Overview

- Recall that OS manages h/w resources
  - Provides virtualization mechanism for sharing resources
  - Provides systems calls for abstracting h/w

- OS is software, when other applications are running, it is not running, so how does it manage resources?
  - How can OS ensure that applications share resources safely?
  - How can OS ensure that apps use the h/w abstraction?

- OS needs help from hardware
Hardware Support for OS

- OS uses three hardware features
  - CPU mode
  - Memory management unit (MMU)
  - Trap instruction

virtualization
abstraction
CPU Modes

- CPUs have two modes: kernel, user
  - In **kernel mode**, every instruction can be executed
  - In **user mode**, only a subset of instructions can be executed

- CPU maintains current mode in its status register

Typical instructions: load, store, add, sub, push, pop …

Allow accessing devices such as disk and timer (io instructions), controlling interrupts
OS Runs in Kernel Mode

- OS runs in kernel mode
  - Has full privileges, access to all memory, devices, etc.
  - Thus called OS kernel

![Diagram showing kernel and user mode with no direct access to hardware](image)
OS Operation

- When h/w boots, it runs in kernel mode
  - BIOS loads OS executable from disk into memory
  - Starts running OS code

- OS ensures that programs only execute in user mode
  - So, programs have limited privileges
    - No direct access to devices, limited memory access
  - When programs need access to device, they MUST call OS

- OS is a privileged, trusted program
  - Correct system operation depends on correct OS design and implementation, but not on correct user programs
  - OS ensures that buggy programs affect themselves, not other apps or OS
Hardware Support for OS

- OS uses three hardware features
  - CPU modes
  - Memory management unit (MMU)
  - Trap instruction
Memory Management Unit

- How can OS provide each program its own private, contiguous (virtual) memory? Recall mem.c
- CPU has a Memory Management Unit (MMU)

1. Programs use **virtual** memory addresses

3. MMU translates virtual address to **physical** memory address

2. CPU sends these virtual addresses to MMU

4. MMU accesses memory using physical addresses
A Simple MMU

- A simple MMU has one **base** register and one **limit** register

- Say program P1 is running
  - Base reg = Base 1 (physical addr)
  - Limit reg = Base 1 + Size 1

- MMU converts virt to phys address:
  - Phys addr = Virt addr + Base reg
  - When P1 reads data at virt addr 0, data at phys addr Base 1 is returned

- Limit register bounds memory accesses
  - Base reg <= Phys addr < Limit reg
  - 0 <= virt addr < Size 1
Using MMU for Memory Isolation

- Assume two programs are running concurrently
  - When P1 starts running
    - Load P1 is available memory
      - base = Base 1, limit = Limit 1
  - When P2 starts running
    - Load P2 in available memory
      - base = Base 2, limit = Limit 2
- Questions:
  - How does OS enable memory isolation?
  - Can programs change MMU registers?
Summary

- OS manages h/w
  - Virtualizes h/w
  - Abstracts h/w

- Implementing this functionality efficiently and securely requires h/w support: CPU modes, MMU and Trap
  - CPU modes: OS runs with full privileges, but programs run with limited privileges, so that they do not have access to h/w
  - MMU: helps provide memory isolation
Think Time: Program Operation

- What if a program tries to cheat?
  - Why can’t a program access a device directly?
  - For a base-limit MMU, what happens if program accesses memory outside its base-limit range?
  - What stops a program from modifying the OS so that the OS runs user code in kernel mode?
  - What stops a program from changing the MMU registers?
Think Time Answers: Program Operation

- What if a program tries to cheat?
  - Why can’t a program access a device directly?
    - IO instructions are privileged. OS ensures that device registers are mapped to OS memory, which the program can’t see.
  - For a base-limit MMU, what happens if program accesses memory outside its base-limit range?
    - MMU/CPU will generate an exception if the address is outside the base-limit range. We will talk about exceptions later.
  - What stops a program from modifying the OS so that the OS runs user code in kernel mode?
    - OS uses the MMU to ensure that programs cannot see, let alone modify OS code.
  - What stops a program from changing the MMU registers?
    - Modifying MMU registers is a privileged operation.
Think Time: CPU Mode

- Operating systems provide administrator/root accounts that have full machine access. Do programs running with root privileges run in kernel mode?
Think Time Answers: CPU Mode

- Operating systems provide administrator/root accounts that have full machine access. Do programs running with root privileges run in kernel mode?
  - No, all programs, even root owned programs, run in user mode. These programs have higher privileges, such as being able to access the files or other users because the OS implements this behavior as part of the system call code, as we will see later.
Think Time: MMU

- How does virtual to physical address translation work when the OS runs two programs in parallel on two cores?
- If programs can only access virtual memory, then how does the OS access all physical memory?
How does virtual to physical address translation work when the OS runs two programs in parallel on two cores?

- Each CPU core has its own MMU, so the two MMUs can perform the address translation independently. The OS sets up the MMU registers on each core for the program running on that core.
If programs can only access virtual memory, then how does the OS access all physical memory?

- This depends on the CPU and MMU. Consider the base and limit register MMU. With that MMU, the OS can setup the base register value to 0 and the limit register value to the maximum physical memory size whenever it runs (e.g., on an interrupt or on a system call). In that case, the virtual address the OS generates will be the SAME as the physical memory address (size base = 0), and the OS will be able to access all physical memory while using virtual addresses!

You should find out how a modern OS (e.g., Linux) accesses all physical memory.
Think Time: Fun Questions

- Would it be possible to implement OS functionality, i.e., virtualization and abstraction, without hardware support?
Would it be possible to implement OS functionality, i.e., virtualization and abstraction, without hardware support?

- One could have an interpreter/simulator that interprets every CPU instruction and provides h/w virtualization and abstraction, but this would be very slow since each CPU instruction would need to be interpreted in software. This would have similarities with the way valgrind works.