OS Structure

• Topics
  • Hardware protection & privilege levels
  • Control transfer to and from the operating system

• Learning Objectives:
  • Explain what hardware protection boundaries are.
  • Explain how applications interact with the operating system and how control flows between them.
What makes the kernel different?

Protection Boundary

Operating System
- file system
- device drivers
- processes
- networking
- virtual memory

HW/SW Interface

Applications

Hardware
Protection Boundaries

- Processor hardware typically provides (at least) two different privilege levels:
  - User mode: how all “regular programs” run.
  - Kernel mode or supervisor mode: how the OS runs.
  - Most processors have only two modes; x86 has four; some older machines had 8!

- The mode in which a piece of software is running determines:
  - What instructions may be executed.
  - How addresses are translated.
  - What memory locations may be accessed (enforced through translation).
Example: Intel

- Four protection levels
  - Ring 0: Most privileged: OS runs here
  - Rings 1 & 2: Ignored in many environments, although, can run less privileged code (e.g., third party device drivers; possibly some parts of virtual machine monitors)
  - Ring 3: Application code

- Memory is described in chunks called segments
  - Each segment also has a privilege level (0 through 3)
  - Processor maintains a “current protection level” (CPL) - usually the protection level of the segment containing the currently executing instruction.
  - Program can read/write data in segments of less privilege than CPL
  - Program cannot directly call code in segments with more privilege.
Which of the following statements are true on the Intel architecture?

- The OS can read application data
- The OS can execute application code
- Applications can read OS data
- Applications can execute code from segments with privilege level 1
Example: MIPS

• Standard two mode processor
  • User mode: access to CPU/FPU registers and flat, uniform virtual memory address space.
  • Kernel mode: can access memory mapping hardware and special registers.
Changing Protection Levels

• Must answer two fundamental questions:
  • How do we transfer control between applications and the kernel?
  • When do we transfer control between applications and the kernel?
• How: Fundamental mechanism that transfers control from less privileged to more privileged is called a trap.
• There are different kinds of traps; this gets us to the when …
When does the OS get to run?

• Sleeping Beauty Approach
  • Hope that something happens to wake you up.
  • What might happen?
    • System calls: An application might want the operating system to do something on its behalf.
    • Exceptions: An application unintentionally does something that requires OS assistance (e.g., divide by 0, read a page not in memory).
    • Interrupts: An asynchronous event (e.g., I/O completion).

• Alarm Clock Approach
  • The OS can set a timer, which will generate an interrupt, which guarantees that the OS gets to run at a specific time in the future.
The OS always decides when the processor transitions from user mode to supervisor mode.

- Allow Retry
  - True
  - False
Transferring Control

• Regardless of why and when control must transfer to the operating system, the mechanism is the same.
• First, we’ll talk about what must happen in the abstract (i.e., not in the context of any particular processor).
• Then, we’ll step through two different hardware platforms and examine how they transfer control.

Key points:
• We can invoke the operating system explicitly via a system call.
• The operating system can be invoked implicitly via an exception (sometimes called a software interrupt), such as a divide by zero or a bad memory reference.
• The operating system can be invoked asynchronously via (hardware) interrupts, such as a timer, an I/O device, etc.
Trap Handling

- Each type of trap is assigned a number. For example:
  - 1 = system call
  - 2 = timer interrupt
  - 3 = disk interrupt
  - 4 = interprocessor interrupt

- The operating system sets up a table, indexed by trap number, that contains the address of the code to be executed whenever that kind of trap happens.

- These pieces of code are called “trap handlers.”

```
2/2/16
CS161 Spring 2016
```
MIPS (sys161) Trap Handling (1)

- MIPS has only 5 distinct traps and those addresses are hardwired (no software dispatch)
  - One each for:
    1. Reset,
    2. NMI (non-maskable interrupt)
    3. Fast-TLB loading
    4. Debug
    5. One for everything else (software must then do further dispatch).

- **Note**: Sys/161 does not support NMI or debug

- Trap handling varies according to the type of trap.
MIPS (sys161) Trap Handling (2)

• The MIPS processor has special registers that get set with vital information at trap time. For example:
  • The **EPC (exception program counter)** tells you the address that caused the exception.
  • The **cause register** is set to a value indicating the source of the trap -- interrupt, exception, system call, and which kind of interrupt/exception/system it was.
  • The **status register** indicates:
    • Mode the processor was in when the interrupt happened.
    • The state of which kinds of interrupts/exceptions are enabled
• In early versions of the MIPS, you return from trap handlers using a combination of a JMP instruction and an RFE (return from exception).
• Later versions have ERET (exception return).
x86 Trap Handling

- Hardware register, traditionally called PIC (Programmable Interrupt Controller), then APIC (advanced PIC) and most recently LAPIC (local advanced PIC, one per CPU in the system)
  - Maps different events to particular locations in IDT (interrupt descriptor table).
  - PIC sends the appropriate value for the interrupt handler dispatch to the processor.
- Recall:
  - x86 has multiple protection levels
  - Cannot directly call code in a different level.
  - So, we need a special mechanism to facilitate the transfer.
- IDT: contains special objects called gates.
  - Gates provide access from lower privileged segments to higher privileged segments.
    - When a low-privilege segment invokes a gate, it automatically raises the CPL to the higher level.
    - When returning from a gate, the CPL drops to its original level.
  - First 32 gates reserved for hardware defined traps.
  - Remaining entries are available to software using the INT (interrupt) instruction.
x86 System Calls

- There are multiple ways to handle system calls and different operating systems use different ways:
  - Linux uses a single designated INT instruction (triggers a software interrupt) and then dispatches again within a single handler (like MIPS).
  - Solaris uses the LCALL instruction (goes through a gate).
  - Some new Linux systems use the newer SYSENTER/SYSEXIT calls.
- The IRET instruction returns from the trap
Recap

• The operating system is just a bunch of code that sits around waiting for something to do (e.g., help out a user process, respond to a hardware device, process a timer interrupt, etc).
• The operating system runs in **privileged mode**.
• Hardware provides some sort of mechanism to transfer control from one privilege level to another.
• We use the term **trap** to refer to any mechanism that transfers control into the operating system.
• There are different kinds of traps:
  • **Interrupts** (caused by hardware; asynchronous)
  • **Exceptions** (software interrupts; synchronous with respect to programs)
  • **System calls**: intentional requests of the operating system on behalf of a program; synchronous with respect to the program)