VIRTUALIZATION

Andrew Herbert

Cambridge University
ANSA
Microsoft Research
EDSAC Replica Project
**DEFINITION (In context of SOSP)**

- *Virtualization* is a property of operating systems that gives the illusion of efficiently running multiple independent computers known as *virtual machines*.
- The virtual machines be directly executed by (i.e., exactly mimic) the underlying physical machine, or they may comprise a more abstract system, parts, or all, of which are simulated by the physical machine.
- The part of the operating system that provides the virtual machine abstraction is commonly called a *virtual machine monitor* or *hypervisor*. 
SOSP & VIRTUALIZATION

- Virtual Machine Monitors
- Layered Abstract Machines
- OS Process as a Virtual Environment
- Addressing Virtual Resources

Note: referenced systems, papers are exemplars, not an exhaustive list.
VIRTUAL MACHINE MONITORS

• Origins in 1965 IBM M44/44X to explore page replacement algorithms

• M44/44X allowed each “virtual machine” to have its own paging strategy to allow measurement and comparison

• Evolved via CP/40 and CP/67 to VM/370
IBM VM/370

- Addressed three needs
  - Time-sharing computer utility
  - Running legacy applications and their operating systems alongside new applications and systems
  - Developing and debugging new operating systems using same time-sharing environment as for applications
VIRTUALIZATION APPROACHES

• Semi-formal model by Popek and Goldberg (1974)
  • Notion of sensitive instructions which reveal processor state

• IBM 360/370 enabled pure virtualization
  • Only privileged instructions in virtual supervisor mode have to be simulated by the VMM

• In contrast to hybrid virtualization
  • Some unprivileged instructions have to be simulated in virtual supervisor mode in addition
  • x86 only became pure with Intel VT-x / AMD SVM (2006)
1975-1995
1975-1995

• IBM VM/370 continues...

• Recursive virtual machines...
  • Lauer & Weyth, 1973
  • CAP (Needham & Walker, 1977)

• Then...
“SPECIAL EFFECT” VMMs

- Hypervisor-based fault tolerance
  - (Schneider & Bressoud 1996)

- ReVirt: enabling intrusion analysis...
  - (Dunlap, King, Cinar, Basrai & Chen 2002)
HOSTED VMMs (TYPE 2)

- Enable desktop OS coexistence
- VMM runs as app on host OS
  - Often with associated kernel driver
  - Uses host OS services (file, network)
  - VMs run guest OS image
- DEC-10 VMM for ITS (Galley, 1973)
- VMWare Desktop for Windows (1999)
NATIVE/BARE METAL VMMs (TYPE 1)

- DISCO (Buignon, Devine, et al. 1997)
  - Hybrid virtualization with binary rewriting and shadowing in place of simulation
- VMWare ESX Server (Waldspurger 2002)
  - System-wide resource multiplexing
  - Ballooning, Content-based Page Sharing
- Xen (Barham et al. 2003)
  - Paravirtualization
BENEFITS OF VIRTUAL MACHINES

• Desktop virtualization
  • OS coexistence
  • Desktop checkpoint/restore, migration

• Server virtualization
  • Server consolidation
  • Multi-tenancy with strong isolation
  • Statistical multiplexing of resources across multiple virtual servers
  • Management framework for server workloads
LAYERED ABSTRACT MACHINES

- Design and implement an operating system as a hierarchical series of abstract machines
- 'THE' multiprogramming system (Dijkstra, 1967)
  - Layer 0: concurrency and interrupt handling (processes and semaphores)
  - Layer 1: automatic memory paging
  - Layer 2: operator’s console
  - Layer 3: input-output
  - Layer 5: Algol batch system
WHY LAYER?

• Build and test layer by layer
• Structured (informal) proof of correctness

• 1972, Liskov, Venus

• 1980s, Comer, XINU
LAYERING FOR SECURITY

• US DoD Trusted Systems – Orange Book
  • A1 – Verified design
  • B2 – Structured protection
  • B3 – Security domains (reference monitor)

• *The foundations of a provably secure operating system (PSOS)* (Feiertag, Neumann, SRI, 1979)
  • Hierarchical Design Methodology, SPECIAL
  • 17 Nominal layers for verification, collapsed to 9 in the implementation
MULTICS KERNEL DESIGN

• 1977, MIT redesign and reimplemention of Multics to meet B2/B3 criteria (Schroeder, Clark & Saltzer, 1977)

• Added additional layers to system in order to remove code from the Multics supervisor so it could be reduced to be a reference monitor suitable for inspection

• *Type Extension*: treat each module as a *type manager*
  • Ensure type dependency graph does not contain cycles
  • Often had to split original modules into upper and lower types
LAYERS AND MODULES

• 1976, Haberman et al., FAMOS
  • Modularization and hierarchy in a family of operating systems
  • Set of components for building a family of (related) operating systems
  • Explored conflict between layers and modules
  • Similar model to Multics type extension
LAYERING AND NETWORKS

• Abstraction layering is not unique to systems
• Protocols defined as interacting peer entities at increasing levels of abstraction
• But Open Systems Interconnection 7-layer model regarded as overblown
• Virtualization by layering
  • Virtual Private Networks
  • Virtual LANs
OBSERVATIONS

• Most of these systems were done by people with a background in programming languages and formal specification and verification.

• Some claim layering leads to inefficiency but XINU gives good evidence otherwise.
VIRTUAL ENVIRONMENTS

• A process is a program executing in a virtual environment (Saltzer, 1966)

• 1950s & 60s – Tyranny of the instruction set
  • Each new machine required everything to be re-written
  • Emulate older machines e.g., IBM 360 – IBM 1400

• Atlas *Extracodes*
  • Undefined instructions that execute subroutines in fast memory – e.g., supervisor calls, library functions
LIBRARIES

- 1970s Emergence of practical general purpose / systems programming languages
- 1980s Unix marked transition to the standard library becoming the virtual environment
  - High level abstract interface for applications
  - Low level concrete interface for the library
- UNIX (via Multics) gave us the byte stream as the universal *virtual device* abstraction replacing messy record-oriented structures of earlier systems
DISTRIBUTED UNIX

• Distributed *single image* implementations of UNIX virtual environment didn’t last, e.g. Locus (Popek, 1981)
  • Caching/Scaling problems (see Satya on file systems)
  • Autonomy of network of workstations model outweighed benefits of centralized management
NAMING VIRTUALIZED RESOURCES

• Useful to have a uniform context [location] independent way of naming sharable [virtual] resources
• Capabilities (see Lampson for security aspects)
  • C-lists (Dennis, Van Horn 1966)
  • Capability-based addressing (Fabry 74)
  • Amoeba (Tanenbaum et al. 1986)
• Grew out of Codewords / descriptors
  • Rice R1, R2 (1959-71), B5000 (1963)
  • Tagged Architectures (Feustal 1972)
PLAN 9

- Bell Labs 1991
  - Every resource is a file
  - Name spaces are mountable, stackable, ...
ACCESSING VIRTUAL RESOURCES

• Lots of arguments about how to access virtual resources
• See Kaashoek, Liskov on threads vs events
• Custom protocol vs optimized RPC vs TCP/IP
• Active versus passive objects (OMG CORBA)
• Workstations versus Processor Banks
  • PARC, MIT Athena, CMU Andrew, Cambridge Distributed System
  • GUI windows as virtual resources, e.g. X window system
TALKING POINTS

• More mileage from using virtualization to modify operating system semantics?
• VMM ≠ TCB, especially for type 2
• Revisit layered abstract machines as a systems design and implementation principle?
• Prove systems correct by default?
  • seL4 (Klein, Elpinstone, Heiser, Andronick, ..., 2009)
  • Verve (Yang & Hawblitzel, 2010)