

Operating Systems Design

3. Definitions, Concepts, and Architecture

Paul Krzyzanowski
pxk@cs.rutgers.edu

Mechanisms & Policies

OS Mechanisms & Policies

- Mechanisms:
 - Presentation of a software abstraction:
 - Memory, data blocks, network access, processes
- Policies:
 - Procedures that define the behavior of the mechanism
 - Allocation of memory regions, replacement policy of data blocks
 - Permissions
- Keep mechanisms, policies, and permissions separate

Processes

- Mechanism:
 - Create, terminate, suspend, switch, communicate
- Policy
 - Who is allowed to create and destroy processes?
 - What is the limit?
 - What processes can communicate?
 - Who gets priority?

Threads

- Mechanism:
 - Create, terminate, suspend, switch, synchronize
- Policy
 - Who is allowed to create and destroy threads?
 - What is the limit?
 - How do you assign threads to processors?
 - How do you schedule the CPU among threads of the same process?

Virtual Memory

- Mechanism:
 - Logical to physical address mapping
- Policy
 - How do you allocate physical memory among processes and among users?
 - How do you share physical memory among processes?
 - Whose memory do you purge when you're running low?

File Systems

- Mechanism:
 - Create, delete, read, write, share files
 - Manage a cache; memory map files
- Policy
 - What protection mechanisms do you enforce?
 - What disk blocks do you allocate?
 - How do you manage cached blocks of data (Per file? Per user? Per process?)

Messages

- Mechanism:
 - Send, receive, retransmit, buffer bytes
- Policy
 - Congestion control, dropping packets, routing, prioritization, multiplexing

Character Devices

- Mechanism:
 - Read, write, change device options
- Policy
 - Who is allowed to access the device?
 - Is sharing permitted?
 - How do you schedule device access?

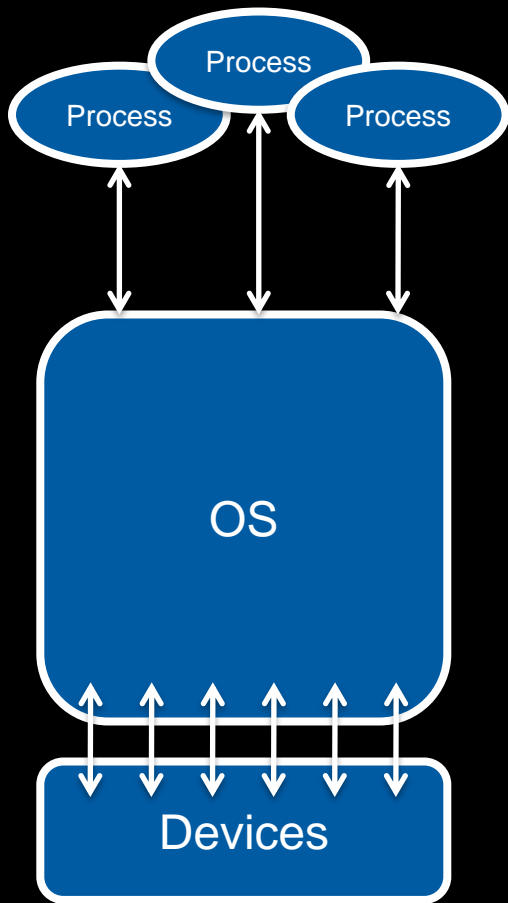
Definitions, Concepts, and Architecture

What is an operating system?

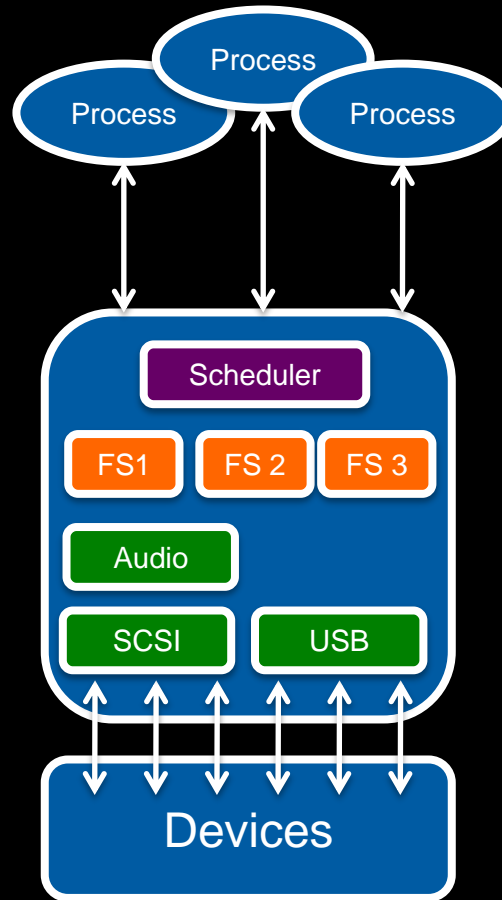
- The first program
- A program that lets you run other programs
- A program that provides controlled access to resources:
 - CPU
 - Memory
 - Display, keyboard, mouse
 - Persistent storage
 - Network

This includes: naming, sharing, protection, communication

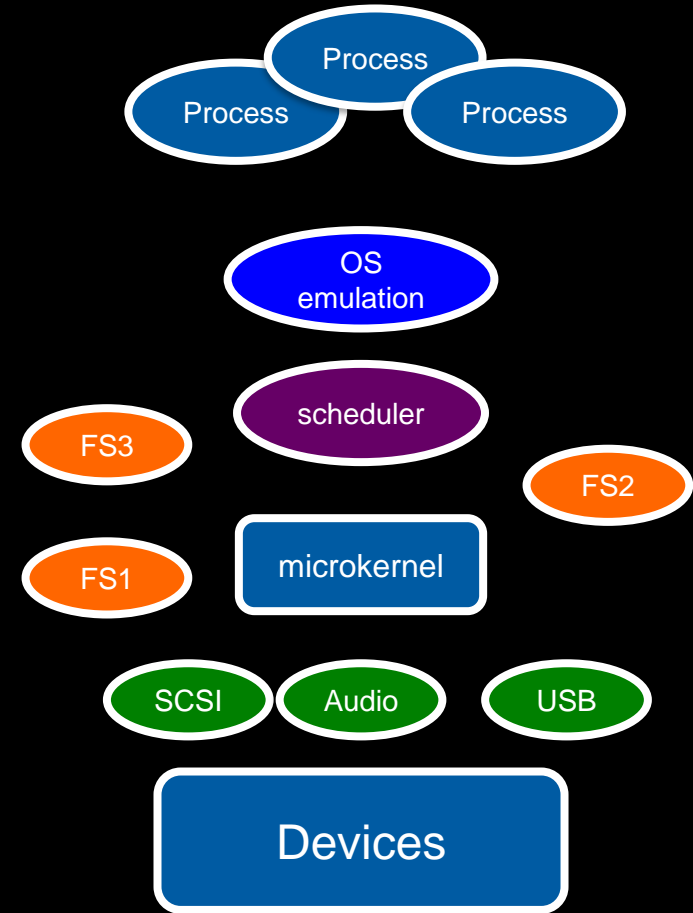
OS Structure



Monolithic



Modular



Microkernel

What's a kernel?

- **Operating System**
 - Often refers to the complete system, including command interpreters, utility programs, window managers, ...
- **Kernel**
 - Core component of the system that manages resource access, memory, and process scheduling

UNIX Kernel (example)

Some of the things it does:

- Controls execution of processes
 - Creation, termination, communication
- Schedules processes for execution on the CPU(s)
- Manages memory
 - Allocates memory for an executing process
 - Sets memory protection
 - Coordinates swapping pages of memory to a disk if low on memory
- Manages a file system
 - Allocation and retrieval of disk data
 - Enforcing access permissions & mutual exclusion
- Provides access to devices
 - Disk drives, networks, keyboards, displays, printers, ...
 - Enforces access permissions & mutual exclusion

User Mode vs. Kernel Mode

- **Kernel mode** = privileged, system, supervisor mode
 - Access restricted regions of memory
 - Modify the memory management unit
 - Set timers
 - Define interrupt vectors
 - Halt the processor
 - Etc.
- CPU knows what mode it's in via a status register
 - You can set the register in kernel mode
 - OS & boot loaders run in kernel mode
 - User programs run in user mode

Violations

- What if a CPU tries to execute something that is available only in kernel mode?
 - (a) nothing, or (more likely)
 - (b) trap (exception)
 - Memory access violation
 - Illegal instruction violation
 - Register access violation
- The OS processes the trap
 - Original program counter is saved
 - OS decides on course of action
 - If needed, restart the offending instruction
- Traps occur:
 - Via software (e.g., INT instruction)
 - Because of an access violation
 - Via a hardware interrupt (e.g., timer)

How do you switch to kernel mode?

Software interrupts (traps)

- Trap vectors are set up in kernel mode (at boot time)
 - Trap pushes the return address on the stack and jumps to a well-known address
 - That address usually contains a *jump* instruction (vector) to the code that will handle that trap
- Returning back to user mode: *return from exception*

Mode Switch: switching between user & kernel mode

System Calls: Interacting with the OS

- Use *trap* mechanism to switch to the kernel
 - Mode switch
- Pass a number that represents the OS service
 - System call number; usually set in a register
- A system call involves:
 - Set system call number
 - Save parameters
 - Issue the trap (jump to kernel mode)
 - OS gets control
 - Return from exception (back to user mode)
 - Retrieve results and return them to the calling function
- System call interfaces are encapsulated as library functions

Interrupts & Preemption

- How do we ensure that the OS gets control?
- Program a timer interrupt
 - On Linux/Intel systems,
Set the 8254 Programmable Interval Timer to generate an interrupt (IRQ 0) approximately every 10 ms.
 - Since 2005: **High Precision Event Timer** (HPET) replaces 8254

Context switch & Mode switch

- An interrupt or trap results in a *mode switch*
 - CPU switches execution from user mode to kernel mode
- An operating system may save a process' state and restore another process' state.
 - **Context switch**
 - Save all registers (including stack pointers, PC, and flags)
 - Load saved registers (including SP, PC, flags)
 - To return to original context: restore registers and return from exception
- **Context switch**: switch to kernel mode, save state so that it can be restored later and reload another process' saved state

Devices

- Character: mice, keyboard, audio, scanner
 - *Byte streams*
- Block: disk drives, flash memory
 - *Addressable blocks (suitable for caching)*
- Network: ethernet & wireless networks
 - *Packet based I/O*
- Bus controllers
 - *Interface with communication busses*

Interacting with devices

- Devices have command registers
 - *Transmit, receive, data ready, read, write, seek, status*
- **Memory mapped I/O**
 - Map device registers into memory
 - Memory protection now protects device access
 - Standard memory load/store instructions can be used to interact with the device

Getting data to/from devices

- When is the device ready?
 - **Polling**
 - Wait for device to be ready
 - To avoid busy loop, check each clock interrupt
 - **Interrupts from the device**
 - Interrupt when device has data or when the device is done transmitting
 - No checking needed – but context switch may be costly

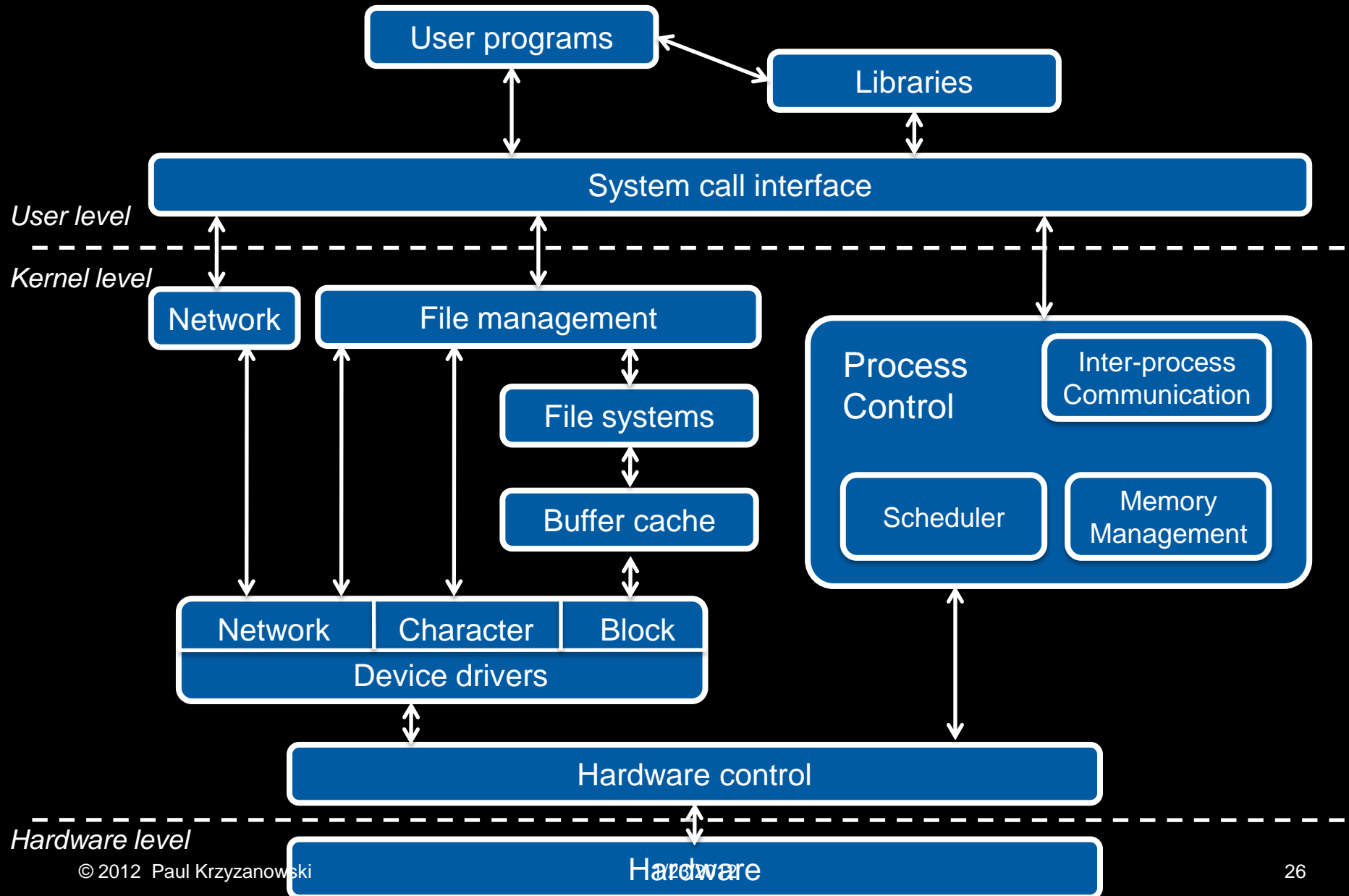
Getting data to/from devices

- How do you move data?
 - Programmed I/O (PIO)
 - Use memory-mapped device registers
 - The processor is responsible for transferring data to/from the device by writing/reading these registers
 - DMA
 - Allow the device to access system memory directly

Files and file systems

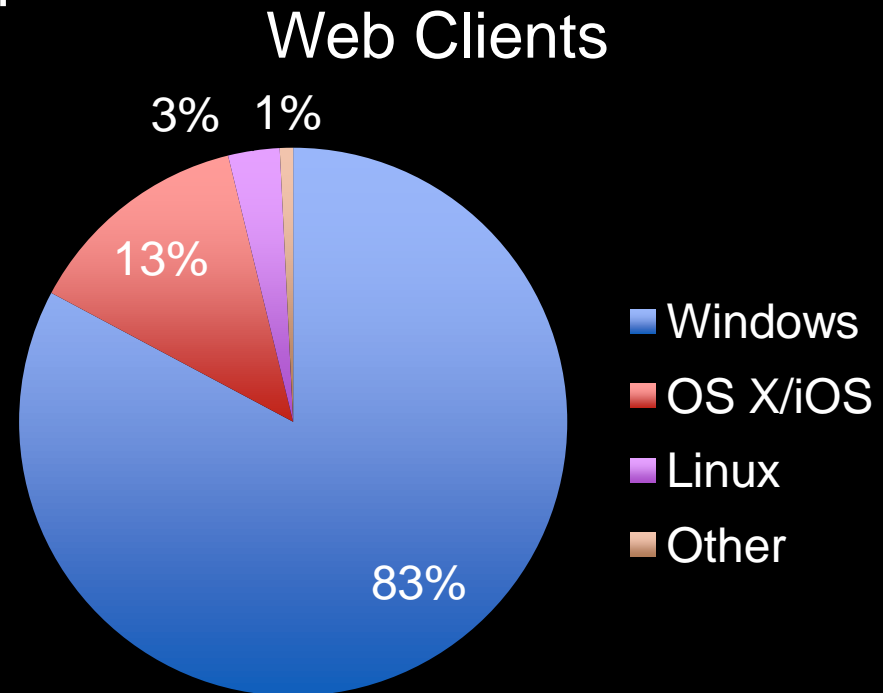
- Persistent storage of data
 - Handle allocation of disk space
- Provide user-friendly names to identify the data
- Associate attributes with the data
 - Create time, access time, owner, permissions, ...
 - Device or data file?

Structure of an operating system



Which OS?

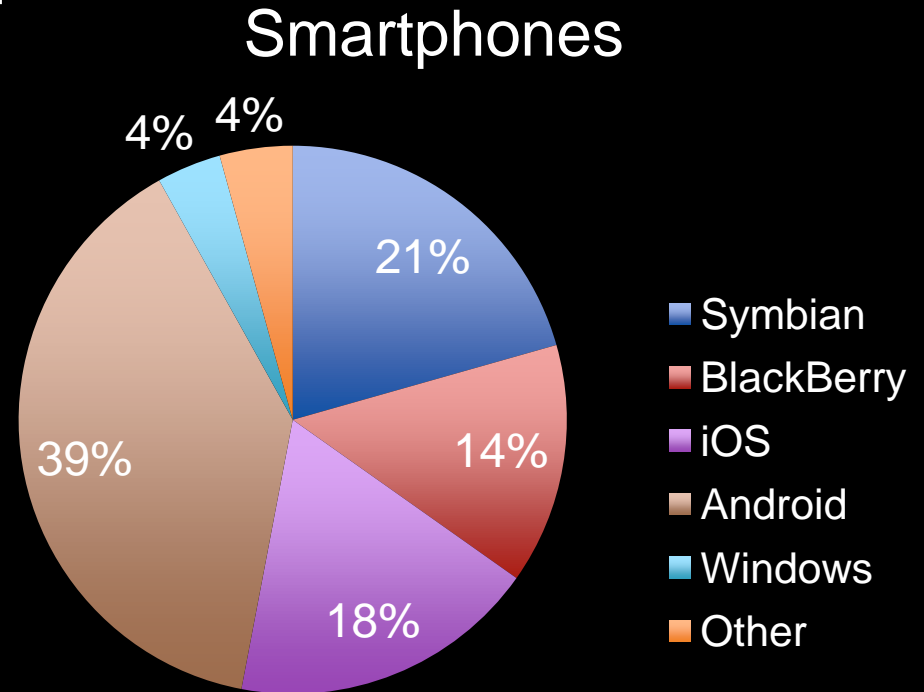
- Hundreds to choose from
- Dominant ones:
 - Windows NT family
 - Windows 7
 - Vista, XP
 - Windows Server 20xx
 - Mac OS X
 - Includes iOS
 - Linux
 - Includes Android
 - Symbian



Source: W3Counter data from http://en.wikipedia.org/wiki/Usage_share_of_operating_systems

Which OS?

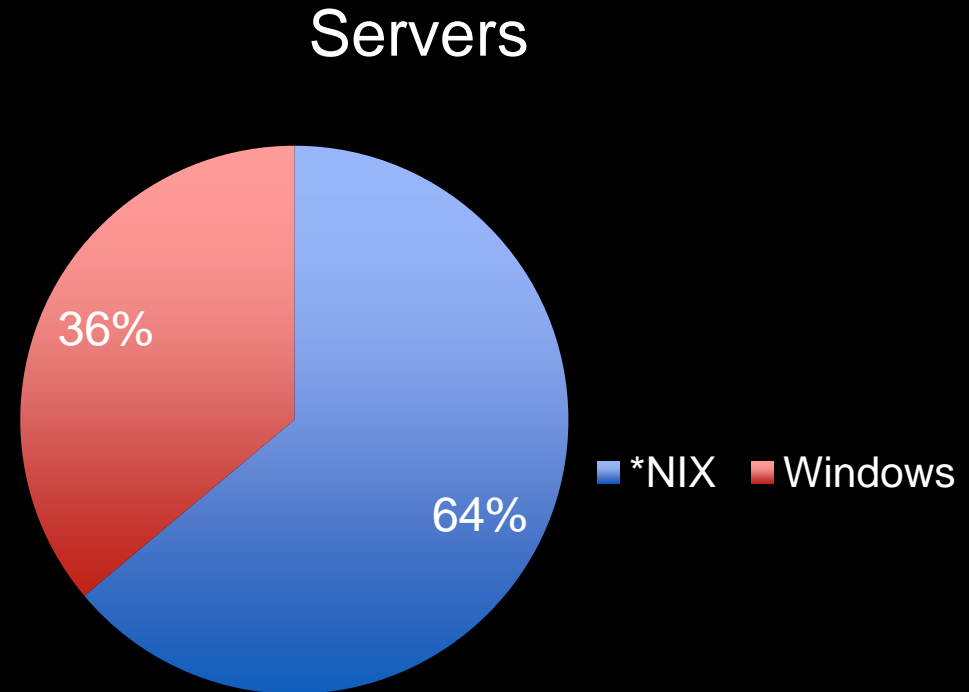
- Hundreds to choose from
- Dominant ones:
 - Windows NT family
 - Windows 7
 - Vista, XP
 - Windows Server 20xx
 - Mac OS X
 - Includes iOS
 - Linux
 - Includes Android
 - Symbian



Source: IDC data from http://en.wikipedia.org/wiki/Usage_share_of_operating_systems

Which OS?

- Hundreds to choose from
- Dominant ones:
 - Windows NT family
 - Windows 7
 - Vista, XP
 - Windows Server 20xx
 - Mac OS X
 - Includes iOS
 - Linux
 - Includes Android
 - Symbian



Source: W3Techs data from http://en.wikipedia.org/wiki/Usage_share_of_operating_systems

POSIX

- UNIX → POSIX
- IEEE (ISO/IEC 9945): defines POSIX environment
 - System interfaces
 - Shell & scripting interface
 - Common utilities
 - Networking interfaces
 - Security interfaces
- POSIX (or close to) systems include
 - Solaris, BSD, Mac OS X, VxWorks, Microsoft Windows Services for UNIX
 - Linux, FreeBSD, NetBSD, OpenBSD, BeOS

The End.