Operating Systems Structure

- How do user programs obtain services from the OS in a safe way?
- By what principles are OSes designed?

COSC 301: Operating Systems
10 September 2012

Joel Sommers
jsommers@colgate.edu
Colgate University

Learning Goals

- Define the roles and purposes of an operating system.
- Describe and contrast user mode and kernel mode in an operating system.
- Identify characteristics of how and describe why operating systems are structured as monolithic, microkernel, and virtualized systems.
- Effectively distinguish between “mechanism” and “policy” with respect to the design of an operating system.
- Describe what a system call is, and sketch the steps that take place to carry out a system call.

The OS as a Virtual Machine

- Physical machine
  - Runs 1 program on the CPU at a time
  - Has 1 physical memory
  - Has devices (like disks) that are really hard to use
- OS provides virtual machine illusion to programs
  - Many CPUs, separate and large memories for each program, easy use of devices
- Key issues: efficiency, protection

What is the role of the OS?

Role #1: Provide standard library to applications

- What is a *resource*?
  - Anything computationally valuable (*e.g.*, CPU, memory, disk, network)
- Advantages of a standard library
  - Allow applications to reuse common facilities
  - Make different devices appear the same
  - Provide higher-level abstractions
- Challenges
  - What are the right abstractions?
  - How much of the hardware should be exposed?

What is the role of the OS?

Role #2: Resource coordinator and allocator

- Advantages of a resource coordinator
  - Virtualize resources so multiple users or applications can share
  - Protect applications from one another
  - Provide efficient and fair access to resources
  - Can arbitrate among conflicting requests
- Challenges
  - What are the right *mechanisms*?
  - What are the right *policies*?

What functionality belongs in the OS?

No single right answer

- Desired functionality depends on outside factors
- OS must adapt to both user expectations and technology changes
  - Change abstractions provided to users
  - Change algorithms used to implement those abstractions
  - Change low-level implementation to deal with hardware
- Current operating systems driven by technology and application trends
- OS design is (almost always) about tradeoffs (performance, usability, security, generality, . . .)

OS components

Definition
Kernel: core components of the OS

- Process scheduler
  - process: a running program
  - Determines when and for how long each process executes
- Memory manager
  - Determines when and how memory is allocated to processes
  - Decides what to do when main memory is full
- File system
  - Organizes named collections of data in persistent storage
- Networking
  - Enables processes to communicate with one another

User Program and OS Interaction

- The OS kernel manages hardware resources, provides services via a standard set of libraries to user programs
  - User applications make system calls via the OS libraries to request kernel services
  - Certain CPU hardware features enable the OS to do this in a safe way
    - Example: user programs shouldn't be allowed to use the `halt` instruction without proper permissions
    - Example: user programs shouldn't be allowed to directly access hardware resources unless explicitly allowed by the OS
  - How the OS interacts with hardware devices depends on the goals of the OS, and capabilities of the hardware
  - The specific communication and/or software mechanisms used by the user applications to obtain library services depend on OS kernel design

Some important CPU features

Standard stuff:

- Basic cycle: fetch, decode, execute
- Instructions: load, store, register ops
- Superscalar pipelining

Special-purpose registers:

- Program counter (instruction pointer)
- Stack pointer (top of current stack)
- PSW (program status word)
  - Condition code bits (results of comparison instructions)
  - Mode bit(s): enables the OS to control access to hardware
    - User mode: only a subset of instructions/features are accessible
    - Kernel mode: all instructions and features are accessible
    - Some CPUs support more than 2 levels of protection

Context switching

- Whenever execution switches between protection domains, a context switch occurs
- Actions on context switch:
  - Save the PC, stack pointer, PSW (as well as any other vital CPU state)
  - Save the contents of all general purpose registers
  - Save and reprogram the memory-management unit registers (more later on what this is all about)
  - Wait while the instructions currently in the CPU pipeline are trashed
  - (Wait for instruction/data caches to load from new program's memory; details depend heavily on CPU architecture)
- Context switches can be expensive...

Operating Systems Structure
What is an operating system?
Hardware
System Calls
Designing an OS

Polling and Interrupts

- Two fundamental ways for an OS to interact with hardware and perform I/O: interrupts and polling

Interrupts:

- Hardware device interrupts CPU from its current task
  - Example: key is pressed on the keyboard
  - Example: packet arrives at a network interface
- OS maintains a list of routines to handle hardware I/O events (interrupt vector)
  - Upon hardware interrupt, OS saves CPU state, switches to kernel mode, uses device id that generated to interrupt to find the interrupt routine to execute
  - Interrupt routine calls into device driver to perform I/O
  - Control is eventually returned to previously executing process at the next instruction
- Interrupt-driven I/O is very commonly used in commodity hardware

Operating Systems Structure
What is an operating system?
Hardware
System Calls
Designing an OS

Polling

- OS checks whether there is I/O to be done on a particular device
- May check repeatedly in a tight loop if it expects input, or may check periodically or infrequently
  - Depends on OS goals, application requirements
- Polling is a common scenario in real-time systems that cannot suffer unpredictable delays due to interrupts
  - It can also improve performance when heavy I/O traffic is expected, *e.g.*, a web server that gets a lot of network traffic
- Hardware must support polling; not all commodity hardware does (need capability to disable interrupts)

Operating Systems Structure
What is an operating system?
Hardware
System Calls
Designing an OS

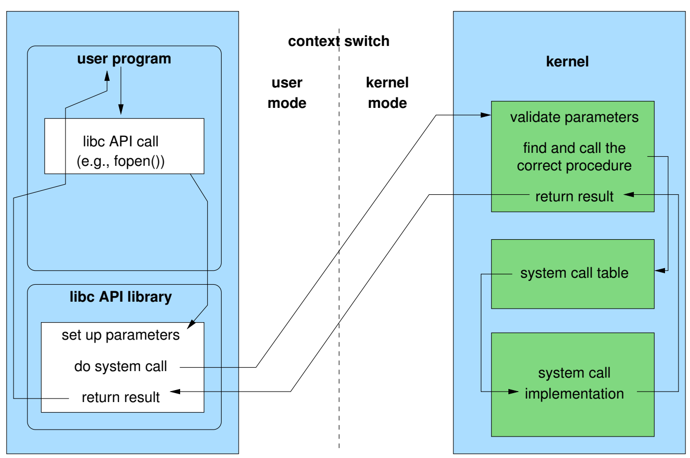
How does a user process ask the OS to perform a function for it?

- Scenario:
  - User program is running with CPU mode in user mode
  - User program needs to obtain information from system hardware, or change the state of system hardware (e.g., perform I/O), or obtain/modify information maintained by the OS (e.g., information about other system processes)
- Problem: user program cannot directly access kernel memory or hardware with the CPU running in user-mode
- Must make a system call to ask the OS to perform some action on its behalf
  - *e.g.*, `fopen()`, `getrusage()`, `fgets()`, ...
  - Can use `strace` to trace syscalls in any executable

Traps

- A system call is the mechanism by which a user process calls a kernel procedure
  - Used to do I/O, read/write files, etc.
- Historically, used a similar mechanism as interrupts
  - User program generates a trap
  - Kernel exception handler is called (similar to interrupt handler)
  - OS identifies which system call is required, and calls the relevant procedure
  - Returns execution to user process on completion
- INT instruction in x86 used to invoke software traps
- 8-bit opcode allows 256 different exceptions
  - Linux: `INT 0x80` is a system call
  - Windows: `INT 0x2E` is a system call
- Modern mechanism: syscall/sysenter instruction (eliminates much of the overhead of a traditional interrupt)

Processing a system call



Typical system call processing sequence

OS kernel design

- How should kernel subsystems communicate with one another?
- Should all kernel subsystems execute in kernel mode or user mode?

Monolithic kernel Communication through function calls; everything runs in the same privileged protection ring
- Pro: fast
- Con: one part of the kernel crashes, and the whole system goes down

Microkernel Small core kernel component runs in privileged mode; kernel subsystems run in user mode; communication among components via message passing
- Pro: typically only core kernel crash can take the system down (isolation)
- Con: overhead of message passing

OS kernel design

Modularization

- Dynamically loadable modules implement various OS subsystems
  - Device drivers are the most common
  - Can also be more fundamental components of the OS, *e.g.*, CPU scheduler
- Pro: Running OS only contains desired/required features; functionality can easily be modified by swapping modules that implement a particular feature
- Con: A little bit harder to implement (far outweighed by pros)
- Somewhat orthogonal issue to kernel communication and isolation design
  - `lsmod` on Linux to see what kernel modules are loaded

Policy versus Mechanism

A key design goal for an OS is to separate policy and mechanism.

- OS kernel should avoid embedding policies
- Provide neutral low-level mechanisms upon which higher layers can implement various policies
- Separation allows policies to change, perhaps on-the-fly

Policy or mechanism?

- Saving the register state of a process
- How long a user process is allowed to execute on the CPU
- The timer interrupt
- Continuing to run the current process when a disk I/O interrupt occurs