Instructor: Larry Rudolph

- **rudolph@cs.nyu.edu**

- office hours: before/after class or on-demand

- Bio:
  - PhD from Courant. Ultracomputer 1981
  - Postdoc @ U of Toronto
  - Faculty at CMU (3), Hebrew University (10), MIT (12+)
  - VMware: founded Mobile Virtual Phone (currently on large number of Android phones; VMware enabled)
  - Founded ReDigi (Used digital music store)
  - Currently @ Two Sigma Investments
Course Overview

• Fundamentals of Virtual Machines
  • Binary Translation (Software virtualization)
  • Hardware Supported Virtualization
  • Files, Network, Cloud virtualization uses
  • Other platforms, e.g. ARM for other devices
  • Java Virtual Machines

• Current and Future uses of VM’s
  • Take control of our data: track the trackers, fool the trackers
  • UPM: Universal Personal Computer (across all devices)
  • Virtual Automobile
  • Experimental, Think outside the box
Course Logistics

• First Half: Technologies
  • 2 or 3 problem sets, readings (source articles)
  • Midterm

• Second Half: (Exotic) Uses of Virtual Machines
  • Random Topics that Exploit VM’s power
  • Define, Work, and report on Group Projects

• Use of open source tools
  • QEMU, Risk-V, KVM, Docker, ….
Motivation

• Suppose you are a computer manufacturer

• Want to build and sell a new type of computer

• Define what the machine does, design it, build it

  • Also need to develop software: boot, firmware, some devices, compiler, etc

• Want software to be ready as soon as h/w is :)

• Need to simulate the computer in order to develop the software.
Simulation

- How to simulate a computer?
  - What is a computer?
    - It has state: Registers, Program Counter,…
    - Executes instructions
      - Loop: {Fetch, Decode, Execute, Update}
    - The ISA: Instruction Set Architecture defines the computer
Simulator Needs

- Data Structures for
  - Machine State (hidden registers)
  - Registers
  - Memory (for code and data)
  - State of Input / Output, Devices
- Simulation of memory, devices
- Process instructions
  - Fetch, Decode and Execute (update state)
Example: Add inst

- Instr <- Memory[PC]
- AND with mask & shift, branch to add routine
- AND with mask & shift to get R1
- AND with mask & shift to get R2
- AND with mask & shift to get mode bits (eg carry)
- r1Val = REG[r1]
- r2Val = REG[r2]
- REG[r1] = r1Val + r2Val
- PC = PC + 4

A really good simulator slowdown of 100 times, usually worse
Simulation Vs Emulation

• What is the difference between “sympathy” and “empathy”?

• Virtualization is more like emulation

  • Nearly no slowdown!
What is Virtualization?

• Operating System done right — strong isolation

• Addition of layer VMM between hardware and software that exports a hardware abstraction to software

• VMM can export exact same HW interface. E.g. 1964 IBM 360 introduced virtual machines — several 360’s running together.

• VMM can export different interfaces
Virtual Machine Origins

VM/370—a study of multiplicity and usefulness
by L. H. Seawright and R. A. MacKinnon

The productivity of data processing professionals and other professionals can be enhanced through the use of interactive and time-sharing systems. Similarly, system programmers can benefit from the use of system testing tools. A systems solution to both areas can be the virtual machine concept, which provides multiple software replicas of real computing systems on one real processor. Each virtual machine has a full complement of input/output devices and provides functions similar to those of a real machine. One system that implements virtual machines is IBM's Virtual Machine Facility/370 (VM/370).¹


Concurrent execution of multiple production operating systems
Testing and development of experimental systems
Adoption of new systems with continued use of legacy systems
Ability to accommodate applications requiring special-purpose OS
Introduced notions of “handshake” and “virtual-equals-real mode” to allow sharing of resource control information with CP
Leveraged ability to co-design hardware, VMM, and guestOS
Formal Definition

- Virtualization constructs isomorphism from guest to host, by implementing functions V() and E()

- All guest state S is mapped onto host state S’ through some function V(S)

- For every state change operation e(S) in the guest is a corresponding state change e’(S’) in the host
Virtualization Properties

• Isolation
• Encapsulation
• Portability
• Interposition
Isolation

• Fault Isolation
  • A fault in an OS cannot affect the other OS’s

• Software Isolation
  • Complex software configurations (e.g. DLL-hell, viruses, security hole) cannot affect other OS’s

• Performance Isolation
  • Depends on: Scheduling, Resource Allocation, Devices
Encapsulation

• The VM can be encapsulated into a file, even one that has been running
  • Dynamic memory, CPU&Device registers

• Size of the file is
  • proportional to virtual HW model
  • independent of guest software configuration

• Allows a VM to be check pointed, migrated
  • Avoid bit-rot. Can run at a later date even after software upgrades. Still cannot checkpoint external communications.
Interposition

• All guest actions mediated by the VMM

• VMM can inspect, modify, divert, hide operations, e.g.
  • Compression, Deduplication
  • Encryption, eg all disk accesses but OS does not know
  • Profiling, e.g. debug the OS
  • Translation, e.g. big-to-little endian, I/O actions
  • Replication, e.g. overcommitment of devices
  • etc.
Why not rewrite OS?

- OS’s are all about interfaces
  - Started closely tied to H/W but then broke loss
  - OS like governments, and became very bloated

- H/W interfaces much smaller, more stable, better defined so can be more reliable

- Unfortunately, VMM has less knowledge of guest apps and what they are doing.

- Appears to be a better split of abstraction layers
  - Microkernels+Library OS never caught on
Virtualization Applications

- Server Consolidation
  - Most data centers used only 10% of their capacity.
  - Convert underutilized servers into VMs
- Simplified Management
  - Datacenter provisioning and monitoring
  - Dynamic load balancing
- Improved Availability
  - Automatic Restart
  - Fault tolerance
  - Disaster Recovery
- Test and Development
- Mobile Virtualization
  - Home/work Dual Persona
Types of Virtualization

- Process Virtualization (Single process under control of VMM / User mode)
  - Language constructs
    - Java (Scala, Clojure), .Net, Python
  - Cross-ISA emulation
- Application Virtualization
  - Sandboxing, mobility, distribution
- Device Virtualization
  - RAID, Dropbox
- Network Virtualization
  - NAT, Overlay networks, fiber channel over Ethernet
- System Virtualization (whole, entire system, run unmodified OS's on top)
  - VMware, Xen, Citrix, KVM, …
System Virtual Machine Monitor (VMM)

- Traditional
- Hosted
- Hybrid
- Hypervisor
Traditional

- IBM VM/370
- Stanford DISCO
- Monitor needs to be self-sufficient
  - CPU scheduler
  - Memory Allocator
  - Device Drivers
  - File System
  - Network Stack
Hosted

- Virtual Machine is executed as an application on an existing operating system
- Reuse existing device drivers (this is a big deal)
- Leverage OS support
  - File System, CPU Scheduler, Protection
Hosted Monitor Architecture

• VMM runs in its own address space at kernel level

• VMM time-shares H/W with the host

• World Switch:
  • When host schedules the guest, or guest gives up control
  • saves registers, page-tables, etc. of host (or the opposite)
  • loads registers, page-tables, etc of guest (or the opposite)
Hosted Monitor Architecture

- Guest CPU and Memory Virtualization handled internally by VMM

- VMM has access to all privileged state of CPU to provide fastest CPU/Memory guest exception
Hosted Monitor Architecture

- VMM forwards all device requests to the UserApp in the host.
- UserApp then calls host OS to access the device
- E.G. Guest App reads USB, traps to guest OS, caught by VMM, world-switch back to UserApp with request, UserApp does USB read, traps to Host OS, …
Hosted Monitor Architecture

- Device interrupts that happen when VMM running
  - Passed back to Host OS.
- Note, CPU exceptions in guest (e.g. page faults, illegal instructions, privileged instructions) are handled by VMM
Hosted Monitor Scheduling

The virtual machine monitor runs in its own address space at kernel level. The VMM shares the hardware with the host. When the UserApp runs in the host, it switches to the VMM by way of a world switch. The world switch saves all the registers, page tables, etc. of the host and then loads the state of the VMM. Initially the state of the VMM is setup by the UserApp. This includes what the registers should be and the structure and contents of the page tables. When the VMM voluntarily gives up the CPU, it world switches back to the host.

Because the VMM does not handle devices it simply forwards all interrupts on to the host.
Hosted Monitor Scheduling

1. The CPU scheduler runs the blue UserApp
2. The UserApp switches to its VMM
3. The blue guest is executed getting its share of CPU
Hosted Monitor Scheduling

- (4) Timer interrupt comes in
- (5) The VMM forwards the timer interrupt to the host. The host scheduler then runs.
Hosted Monitor Scheduling

- (6) The host scheduler reschedules the blue UserApp and schedules the green UserApp

- (7) The green UserApp does world switch to its VMM

- (8) The green guest gets its portion of CPU time
Hosted Architecture Tradeoffs

• Pro’s
  • Installs like an application
    • No disk partitioning needed
    • Virtual disk is a file on the host file system
    • No host reboot needed (just process startup)
  • Runs like an application
    • Uses host schedulers, can be many running, startup and terminate as usual

• Con’s
  • I/O path is slow — world switch takes a lot of time doing things like flush cache, page tables, etc.
  • Relies on Host for scheduling, resource allocation, security
  • Many VMM’s making use of many devices causes many more interrupts
VMware ESX Server

- Traditional Arch for Guest OS’s
- VMKernel called without world-switch
- Service Console for VM’s management

Figure 1: ESX Server architecture

Hybrid — Xen

- Para-virtualization
  - Linux Guest
- Hardware-supported virtualization
  - Unmodified Windows
- Isolated Device Drivers

Source: Ottawa Linux Symposium 2006 presentation.
http://www.cl.cam.ac.uk/research/pubs/papers/

- Hypervisor: Small Size, Runs in Special Hardware mode
- Guest OS runs in normal privilege level
- Good for Security, Fault Tolerance, System Management
Next Lecture

Binary Translation
i.e. Virtualizing the CPU

Never Give Up Control
Popek & Goldberg


- Definition of Virtualizable if all privileged instructions trap when executed in user mode

- Surprisingly, this is not true for x86!
  - pushf/popf