Aims of virtualization

- Multiplex resources
  - Give the illusion that you own the resources.
- Isolation/abstraction
  - Software does not need to know the details of the hardware on which it runs.
  - (avoid interference, safety, etc.)
- Process vs. System Virtualization
  - Process virtualization
    - JVM ("write once, run everywhere" model)
    - Dynamic Binary Translators (ISA: Rosetta, Mambo; OS&library calls: Wine)
    - Dynamic Binary Optimizers – program shepherding (Pin, Valgrind)
Aims of virtualization

- Multiplex resources
Aims of virtualization

- Multiplex resources

Give the illusion that you own the resources.
Aims of virtualization

- Multiplex resources

Give the illusion that you own the resources.

- Isolation/abstraction
Aims of virtualization

- Multiplex resources
  
Give the illusion that you own the resources.

- Isolation/abstraction
  
Software does not need to know the details of the hardware on which it runs.
Previous Lecture: Virtualization Technologies

- Aims of virtualization
  - Multiplex resources
  
  Give the illusion that you own the resources.
  
  - Isolation/abstraction
  
  Software does not need to know the details of the hardware on which it runs.
  
  - (avoid interference, safety, etc.)
Aims of virtualization

Multiplex resources

Give the illusion that you own the resources.

Isolation/abstraction

Software does not need to know the details of the hardware on which it runs.

(avoid interference, safety, etc.)

Process vs. System Virtualization
Previous Lecture: Virtualization Technologies

- Aims of virtualization
  - Multiplex resources
  
  *Give the illusion that you own the resources.*
  
  - Isolation/abstraction
  
  *Software does not need to know the details of the hardware on which it runs.*
  
  - (avoid interference, safety, etc.)

- Process vs. System Virtualization

- Process virtualization
Previous Lecture: Virtualization Technologies

- Aims of virtualization
  - Multiplex resources
  
  Give the illusion that you own the resources.
  
  - Isolation/abstraction

  Software does not need to know the details of the hardware on which it runs.
  
  - (avoid interference, safety, etc.)

- Process vs. System Virtualization

- Process virtualization
  
  - JVM (“write once, run everywhere” model)
Aims of virtualization

- Multiplex resources

Give the illusion that you own the resources.

- Isolation/abstraction

Software does not need to know the details of the hardware on which it runs.

- (avoid interference, safety, etc.)

Process vs. System Virtualization

Process virtualization

- JVM ("write once, run everywhere" model)
- Dynamic Binary Translators (ISA: Rosetta, Mambo; OS&library calls: Wine)
Previous Lecture: Virtualization Technologies

- Aims of virtualization
  - Multiplex resources
  - Isolation/abstraction

*Give the illusion that you own the resources.*
  - (avoid interference, safety, etc.)

- Process vs. System Virtualization

- Process virtualization
  - JVM ("write once, run everywhere" model)
  - Dynamic Binary Translators (ISA: Rosetta, Mambo; OS&library calls: Wine)
  - Dynamic Binary Optimizers – program shepherding (Pin, Valgrind)
Today’s Lecture – Learning Objectives

- To understand the implementation choices and details of System Virtualization
Today’s Lecture – Learning Objectives

▶ To understand the implementation choices and details of System Virtualization

▶ how virtualization works in modern architectures
Today’s Lecture – Learning Objectives

▶ To understand the implementation choices and details of System Virtualization

▶ how virtualization works in modern architectures
▶ what are the choices and characteristics of such implementations
Aims and Definitions

Unvirtualized

- Application
- Operating System
- Hardware

Virtualized

- Applications
- Guest A Operating System
- Virtual Machine Monitor (VMM) / Hypervisor
- Guest B Operating System
- Host Hardware
Hosted Virtualization

- Application
- Applications
- Host Operating System
- Guest A
  - Operating System
- Virtual Machine Monitor (VMM) / Hypervisor
- Guest B
  - Operating System
- Host Hardware
XEN Guest 0 Virtualization

Applications

Guest 0
Operating System

Virtual Machine Monitor (VMM) / Hypervisor

Guest A
Operating System

Applications

Guest B
Operating System

Applications

Host Hardware
Revision: OS Protection/Privilege

- OS handles physical resources

Diagram:

- Application
- Operating System
- Hardware

Unvirtualized
Revision: OS Protection/Privilege

- OS handles physical resources
  - Privileged

Diagram:
- Application
- Operating System
- Hardware
- Unvirtualized
OS handles physical resources
  ▶ Privileged

Application isolated from resources
Revision: OS Protection/Privilege

- OS handles physical resources
  - Privileged

- Application isolated from resources
  - Non-privileged
Virtualization Protection/Privilege

- VMM handles physical resources

![Diagram showing the relationship between applications, guest operating systems, virtual machine monitor (VMM), and host hardware.]

Virtualized
Virtualization Protection/Privilege

- VMM handles physical resources
- Privileged

(Guest A: Operating System, Virtual Machine Monitor (VMM) / Hypervisor, Host Hardware)

(Guest B: Operating System, Virtual Machine Monitor (VMM) / Hypervisor, Host Hardware)

Applications

Virtualized
Virtualization Protection/Privilege

- VMM handles physical resources
  - Privileged

- Guest OS isolated from resources
Virtualization Protection/Privilege

- VMM handles physical resources
  - Privileged

- Guest OS isolated from resources
  - non- (or less-) privileged

Applications
Guest A Operating System
Virtual Machine Monitor (VMM) / Hypervisor
Guest B Operating System
Host Hardware

Virtualized
Virtualization Protection/Privilege

- VMM handles physical resources
  - Privileged

- Guest OS isolated from resources
  - non- (or less-) privileged

VMM gets control on every guest OS access to physical resource
Guarded Physical Resources

- Timers
- CPU registers

- Device Control Registers
Guarded Physical Resources

- Timers
- CPU registers
  - Interrupt Enable
- Device Control Registers
Guarded Physical Resources

- Timers
- CPU registers
  - Interrupt Enable
  - Page Table Base
- Device Control Registers
Guarded Physical Resources

- Timers
- CPU registers
  - Interrupt Enable
  - Page Table Base
- Device Control Registers
  - Programmed I/O?
  - Interrupt I/O?
  - DMA I/O?
Guarded Physical Resources

- Timers
- CPU registers
  - Interrupt Enable
  - Page Table Base
- Device Control Registers
  - Programmed I/O?
  - Interrupt I/O?
  - DMA I/O?
- Interrupts (may be for different Guest?)
Guarded Physical Resources

- Timers
- CPU registers
  - Interrupt Enable
  - Page Table Base
- Device Control Registers
  - Programmed I/O?
  - Interrupt I/O?
  - DMA I/O?
- Interrupts (may be for different Guest?)
- Memory Mapping (page tables)
VMM Entry from Guest

- VMM designers are (a bit) lucky
VMM Entry from Guest

- VMM designers are (a bit) lucky
  - Many Guest accesses to physical resources cause trap in non-privileged mode
VMM Entry from Guest

- VMM designers are (a bit) lucky
  - Many Guest accesses to physical resources cause trap in non-privileged mode
  - So, running the OS in non-privileged mode suffices
VMM Entry from Guest

- VMM designers are (a bit) lucky
  - Many Guest accesses to physical resources cause trap in non-privileged mode
  - So, running the OS in non-privileged mode suffices

- BUT some instructions behave differently (without trapping) in privileged and non-privileged mode (e.g. Intel “Store into Flags”)
Accessing Memory under Virtualization

Unvirtualized

Virtual Address \(\rightarrow\) OS Page Tables \((+\text{TLB for efficiency})\) \(\rightarrow\) Physical Address

Virtualized

Virtual Address \(\rightarrow\) OS Page Tables \(\rightarrow\) VMM Page Tables \(\rightarrow\) Physical Address
Accessing Memory under Virtualization

Unvirtualized

Virtual Address → OS Page Tables (+TLB for efficiency) → Physical Address

Virtualized

Virtual Address → OS Page Tables → VMM Page Tables → Physical Address

What about TLBs?
Interfacing Guest OS and VMM

Three solutions today:
Interfacing Guest OS and VMM

Three solutions today:

► Software (static)
Interfacing Guest OS and VMM

Three solutions today:

- Software (static)
- Software (dynamic)
Interfacing Guest OS and VMM

Three solutions today:

- Software (static)
- Software (dynamic)
- Hardware (dynamic)
ParaVirtualization

Modify Guest OS to be Virtualization-aware:

- call VMM for all privileged operations
- cooperate with VMM over shared page tables
- call VMM for input-output
ParaVirtualization

Modify Guest OS to be Virtualization-aware:

- call VMM for all privileged operations
ParaVirtualization

Modify Guest OS to be Virtualization-aware:

- call VMM for all privileged operations
- cooperate with VMM over shared page tables
ParaVirtualization

Modify Guest OS to be Virtualization-aware:

- call VMM for all privileged operations
- cooperate with VMM over shared page tables
- call VMM for input-output
ParaVirtualization

Modify Guest OS to be Virtualization-aware:

- call VMM for all privileged operations
- cooperate with VMM over shared page tables
- call VMM for input-output

Advantages? Disadvantages?
Detect and Fix Interfaces in VMM

- Detecting
  - Write-protect Guest OS page tables
  - Code-scan (Dynamic Binary Translation?) Guest OS for unsafe instructions – plant traps

- Fixing
  - Use write-error trap to detect guest page-table writes
  - Provide “shadow page tables” for hardware TLBs
  - Use “illegal instruction” and “trap” traps
Detect and Fix Interfaces in VMM

- Detecting
  - Write-protect Guest OS page tables
Detect and Fix Interfaces in VMM

► Detecting
  ► Write-protect Guest OS page tables
  ► Code-scan (Dynamic Binary Translation?) Guest OS for unsafe instructions – plant traps

► Fixing
Detect and Fix Interfaces in VMM

- **Detecting**
  - Write-protect Guest OS page tables
  - Code-scan (Dynamic Binary Translation?) Guest OS for unsafe instructions – plant traps

- **Fixing**
  - Use write-error trap to detect guest page-table writes
Detect and Fix Interfaces in VMM

- **Detecting**
  - Write-protect Guest OS page tables
  - Code-scan (Dynamic Binary Translation?) Guest OS for unsafe instructions – plant traps

- **Fixing**
  - Use write-error trap to detect guest page-table writes
  - Provide “shadow page tables” for hardware TLBs
Detect and Fix Interfaces in VMM

detecting
- Write-protect Guest OS page tables
- Code-scan (Dynamic Binary Translation?) Guest OS for unsafe instructions – plant traps

fixing
- Use write-error trap to detect guest page-table writes
- Provide “shadow page tables” for hardware TLBs
- Use “illegal instruction” and “trap” traps
Detect and Fix Interfaces in Hardware

- Requirement

VMM runs more-privileged than Guest OS

Hardware provides Application/OS and VMM modes

When Virtualization is active, all OS accesses to physical resources trap to VMM

Advantages? Disadvantages?
Detect and Fix Interfaces in Hardware

- Requirement
  - VMM runs more-privileged than Guest OS
Detect and Fix Interfaces in Hardware

- Requirement
  - VMM runs more-privileged than Guest OS
- Hardware provides Application/OS and VMM modes
Detect and Fix Interfaces in Hardware

- **Requirement**
  - VMM runs more-privileged than Guest OS

- Hardware provides Application/OS and VMM modes

- When Virtualization is active, all OS accesses to physical resources trap to VMM
Detect and Fix Interfaces in Hardware

- Requirement
  - VMM runs more-privileged than Guest OS

- Hardware provides Application/OS and VMM modes

- When Virtualization is active, all OS accesses to physical resources trap to VMM

Advantages? Disadvantages?