Advanced Topics: Virtual Machine Monitors

CS 537: Introduction to Operating Systems

Louis Oliphant

University of Wisconsin - Madison

Fall 2024

Administrivia

- Project 6 – Due Nov 27 and Dec 6
- Final Exam: – Dec 19, 10:05-12:05

Review SSDs

- Physical Organization (SLC,MLC,TLC and Blocks and Pages)
- IO and translation to Read (a page), Erase (a block), Program (a page) Operations
- Flash Translation Layer (FTL) including mapping tables
- Log System, including data organization and garbage collection

Persistence Summary

- Managing I/O devices significant part of OS
- Disk Drives, SSDs (pages, blocks)
- File Systems: OS provided API to access disk
- Simple FS: FS layout with supberblock, bitmaps, inodes, datablocks
- Fast File System: Key idea – put inode & data close together, namespace locality
- FSCK, Journaling – Handling/Preventing data inconsistencies
- Log Structured File System - Organize data based on writes

Quiz 20 SSDs

https://tinyurl.com/cs537-fa24-q20-1

Agenda

- Motivation
- Types of Virtualization
- Requirements for Virtualization
- Virtualizing on Older Hardware
- Virtualizing on Modern Hardware
    - Virtualizing the CPU
    - Virtualizing Memory

Why Virtualize?

- Desktop
    - Operating System Diversity
    - Development / Testing
- Servers
    - Server Consolidation
    - Rapid Provisioning
    - Security
    - High-availability
    - Distributed Resource Scheduling
    - Cloud Computing

Structure of Virtualization Landscape

Virtual Machine

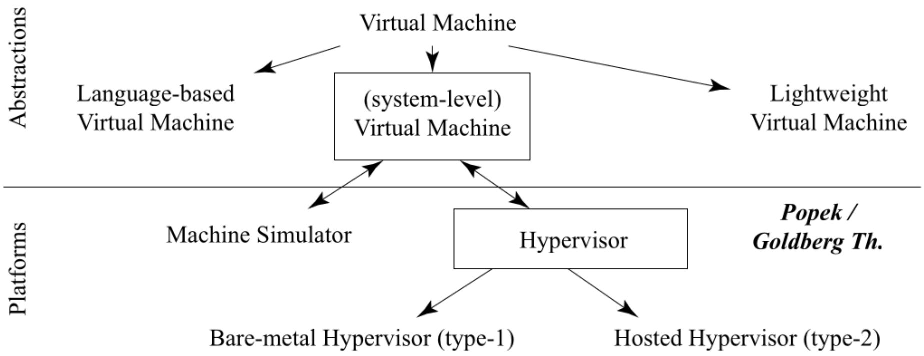
A **virtual machine** is a complete compute environment with its own isolated processing capabilities, memory, and communication channels.

- **Language-based virtual machines**: Java Virtual Machine, Javascript engine, etc.
- **Lightweight virtual machines**: Rely on combination of hardware and software mechanisms to isolate (sandbox) applications, e.g. docker.
- **System-level virtual machines**: the isolated environment resembles the hardware of a computer. Each VM runs its own independent OS, called the guest OS.

System-level Virtual Machines

- **Machine Simulator** – typically a user-level application, providing accurate simulation of virtualized architecture. Runs at small fraction of native speed.
- **Hypervisor** – Relies upon **limited direct execution** for maximum efficiency, relying upon traps which are emulated by the hypervisor.
  - **Bare-metal Hypervisor (type-1)** – Hypervisor is in direct control of all resources of the physical machine.
  - **Hosted Hypervisor (type-2)** – operates as part of or on top of an existing host OS.

Classification of Virtual Machines


Abstractions
Virtual Machine
Language-based Virtual Machine
(system-level) Virtual Machine
Lightweight Virtual Machine
Platforms
Machine Simulator
Hypervisor
Popek / Goldberg Th.
Bare-metal Hypervisor (type-1)
Hosted Hypervisor (type-2)

Popek / Goldberg Theorem

The processor's system state, called the **processor status word (PSW)** consists of the tuple (M, B, L, PC):

- the execution level $M = \{s, u\}$ (superuser or user mode)
- the segement register (B,L); (Segmented Memory Model) and
- the current program counter (PC), a virtual address

A virtual machine monitor may be constructed if the set of sensitive instructions for a computer is a subset of the set of privileged instructions.

$$\{\textit{control-sensitive}\} \cup \{\textit{behavior-sensitive}\} \subseteq \{\textit{privileged}\}$$

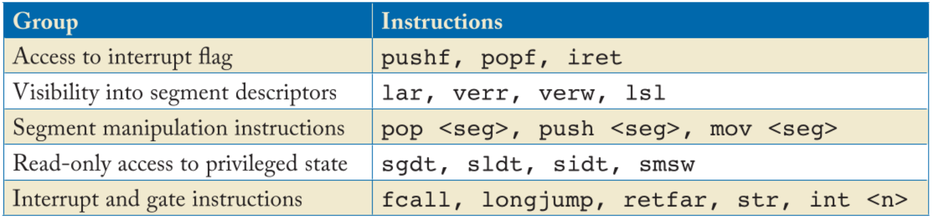
- **control-sensitive** – instructions that can update the system state.
- **behavior-sensitive** – instructions whose behavior depends upon the system state.
- **privileged** – instructions that can only be executed in supervisor (i.e. kernel) mode and causes a trap when attempted from user mode.

Criteria/Goals for Virtualization

- **Equivalence** – The exposed resource is equivalent with the underlying computer.
- **Safety** – Isolation requires that the virtual machines are isolated from each other as well as from the hypervisor.
- **Performance** – The virtual system must *show at worst a minor decrease in speed.*

Older Hardware Did Not Meet Popek / Goldberg Criteria

Table 2.2: List of sensitive, unprivileged x86 instructions



Group	Instructions
Access to interrupt flag	`pushf, popf, iret`
Visibility into segment descriptors	`lar, verr, verw, lsl`
Segment manipulation instructions	`pop <seg>, push <seg>, mov <seg>`
Read-only access to privileged state	`sgdt, sldt, sidt, smsw`
Interrupt and gate instructions	`fcall, longjump, retfar, str, int <n>`

Para-virtualization

Running a modified OS to run on a VMM. This is a way to give the VMM information to improve performance or provide safety because the environment is not completely isolated. Research shows that a properly-designed para-virtualized system, with just the right OS changes, can be made to be nearly as efficient a system without a VMM.

- Example 1 – OS Idling – VMM notices switch to low-power mode so vm will not be scheduled
- Example 2 – demand zeroing – modify the OS to not zero pages because VMM will do this.

Information Gap

VMM often doesn't know too much about what the OS is doing or wanting:

Example 1 – OS is idling:

```
while (1)
   ; // the idle loop
```

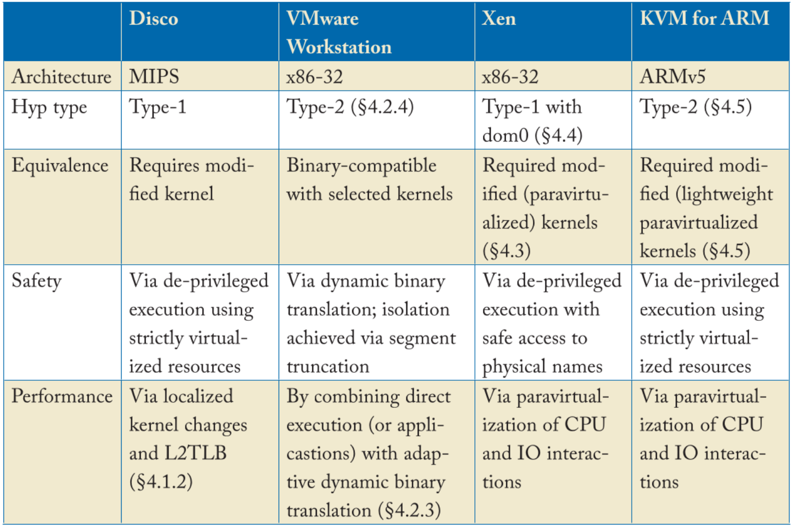
Example 2 – Allocating Pages:

Demand zeroing of pages before mapping into a process's page table.

Xen

Xen uses paravirtualization which sacrafices aspects of the equivalence property for a higher degree of efficiency or scalability. Xen simply undefined all of the 17 non-virtualizable instructions which must not be used. As a replacement, Xen defines an alternate interrupt architecture which consists of explicit calls from the guest OS to the hypervisor.

Virtualizing Without Hardware Support



	Disco	VMware Workstation	Xen	KVM for ARM
Architecture	MIPS	x86-32	x86-32	ARMv5
Hyp type	Type-1	Type-2 (§4.2.4)	Type-1 with dom0 (§4.4)	Type-2 (§4.5)
Equivalence	Requires modified kernel	Binary-compatible with selected kernels	Required modified (paravirtualized) kernels (§4.3)	Required modified (lightweight paravirtualized kernels (§4.5)
Safety	Via de-privileged execution using strictly virtualized resources	Via dynamic binary translation; isolation achieved via segment truncation	Via de-privileged execution with safe access to physical names	Via de-privileged execution using strictly virtualized resources
Performance	Via localized kernel changes and L2TLB (§4.1.2)	By combining direct execution (or applicastions) with adaptive dynamic binary translation (§4.2.3)	Via paravirtualization of CPU and IO interactions	Via paravirtualization of CPU and IO interactions

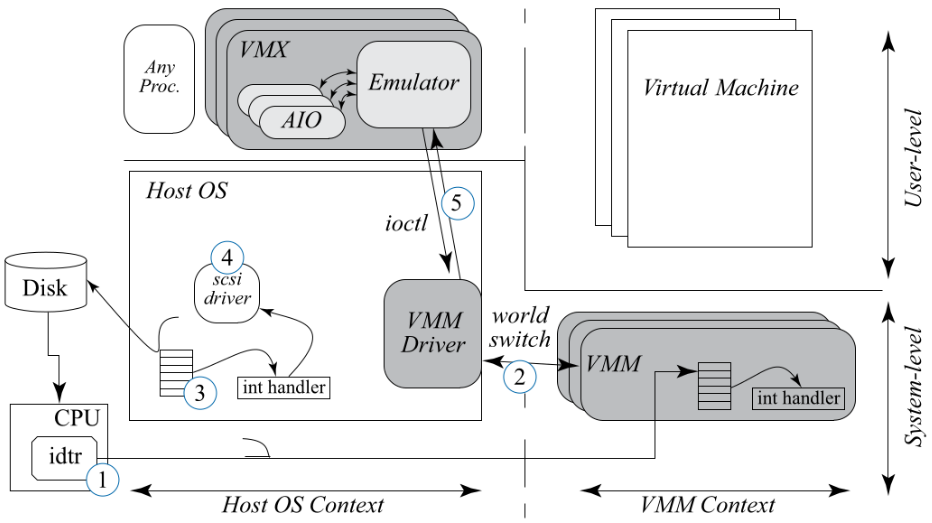
VMWare

VMM combined a direct execution subsystem to run guest applications and a dynamic binary translator (DBT) for running the guest operating system.

The DBT compiles a group of instructions, a basic block, into a fragment of executable code. Code is stored in large buffer called translation cache then use chanining, which allows direct jumps between compiled fragments.

VMWare's virtualization with DBT and shadow paging introduced a degree of atypical complexity to the current system software.

VMWare Virtualization



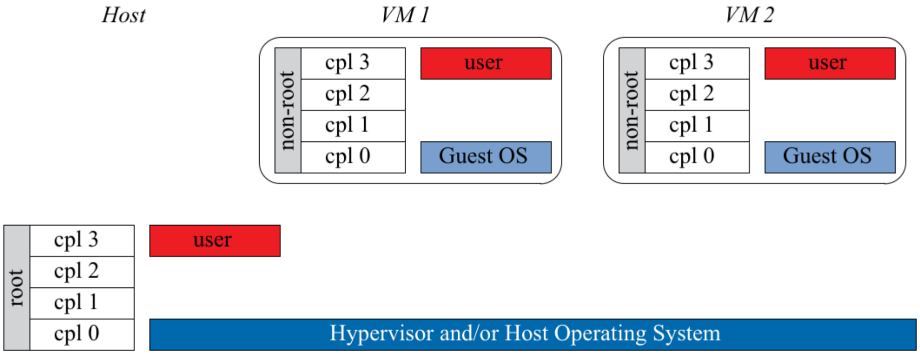
x86-64-bit virtualization with VT-x

True Hardware Support meeting Popek / Goldberg Criteria

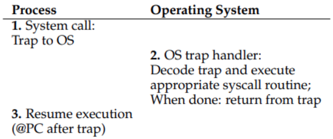
Do not change the semantics of individual instructions, instead duplicate the entire visible state and introduce a new mode of execution: the **root mode**.

- Process is in root mode or non-root mode.
- Special new instructions for detecting mode (only available in root mode, otherwise a trap is caused).
- New mode only used for virtualization
- Each mode has own address space
- Each mode has own interrupt flag

VT-x Architecture

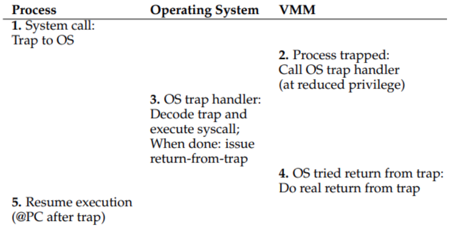


Virtualizing the CPU (System Call)



Process	Operating System
1. System call: Trap to OS	
	2. OS trap handler: Decode trap and execute appropriate syscall routine; When done: return from trap
3. Resume execution (@PC after trap)	

Figure B.2: **System Call Flow Without Virtualization**

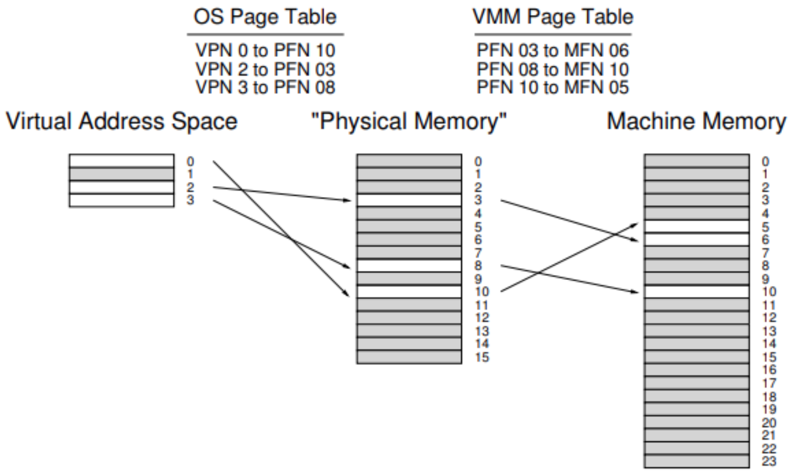


Process	Operating System	VMM
1. System call: Trap to OS		
		2. Process trapped: Call OS trap handler (at reduced privilege)
	3. OS trap handler: Decode trap and execute syscall; When done: issue return-from-trap	
		4. OS tried return from trap: Do real return from trap
5. Resume execution (@PC after trap)		

Figure B.3: **System Call Flow with Virtualization**

Anytime the underlying machine tries to perform a privileged instruction, a trap occurs and the VMM can emulate the behavior of the OS or call back into the OS at a reduced privelege level. This is known as the **trap and emulate** model.

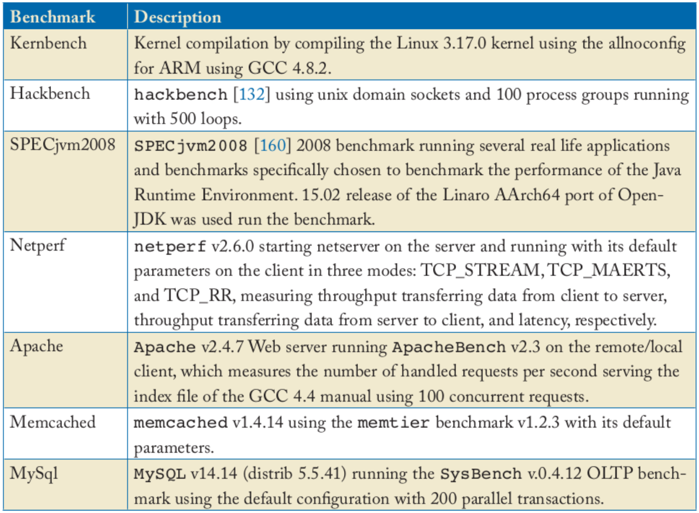
Virtualizing Memory (Page Fault)



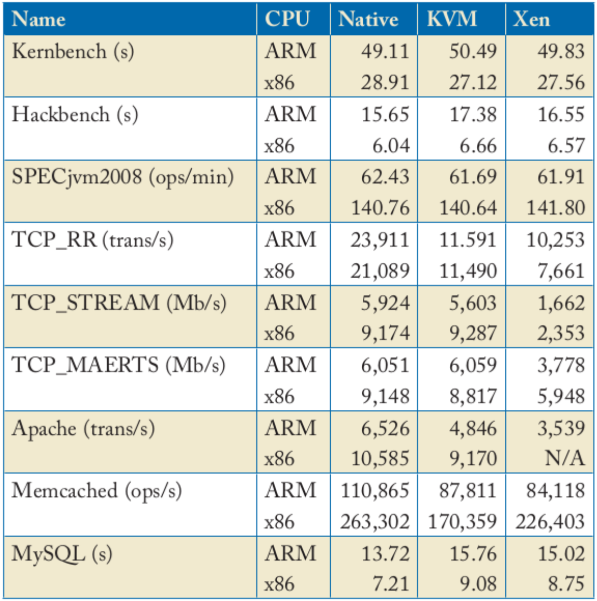
Process	**Operating System**	**Virtual Machine Monitor**
1. Load from mem TLB miss: Trap		
		2. VMM TLB miss handler: Call into OS TLB handler (reducing privilege)
	3. OS TLB miss handler: Extract VPN from VA; Do page table lookup; If present and valid, get PFN, update TLB	
		4. Trap handler: Unprivileged code trying to update the TLB; OS is trying to install VPN-to-PFN mapping; Update TLB instead with VPN-to-MFN (privileged); Jump back to OS (reducing privilege)
	5. Return from trap	
		6. Trap handler: Unprivileged code trying to return from a trap; Return from trap

7. Resume execution
(@PC of instruction);
Instruction is retried;
Results in TLB hit		

Performance

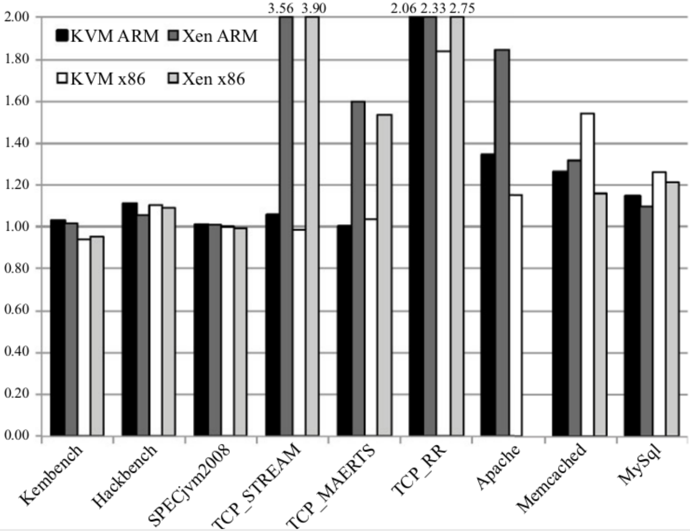


Benchmark	Description
Kernbench	Kernel compilation by compiling the Linux 3.17.0 kernel using the allnoconfig for ARM using GCC 4.8.2.
Hackbench	`hackbench` [132] using unix domain sockets and 100 process groups running with 500 loops.
SPECjvm2008	`SPECjvm2008` [160] 2008 benchmark running several real life applications and benchmarks specifically chosen to benchmark the performance of the Java Runtime Environment. 15.02 release of the Linaro AArch64 port of OpenJDK was used run the benchmark.
Netperf	`netperf` v2.6.0 starting netserver on the server and running with its default parameters on the client in three modes: TCP_STREAM, TCP_MAERTS, and TCP_RR, measuring throughput transferring data from client to server, throughput transferring data from server to client, and latency, respectively.
Apache	`Apache` v2.4.7 Web server running `ApacheBench` v2.3 on the remote/local client, which measures the number of handled requests per second serving the index file of the GCC 4.4 manual using 100 concurrent requests.
Memcached	`memcached` v1.4.14 using the `memtier` benchmark v1.2.3 with its default parameters.
MySql	`MySQL` v14.14 (distrib 5.5.41) running the `SysBench` v.0.4.12 OLTP benchmark using the default configuration with 200 parallel transactions.



Name	CPU	Native	KVM	Xen
Kernbench (s)	ARM	49.11	50.49	49.83
	x86	28.91	27.12	27.56
Hackbench (s)	ARM	15.65	17.38	16.55
	x86	6.04	6.66	6.57
SPECjvm2008 (ops/min)	ARM	62.43	61.69	61.91
	x86	140.76	140.64	141.80
TCP_RR (trans/s)	ARM	23,911	11.591	10,253
	x86	21,089	11,490	7,661
TCP_STREAM (Mb/s)	ARM	5,924	5,603	1,662
	x86	9,174	9,287	2,353
TCP_MAERTS (Mb/s)	ARM	6,051	6,059	3,778
	x86	9,148	8,817	5,948
Apache (trans/s)	ARM	6,526	4,846	3,539
	x86	10,585	9,170	N/A
Memcached (ops/s)	ARM	110,865	87,811	84,118
	x86	263,302	170,359	226,403
MySQL (s)	ARM	13.72	15.76	15.02
	x86	7.21	9.08	8.75

Performance Results Normalized


■ KVM ARM
■ Xen ARM
□ KVM x86
□ Xen x86
Kernbench
Hackbench
SPECjvm2008
TCP_STREAM
TCP_MAERTS
TCP_RR
Apache
Memcached
MySql