Protection via Virtual Memory

1

User/Supervisor Mode

Architecture must provide at least two modes to indicate whether the running process is a user process or an operating system process (kernel/supervisor process).

2

Protected Processor State

A portion of processor state that users can use but not write, including user/supervisor mode bit, exception enable/disable bit, and memory protection information.

3

Mode Transition

Mechanisms for the processor to switch between user mode and supervisor mode, typically through system calls and returns.

4

Memory Protection

Mechanisms to limit memory accesses to protect the memory state of a process without having to swap the process to disk on a context switch.

These architectural features allow processes to share hardware while preventing interference between them.

Page-Based Memory Protection

- The most popular memory protection scheme adds protection restrictions to each page of virtual memory.
  - Fixed-sized pages (typically 4KB or 8KB) are mapped from virtual to physical address space via a page table, with protection restrictions included in each entry.
- These restrictions determine whether a user process can read or write to a page, and whether code can be executed from it.
  - Since only the OS can update the page table, this mechanism provides total access protection.
- To avoid the performance penalty of address translation, systems use a Translation Lookaside Buffer (TLB) that caches recent address translations, taking advantage of locality principles.

The Need for Stronger Protection

> "A virtual machine is taken to be an efficient, isolated duplicate of the real machine... a VMM has three essential characteristics. First, the VMM provides an environment for programs which is essentially identical with the original machine; second, programs run in this environment show at worst only minor decreases in speed; and last, the VMM is in complete control of system resources."

— Gerald Popek and Robert Goldberg, "Formal requirements for virtualizable third generation architectures," 1974

Despite hardware faithfully enforcing memory protection, we depend on the accuracy of operating systems consisting of tens of millions of lines of code. With bugs measured per thousand lines of code, production operating systems contain thousands of vulnerabilities that are routinely exploited.

This problem has led to exploration of protection models with much smaller code bases than the full OS, such as Virtual Machines.

Virtual Machines

- Virtual Machines (VMs) were first developed in the late 1960s and have remained important in mainframe computing. They've recently gained popularity due to:
  - Increasing importance of isolation and security in modern systems
  - Failures in security and reliability of standard operating systems
  - Need to share single computers among many unrelated users (datacenters/cloud)
  - Dramatic increases in processor speeds making VM overhead more acceptable
- System Virtual Machines present the illusion that users have an entire computer to themselves, including a copy of the operating system, while a single computer runs multiple VMs supporting different operating systems.

Virtual Machine Monitors (VMMs)

Definition

The software that supports VMs is called a virtual machine monitor (VMM) or hypervisor. It determines how to map virtual resources to physical resources through time-sharing, partitioning, or software emulation.

Size & Security

VMMs are much smaller than traditional operating systems; the isolation portion is perhaps only 10,000 lines of code, making them potentially more secure than full operating systems.

Performance

Virtualization overhead depends on the workload. User-level processor-bound programs have near-zero overhead, while I/O-intensive workloads with many system calls can have higher overhead.

The goal of architecture and VMM design is to run almost all instructions directly on native hardware when guest VMs run the same ISA as the host.

Commercial Benefits of Virtual Machines

Managing Software

VMs provide an abstraction that can run complete software stacks, including legacy operating systems. A typical deployment might include:

- Some VMs running legacy OSes
- Many running current stable OS releases
- A few testing next OS releases

Managing Hardware

VMs allow separate software stacks to run independently yet share hardware, consolidating servers. Benefits include:

- Running applications with compatible OS versions
- Improving dependability through isolation
- Supporting migration of running VMs between computers
- Balancing load or evacuating from failing hardware

Requirements and Challenges of VMMs

Isolation Requirements

Guest software should behave exactly as if running on native hardware, and should not be able to change allocation of real system resources directly.

Processor Virtualization

The VMM must control privileged state, address translation, I/O, exceptions and interrupts. It must run at a higher privilege level than guest VMs.

Memory Virtualization

VMMs separate virtual, real, and physical memory, using shadow page tables to map directly from guest virtual address space to physical address space.

I/O Virtualization

The most difficult part due to device diversity and sharing. VMMs provide generic device drivers and handle mapping virtual to physical I/O devices.

ISA Support for Virtualization

- Virtualizable Architectures
  - If VMs are planned during ISA design, it's easier to reduce instructions that must be executed by a VMM.
  - Most instruction sets (80x86 and RISC architectures) were created without virtualization in mind, requiring special handling of problematic instructions.
- Requirements for Virtualization
  - At least two processor modes: system and user
  - A privileged subset of instructions available only in system mode
  - All system resources must be controllable only via privileged instructions
  - Trapping of privileged instructions when executed in user mode

Paravirtualization: The Xen Approach

Paravirtualization Concept

Allows small modifications to the guest OS to simplify virtualization, reducing inefficiencies where guest OS work is ignored by the VMM.

Xen Implementation

Xen maps itself into the upper 64 MB of each VM's address space. It uses 80x86's four protection levels: VMM at level 0, guest OS at level 1, and applications at level 3.

OS Modifications

Linux port to Xen changes about 3000 lines, or 1% of the 80x86-specific code, without affecting application-binary interfaces.

I/O Handling

Xen assigns privileged VMs (driver domains) to each hardware I/O device. Regular VMs (guest domains) use simple virtual device drivers that communicate with physical drivers.

This approach simplifies virtualization while maintaining strong isolation between virtual machines.