Design of Operating Systems

- OS manages h/w resources
  - Ensures that multiple programs can share h/w safely
  - Makes it simpler for a program to access h/w
Key Concepts in OS Design

- How does OS ensure that programs share h/w safely?
  - OS uses a virtualization mechanism to partition resources
  - Thus, apps are not aware that other apps are running

- How does OS make it simpler to access h/w?
  - OS provides an abstraction for h/w, via system calls
Isolation via Virtualization

- Virtualization provides the illusion that each program is running on its own machine
  - Program thinks it has full access to CPU, memory, disk
  - This is called virtualization because there is *one physical machine*, but the OS provides the illusion of *multiple virtual machines*
Benefits of Virtualization

- Programs are not aware that other programs exist, i.e., each program is isolated from others
  - Programs can be written independently of each other
  - Program cannot accidentally overwrite another program’s memory or files, causing it to crash
  - If a program uses too much resources only its performance degrades (performance isolation)

- Programs can be written portably
  - Regardless of amount of physical resources that are available
Implementing Virtualization

- OS implements virtual machines by virtualizing each physical resource
  - CPU -> Threads
    - OS provides illusion that a large number of virtual CPUs, called threads, are available to programs
  - Memory -> Virtual Memory
    - OS provides the illusion that each program has access to a large amount of contiguous, private memory
  - Disk (Hard Drive) -> Files
    - OS provides the illusion that a large number of virtual disks, called files, are available to programs
  - Network Card/Interface -> Socket

- Comparing physical and virtual machines
Concurrency

- Thread abstraction allows a program to perform several tasks concurrently or in parallel

- Speeding up a webserver using threads

- Concurrent threads require synchronization
  - Otherwise, the program may generate incorrect results

- The OS runs multiple programs and thus needs to synchronize them as well
Abstraction via System Calls

- OS provides an abstraction of h/w via system calls
  - System calls are operating systems functions that provide portable access to hardware and other OS functionality
  - Make it easier to write programs
- Set of system calls is OS API
- Examples:
  - Create/destroy thread or process
  - Allocate/deallocate memory from system
  - Read/write a file
From app perspective, system call seems like a function call.

Will discuss need for traps (like a door) later.
OS Interface

Application layer

system calls: thread_create(), read(), write(), thread_join() ...

OS interface

OS Kernel

Thread scheduler
Memory management
Device Mgmt
File System
Network Comm.
Protection
Process Mgmt
Security

physical machine interface

Hardware layer

Network
CPU
Memory

Printer

Monitor
Disk

Video Card
Summary

- Two important OS concepts
  - OS isolates applications using virtualization
  - OS simplifies writing applications by providing a hardware abstraction

- Virtualization
  - Each program thinks it owns machine, i.e., CPU, mem, disk
  - Programs don’t need to worry about other running programs
    - E.g., a program cannot corrupt other programs
  - Programs can be written portably
    - Don’t need to worry about actual physical resources available

- Abstraction
  - Simpler interface to h/w implemented via system calls
  - Eases programing, improves portability
Think Time

- What is the difference between a library call, e.g., a call to the printf function, and a system call, e.g., the write system call?
- What are the properties of an “ideal” virtual machine?
What is the difference between a library call, e.g., a call to the printf function, and a system call, e.g., the write system call?

- A library call, such as the printf function, is part of the code of an application. A system call, such as the write system call, is part of the code of an operating system. From an application’s perspective, both calls seem like function calls and returns, but they are invoked differently. A library call is invoked like a function call, using a call instruction. A system call is invoke using a trap instruction. Incidentally, the printf library call uses the write system call to print data to a terminal. So, when a program calls the printf function call, it indirectly calls the write system call. Remzi’s textbook has an excellent description of the differences between a library and an OS.
What are the properties of an “ideal” virtual machine?

The ideal virtual machine will behave exactly like a physical machine, e.g., a program running on one physical machine can’t directly affect a program running on another physical machine. With an ideal virtual machine, a program should not even be able to detect that another program is running. In practice, since resources are shared, it is hard to ensure this ideal behavior. Specifically, programs can often detect that they are not running at full speed, and they do not have access to all physical memory because other programs are running.