area (CPU scheduling, file systems, memory management), begin by asking two questions
  - What's the hardware interface? (The Physical Reality)
  - What is the application interface? (The Nicer Interface for programmer producivity)
- Key questions:
  - Why is the application interface defined the way it is?
  - Should we push more functionality into applications, the OS, or the hardware?
  - What are the tradeoffs between programmability, complexity, and flexibility?

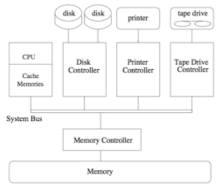
Operating System Functions

- *Service provider*
  - Provide standard facilities
    - File system
    - Standard libraries
    - Window system
    - …
- *Coordinator*: three aspects
  - Protection: prevent jobs from interfering with each other
  - Communication: enable jobs to interact with each other
  - Resource management: facilitate sharing of resources across jobs.
- Operating systems are everywhere
  - Single-function devices (embedded controllers, Nintendo, …)
    - OS provides a collection of standard services
    - Sometimes OS/middleware distinction is blurry
  - Multi-function/application devices (workstations and servers)
    - OS manages application interactions

Why do we need operating systems?

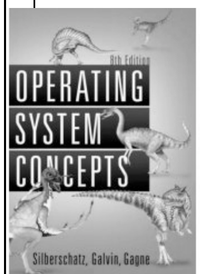
- Convenience
  - Provide a high-level abstraction of physical resources.
    - Make hardware usable by getting rid of warts & specifics.
  - Enable the construction of more complex software systems
  - Enable portable code.
    - MS-DOS version 1 boots on the latest Intel Core.
    - Would games that ran on MS-DOSv1 work well today?
- Efficiency
  - Share limited or expensive physical resources.
  - Provide protection.

Computer Architecture & Processes



- CPU - the processor that performs the actual computation
- I/O devices - terminal, disks, video board, printer, etc.
- Memory - RAM containing data and programs used by the CPU
- System bus - the communication medium between the CPU, memory, and peripherals

Evolution?



- What does this book cover imply to you?
- Do OSes evolve? How?
  - New hardware
    - Multi-core, GPUs, power management
  - New applications
    - Cloud, mobile apps, games, VoIP

Evolution of Operating Systems

- Why do operating systems change?
  - Key functions: hardware abstraction and coordination
  - Principle: Design tradeoffs change as technology changes.
- Comparing computing systems from 1981 and 2007

	1981	2007	Factor
MIPS	1	57,000	57,000
$/SPECInt	$100K	$2	50,000
DRAM size	128KB	2GB	16,000
Disk size	10MB	1TB	100,000
Net BW	9600 bps	100 Mb/s	10,000
Address bits	16	64	4

- Energy efficiency and parallelism loom on the horizon
  - Data centers consume ~3% of US energy
  - No more single-core CPUs

From Architecture to OS to Application, and Back

Hardware	Example OS Services	User Abstraction
Processor	Process management, Scheduling, Traps, Protections, Billing, Synchronization	Process
Memory	Management, Protection, Virtual memory	Address space
I/O devices	Concurrency with CPU, Interrupt handling	Terminal, Mouse, Printer, (System Calls)

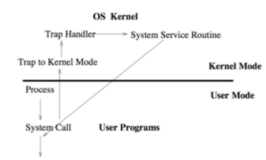
From Architectural to OS to Application, and Back

OS Service	Hardware Support
Protection	Kernel / User mode Protected Instructions Base and Limit Registers
Interrupts	Interrupt Vectors
System calls	Trap instructions and trap vectors
I/O	Interrupts or Memory-Mapping
Scheduling, error recovery, billing	Timer
Synchronization	Atomic instructions

Interrupts - Moving from Kernel to User Mode

User processes may not:

- address I/O directly
- use instructions that manipulate OS memory (e.g., page tables)
- set the mode bits that determine user or kernel mode
- disable and enable interrupts
- halt the machine



but in kernel mode, the OS does all these things

- a status bit in a protected processor register indicates the mode
- Protected instructions can only be executed in kernel mode.
- On interrupts (e.g., time slice) or system calls

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: lots of systems per user
- Phase 4: Ubiquitous computing/Cloud computing
  - Cell phone, mp3 player, DVD player, TIVO, PDA, iPhone, eReader
  - Software as a service, Amazon's elastic compute cloud

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: lots of systems per user
- Phase 4: Ubiquitous computing

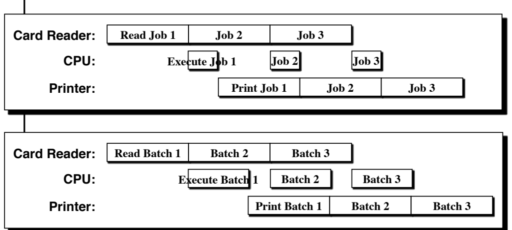
A Brief History of Operating Systems
Hand programmed machines ('45-'55)

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

$$\frac{\textit{Execution time}}{\textit{Execution time} + \textit{Card reader time}}$$

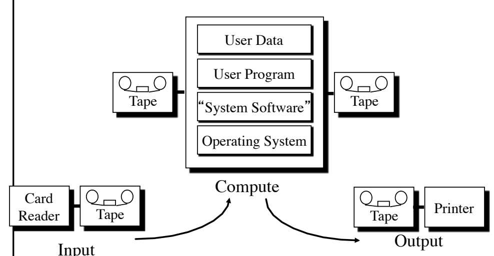
Batch/Off-line processing ('55-'65)

- Batching *v.* sequential execution of jobs



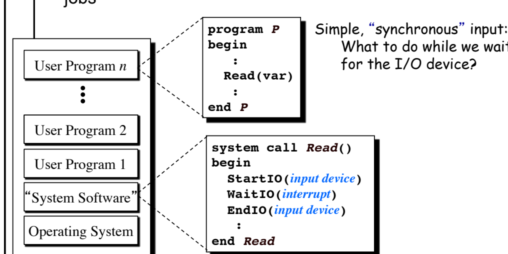
Batch processing ('55-'65)

- Operating system = *loader* + *sequencer* + *output processor*



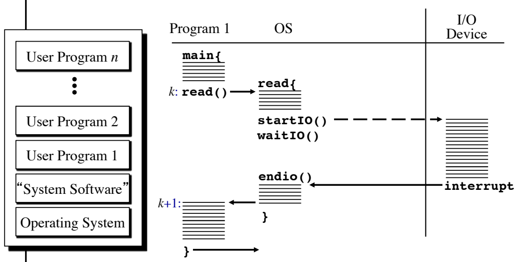
Multiprogramming ('65-'80)

- Keep several jobs in memory and multiplex CPU between jobs



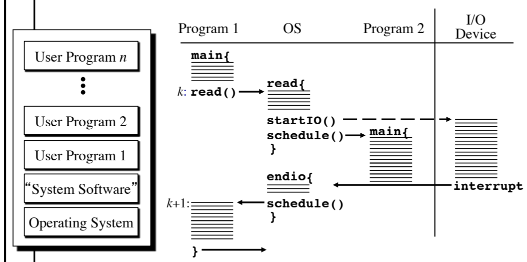
Multiprogramming (‘65- ‘80)

- Keep several jobs in memory and multiplex CPU between jobs



Multiprogramming (‘65- ‘80)

- Keep several jobs in memory and multiplex CPU between jobs

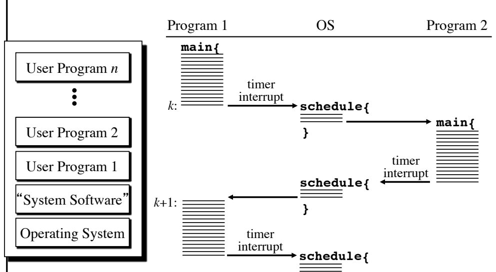


History of Operating Systems: Phases

- 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
- Phase 3: Hardware is very cheap, humans are very expensive
  - Personal computing: One system per user
  - Distributed computing: lots of systems per user
- Phase 4: Ubiquitous computing

Timesharing (‘70-)

- A timer interrupt is used to multiplex CPU among jobs

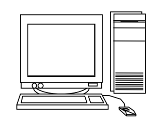


History of Operating Systems: Phases

- 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: lots of systems per user
- Phase 4: Ubiquitous computing

Operating Systems for PCs

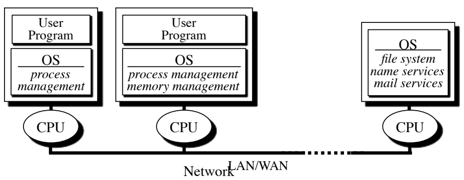
- Personal computing systems
  - Single user
  - Utilization is no longer a concern
  - Emphasis is on user interface and API
  - Many services & features not present



- Evolution
  - Initially: OS as a simple service provider (simple libraries)
  - Now: Multi-application systems with support for coordination and communication
  - Growing security issues (e.g., online commerce, medical records)

Distributed Operating Systems

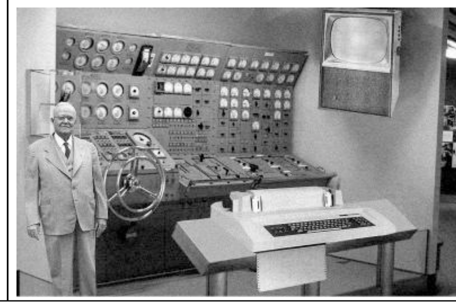
- Typically support distributed services
  - Sharing of data and coordination across multiple systems
- Possibly employ multiple processors
  - Loosely coupled *v.* tightly coupled systems
- High availability & reliability requirements
  - Amazon, CNN



Increasing importance of security

- Older OSes (including Unix) were not designed with security as a big concern. Why not?
  - Users were typically employees at a company, external consequences for bad behavior
  - Programmers and system designers could assume users would generally "do the right thing", but may make honest mistakes
- What changed in the 90s?
  - The internet!
  - Lots of computers administered by amateurs
  - Connected to mean people all over the world
  - Programs and systems have to defend against abuse

In the year 2000…



History of Operating Systems: Phases

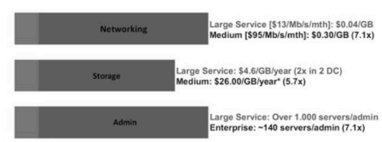
- Phase 1: Hardware is expensive, humans are cheap
  - User at console: single-user systems
  - Batching systems
  - Multi-programming systems
- Phase 2: Hardware is cheap, humans are expensive
  - Time sharing: Users use cheap terminals and share servers
- Phase 3: Hardware is very cheap, humans are very expensive
  - Personal computing: One system per user
  - Distributed computing: lots of systems per user
- Phase 4: Ubiquitous computing/Cloud computing
  - Everything will have computation, from pacemakers to toasters
  - Computing centralizing
  - "I think there is a world market for maybe five computers" – Tomas J. Watson, 1943 (president of IBM)

What is cloud computing?

- **Cloud computing** is where dynamically scalable and often virtualized resources are provided as a service over the Internet (thanks, wikipedia!)
- Infrastructure as a service (IaaS)
  - Amazon's EC2 (elastic compute cloud)
- Platform as a service (PaaS)
  - Google gears
  - Microsoft azure
- Software as a service (SaaS)
  - gmail
  - facebook
  - flickr

Services Economies of Scale

- Substantial economies of scale possible
- 2006 comparison of very large service with small/mid-sized: (~1000 servers):



- High cost of entry
  - Physical plant expensive: 15MW roughly $200M
- Summary: significant economies of scale but at very high cost of entry
  - Small number of large players likely outcome

Thanks, James Hamilton, amazon

Multi-core

- New hotness in CPU design. Not going away.
  - Why?
- Being able to program with threads and concurrent algorithms will be a crucial job skill going forward
  - Don't leave SBU without mastering these skills
  - We will do some thread programming in Lab 3

Editorial on 2.4

- Textbook implies modern OSes are microkernels
- This is false
  - Windows NT and OSX were designed as microkernels
  - Then reverted to essentially monolithic designs in practice
- Linux was never a microkernel
  - Google the famous Torvalds v Tanenbaum debate
- Similarly, Distributed OSes are mostly abandoned
  - I think cloud and other distributed systems are better described as loose "confederations" of systems

2.4: Object orientation

- Objects are a key feature of the Windows NT kernel design
  - IMO a good idea
- Linux actually has its own bizarre version of object orientation using C structs and function pointers
  - In Unix, everything is a file
  - How did they pull this off?
  - Poor-man's object inheritance

Richer Operating Systems
Information organization

- Is it better to search for data (google), or organize it hierarchically (file folders)?
  - Organization along a particular set of ideas (schema) might not be ideal for a different set of ideas.
  - Gmail search vs. mail folders
- Integration of search in Vista and MacOS.
  - Do you use My Documents folder, or do you maintain your own directories? use both a lot?

Course Overview

- OS Structure, Processes and Process Management
- CPU scheduling
- Threads and concurrent programming
  - Thread coordination, mutual exclusion, monitors
  - Deadlocks
- Disks & file systems
  - Distributed file systems
- Virtual memory & Memory management
- Security