CS 695: Virtualization and Cloud Computing

Raja Gond

Last updated February 23, 2023

§1. Introduction

1.1. Virtualization and cloud computing

03/01/2023

`What is the cloud?`

Commodity servers with lots of computing and storage, connected with high-speed networking, located in data centers

- collection of servers/resources (centralised/distributed)
- On-demand resources and it is not managed by the user itself
- Networked / internet-scale applications
- pay-on-use model. Real-life analogies (Rental Business e.g Ola, uber and electricity usage)

There are two views. a. Client View → pay-on-use, on-demand and performance. b. Provider View → efficiency

`What is virtualization?`

- Multiple virtual machines (VMs) can run inside a physical machine (PM)
- VM gives user an illusion of running on a physical machine
- Containers are like lightweight VMs

Virtualization is a building block for cloud computing. It enables multiple clients share the cloud's compute resources and multiple users on VMs/containers can share same cloud server. Cloud storage/big data systems for efficient storage and retrieval of data. In addition to compute, clouds also manage large amounts of data.

`Why cloud computing?`

Public cloud providers (Amazon AWS, Microsoft Azure, Google Cloud etc) setup and maintain data centers with high-end servers

computing-as-service(a. client and b. provider)

- Powerful CPUs, lots of memory, disk storage etc., available to users
- Organizations can also run a private cloud only for their users

- `Why run applications on cloud and not on "bare metal" servers?`

- Multiplexing gains: multiple VMs can share the system resources
- Lower overhead of maintenance: hardware/software maintained by providers
- Flexibility: VMs can move to another machine if one fails
- Pay as per usage: no need to invest in servers if only lightly used

- `Disadvantages of running applications on cloud`

- Performance: longer delay to access servers via internet
- Higher cost if heavily used

1.2. Virtualization Terminology

System virtualization: System virtualization is a technique that allows a single physical computer or server to run multiple operating systems or multiple instances of a single operating system simultaneously. This is achieved by using virtualization software, which creates a virtual version of the hardware on which the operating systems can run. This allows multiple operating systems to share the underlying hardware resources of the computer, such as the CPU, memory, and storage, while still running independently of each other. System virtualization can be useful for a variety of applications, such as running multiple operating systems on a single computer for testing or development purposes, or running multiple instances of an application on a single server to improve its performance and scalability. **Process virtualization** (e.g., Java virtual machine) which lets a single process run on a different architecture from underlying machine

Hypervisor or virtual machine monitor (VMM): a piece of software that allows multiple VMs to run on a PM

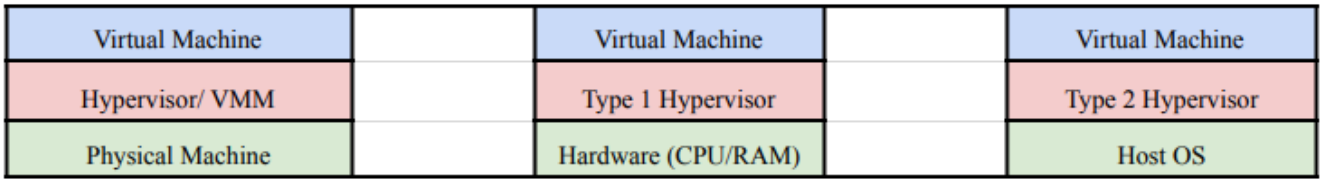

Figure 1: Virtualization

Guest OS runs inside the VM, and host OS runs on the PM
Type 1 hypervisor: runs directly on the hardware, no need for host OS
Type 2 (hosted) hypervisor: runs as an application on top of host OS

1.3. Challenges to virtualization

Guest OS expects complete control over hardware, but VMM must multiplex multiple guests on the same hardware

- Understand how operating systems work (prerequisite)
- How to trick the guest OS into relinquishing hardware control?

We will study the following ways to design virtual machine monitors. Understand how CPU, memory, I/O devices are virtualized with each of the above techniques

- Hardware-assisted virtualization (e.g., KVM/QEMU): modern CPUs have support for virtualization and VMMs are built over this support
- Full virtualization (e.g., VMWare): Original technique to run unmodified OS over original hardware with no virtualization support
- Paravirtualization (e.g., Xen): Modify OS source code to be compatible with virtualization

1.4. Introduction to Cloud Computing

Architecture of cloud applications: compute and storage options

- Compute in VMs, containers etc.
- Traditional storage in databases, now moving to simpler key-value stores etc.

Cloud storage techniques

- In-memory key-value stores (Amazon Dynamo)
- Semi-structured data storage (Google Bigtable)

- Application-specific storage (Facebook's Haystack to store photos)
- Caching-based optimizations to the cloud storage layer (Facebook's memcache)

06/01/2023
`Cloud Computing`

- Computing as a service.
  - pay-on-use
  - on-demand
  - elasticity
  - h/w-ed
- There are different models which provide computing as a service.
  - **IaaS**: Infrastructure as a Service (VMs / Containers)
  - **SaaS**: Software as a Service
  - **PaaS**: Platform as a Service
  - **FaaS**: Function as a Service

  All of these need to be provided with client requirement with efficiency and cost features. Providers ends up using mechanisms- abstractions/implementation, tools optimization and resources management.
- We will focus on **IaaS** → Virtualization/VMs, Containers.

What is VM abstraction? What is an abstraction in compute world? Why abstraction?
Abstraction - logical entity of defined functionality and usage interface. Why? reuse, hide complexity(layering functionality). Examples → Car, bank account, Process - it allows breaking the work into manageable pieces. Address space, files, API.
Process decouples building and execution of the program.

`Application Binary Interface(ABI)`. why binary? Think of it as the compiled version of an API (or as an API on the machine-language level). When you write source code, you access the library through an API. Once the code is compiled, your application accesses the binary data in the library through the ABI. The ABI defines the structures and methods that your compiled application will use to access the external library (just like the API did), only on a lower level. Your API defines the order in which you pass arguments to a function. Your ABI defines the mechanics of how these arguments are passed (registers, stack, etc.). Your API defines which functions are part of your library. Your ABI defines how your code is stored inside the library file so that any program using your library can locate the desired function and execute it.

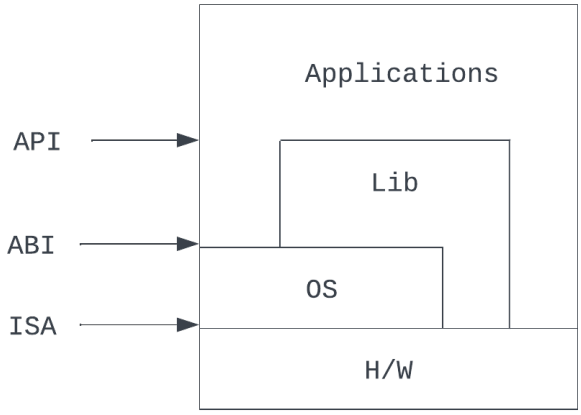

Figure 2: Systems view

transistors → gates → logic units → interconnects → ISA → drivers → OS → libraries → more libraries → applications!

1.5. Process Abstraction

Decoupling of program and program instance(binary)
`How does the OS support the process entity?`

- Metadata:- PCB - Process Control Block, store info. about the process such as pid, pgd - page directory, files
- allocates memory to store program + runtime state

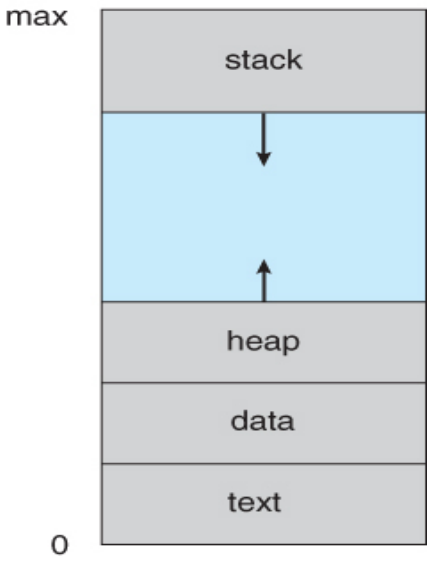

Figure 3: Memory Image

1.5.1. What is CPU state when a process executes?

- Registers
  - PC/eip - Program counter ← pointer to the next/current instruction
  - sp/esp - top of the stack ← storing functions parameters (arguments, return values)
  - CR_x - Control Registers e.g. CR3 points to the page table
  - eax/ebx/ecx - general purpose registers - process context on the cpu

save & restore process context enables multiplexing of processes ← many processes can time share the cpu

1.5.2. process and systems view

Process View	Systems(OS) View
- No other processes - CPU owner - Zero starting address space - files, fs - Network endpoints - Device endpoints	- CPU type - ISA - disk, disk types physical addresses(NVMA0 and NVMA1) Protection mechanism for memory access device specs/ h/w support / privilege levels

1.6. Main-Building Blocks of an OS for providing abstractions?

OS is the sole owner of all resources!

1. Interrupts/interrupt-driver execution
2. (privileged) modes of execution (per instruction)
3. system call interface

IDT - Interrupt Descriptor Table. `idtr` switching is required when we switch VM.

1.7. New abstraction called Virtual Machine(VM) (Lecture # 2) 11/01/2023

Environment to allow the OS-view

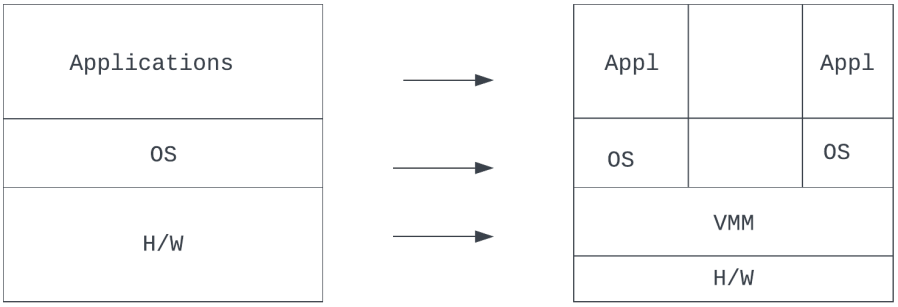

Figure 4: should be viewed as (left) but provided as (right) this view is consistent with physical machine

1.7.1. Why virtual Machines?

- Building blocks for Iaas (virtual compute infrastructure, on-demand, flexible)
- deeper isolation semantics
- multi ISA, multi-OS runtimes on a single physical machine.
- Debugging OS level!
- new runtime execution: migration, snapshot/checkpoints, record-replay and features(maintenance & upgrades)
- VM image introspection

1.7.2. VMM design

- Categories of Hypervisors
  - Bare Metal(Type 1) (vmware esx, xen) vs. Hosted(Type 2) (KVM, vmware VirtualBox)
  - Full-Virtualization(vmware esx) vs Para Virtualization(xen, citrix-xen, KVM)
- Challenges of design?
  - Iass ← compute ← CPU, memory, IO
  - ownership - CPU, devices
    Who is the owner? Guest OS believes that it is the owner. VMM has to be the owner.
    e.g.
    a. update to CR3
    - OS can write any value - access other VM's memory

b. handling system calls (ABI)

Application - x86
fopen → open → %eax ← syscall no
int 0x80 ← interrupt of vector 0x80
change in privilege mode(user to kernel)
jmp to the interrupt handler

```
system_call_handler(){
    fn = sys_call_table[eax]; //eax has system call no
    call fn;
    iret;
}
```

- Device virtualization → devices cannot be chopped up (without breaking them physically)

- Principles of VMM design
  1. efficiency → performance
  2. resource control → ownership
  3. equivalence → fidelity → an OS/application should behave equivalently on a PM/VM.

§2. Design of CPU Virtualization with VMs by VMM (Lecture # 3) 13/01/2023

2.1. Trap & Emulate/Virtualize

Native	VMM
Ring 3 (user mode) applications	Ring 3 - applications
Ring 0 (kernel mode) OS	Ring 1 - Guest OS Ring 0 - VMM

In the x86 ISA, **S**: Set of sensitive instructions

- they influence "system" behaviour
- they need certain special privileges. → CPU always operates with different Privilege Level configurations. CPL - Current execution PL ⇒ CPL = 0 or CPL < req. PL. If not enough privilege CPU generates a trap. OS handles the trap.

Guest OS issues a write to CR3(CR3 - points to the physical page)

- Trap!!
- VMM can handle (key to virtualize CR3 - error checks - bound checks) the trap.
- x: guest OS ↔ x+200: vmm

2.1.1. Issues

1. VMM needs to know the semantics of the OS state! e.g interrupt handlers.
2. all sensitive instructions do not generate a trap.

   C: critical instructions - **S** which do not generate a trap on not enough privileges.

   `popf` (with enough privileges - no trap - no update):- which pops the top of the stack and updates the EFLAGS (bit # 9 enable/disable interrupts) registers.

   breaks equivalence and missed the virtualization opportunity

3. `mov %cs, memlocation`, `mov %ss, memloc` (examine 2 LSB bits to find the CPL.)
   `sidtr` - stores the location of `idt` in a memory address
   information crosses layers with no/invalid semantics.

efficiency - ?, control - x/?, equivalence - x

2.2. Scan & Patch

vmware esx - binary translation

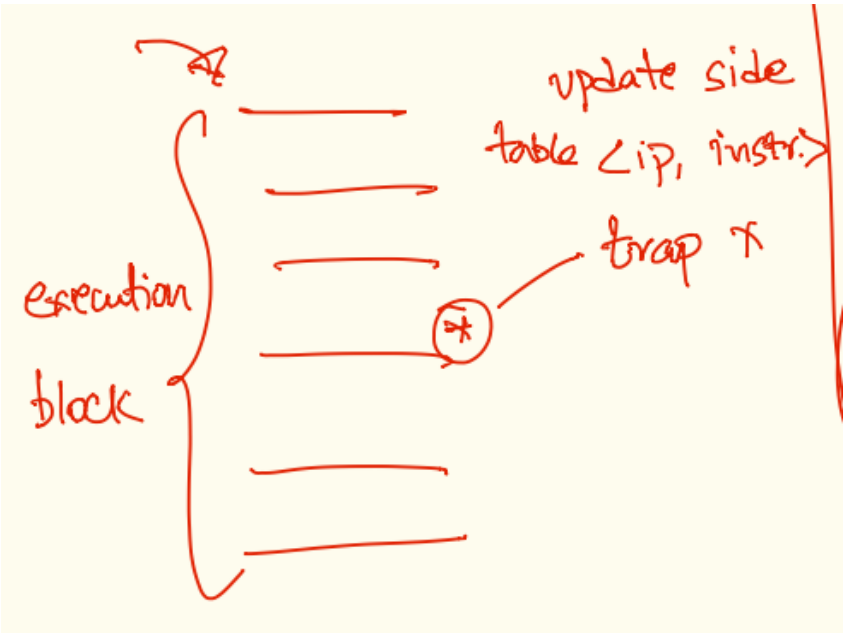

Figure 5: Scan & Patch

CPU: $\overleftrightarrow{VMM}\overleftrightarrow{Patched\ OS}\overleftrightarrow{VMM}\overleftrightarrow{Patched\ OS}$
efficiency - ?, control - y, equivalence - y

2.3. Para Virtualization

Xen

- Don't issue critical instructions.
- Request(through hypercall - ABI e.g mmu_update(CR3, ...)) hypervisor for each sensitive task.
- (Guest) OS knows that the hardware is being virtualized.

efficiency - y, y control - y, equivalence - x

2.4. Hardware-assisted CPU Virtualization

Intel VT-X, AMD-V

1. VMX mode (root — PL3 - PLO, not-root — PL3 - PL0)
2. VMX instructions
3. VMX state
4. VMX commands (vmresume, vmlaunch, vmexit, vmptrld - load pointer to vmcs)
5. vmcs state (VM config is saved and restored as per vm basis)

2.4.1. Operations

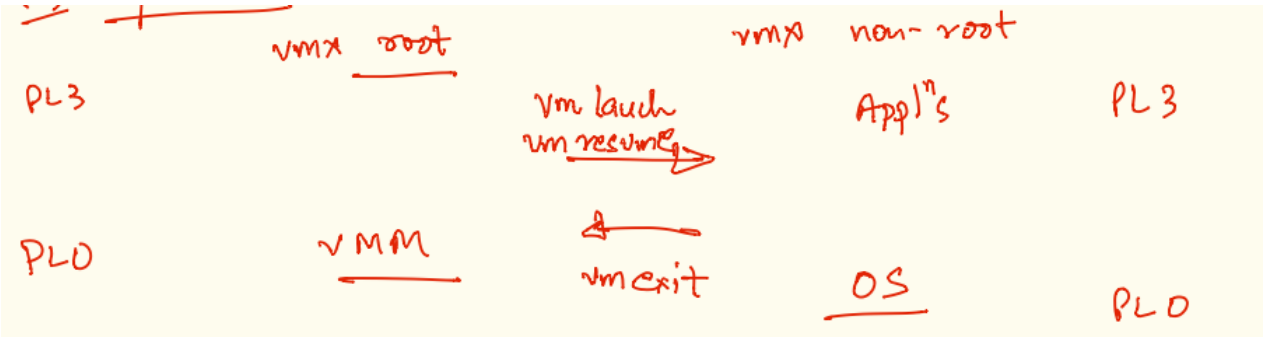

Figure 6: vmx - operations

1. non-root (execution context) is configured by VMM.
2. program/configure the VMexit conditions.

2.4.2. Configurable settings of VMEXIT

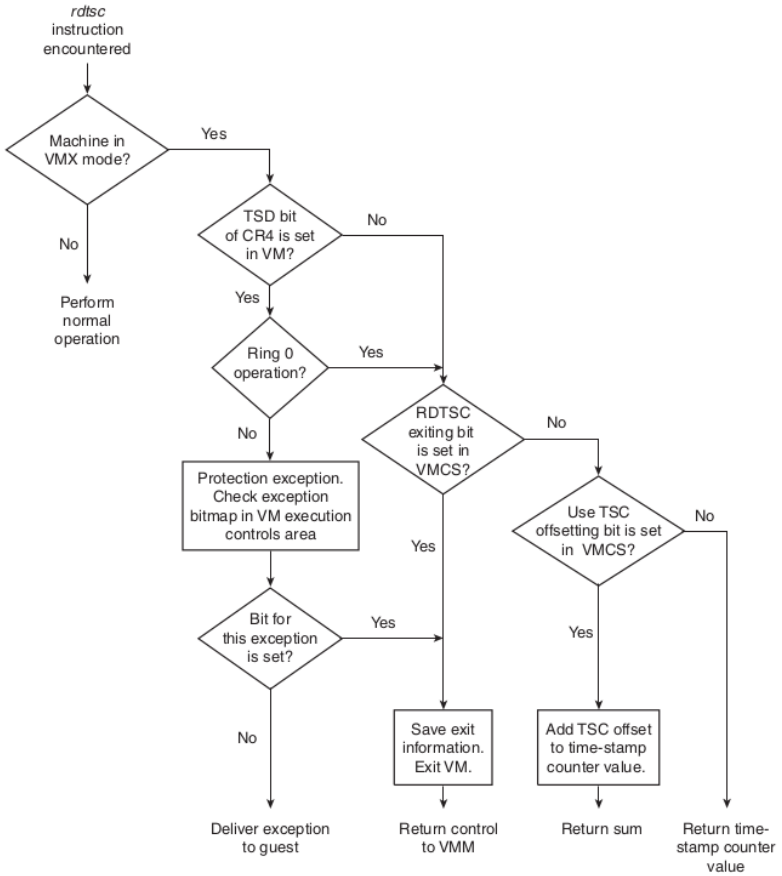

Figure 7: Actions Taken by Hardware When a Read Time-Stamp Counter (rdtsc) Instruction Is Encountered.

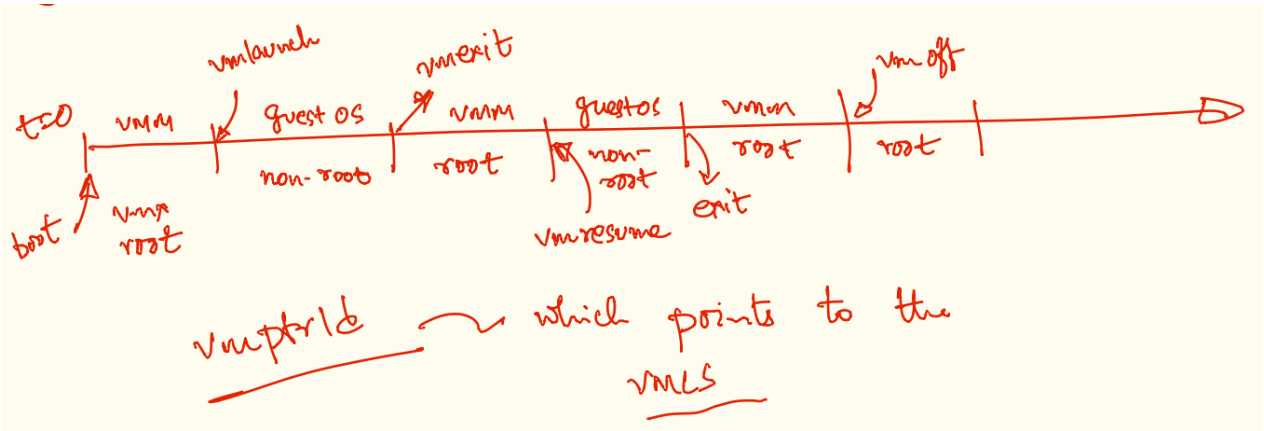

Figure 8: Time-Stamp Execution

2.4.3. VMCS - Virtual Machine Control Structure (in-memory data structure)

- `guest area` - to save guest state on exit
- `host area` - to save host state(context of host) on entry
- `vm exit info` - area to store state on exit
- `vm execution control` - when(conditions) to exit
- `vm exit control` - what to store on exit
- `vm entry control` - what to store on entry

Every VM has its own VMCS, only updated by hypervisor(through vmx command) or VMEXIT(hardware)

2.4.4. vmcall - hardware assisted hypercall

Time is little tricker in VM. Coupling with hardware makes VM time notions a little problematic. Native OS gets the notion of time using a timer interrupt and TSC. rdtsc - read time stamp counter (issue this from inside a VM).

§3. Memory Virtualization with VMs (Lecture # 4) 18/01/2023

3.1. Native system

- virtual address space
- starts at address 0 and linear
- per process
- isolated

All done through segmentation + paging.

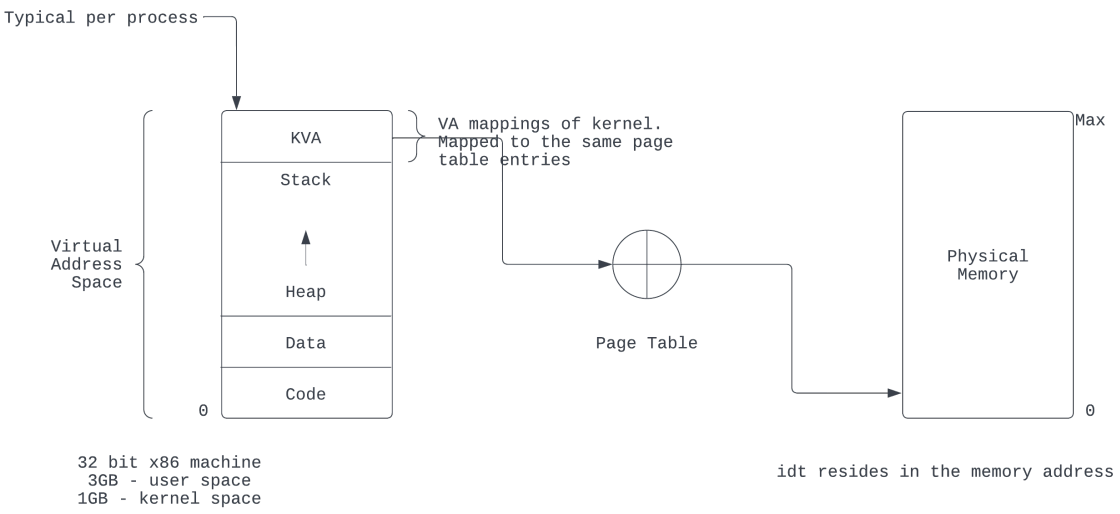

Figure 9: Memory view

The kernel also accesses addresses through the page table. Hardware assistance is required for address space abstraction.

3.1.1. Hardware assistance for address space abstraction

CPU modes

- real mode - no virtual memory. all memory accesses are physical addresses. access range(size): 1MB. drastically reduced instructions set
- protected mode - whole 32-bit, privilege levels, memory access/protection(segmentation) + paging ((virtual) address space $\rightarrow$ v2p via the mmu). 4KB pages in 32-bit.

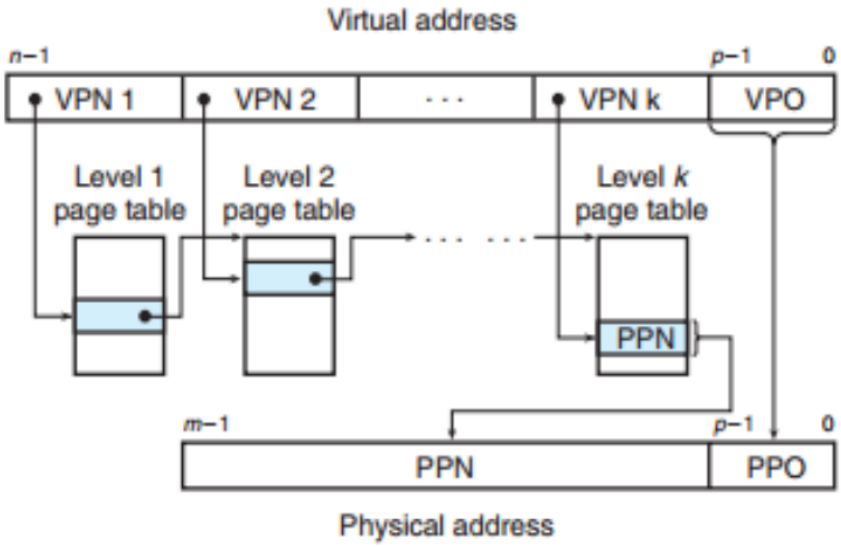

Figure 10: Page Table

MMU looks up CR3 register of CPU (x86) which stores the location of the page table of the current process

  - Looks up page number in page table to translate an address
  - CR3 reset on every context switch

  PTE: Page Table Entry. MSB 20bits - specify a page start address (PPN)[specify where to find this page if swapped]. LSB 12bits - pte flags.

- page fault
  - exception (handle through trap handler). CR2 register stores the VA
  - re-execute the faulty instruction.

3.2. Memory Virtualization for VMs (Lecture # 5) 20/01/2023

How does the processor gets the information for setting up the guest, what registers it should load and which interrupts or CPU instructions cause a VM-Exit? For all this information the VMM needs the Virtual Machine Control Structure (VMCS) which holds all the information about the configuration and which rules it has to obey. So, how does this structure looks like?

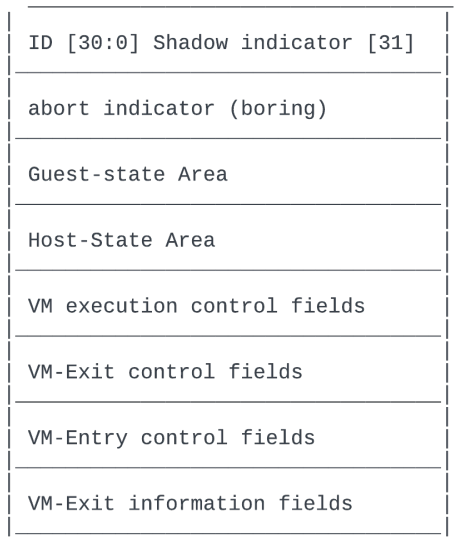

Figure 11: VMCS

- per VM per cpu
- updated by VMM(root + ring 0) or CPU (h/w)

3.2.1. Address Space Abstraction

- 0 starting, linear, isolation, per process (via paging + segmentation support)
- h/w assistance: mmu, privilege modes, CR3, CR0, CR2, pte flags, page fault/trap (address is in CR2)
- the same /constant OS view is required for virtualization
- memory virtualization/abstraction mechanism → paging + swapping.
- memory management policy(LFU, LRU, ARC)

3.2.2. How to provide this view/support for VMs?

`Multiplexing`: creates multiple virtual objects from one instance of a physical object. Many virtual objects to one physical. For example - a processor is multiplexed among a number of processes or threads. `Aggregation`: creates one virtual object from multiple physical objects.

- Multiplexing VMs! VM memory! (all physical memory is not available to an OS/VM.
- Hypervisor promises the VM (for e.g. 4 CPUs, 32 GB memory)
  Design:
  1. Time multiplexing of memory
     $\overset{vmm}{\longleftrightarrow}$ (physical memory)
     $\overset{vm1}{\longleftrightarrow}$
     $\overset{vmm}{\longleftrightarrow}$
     $\overset{vm2}{\longleftrightarrow}$
     (very very costly)
  2. Spatial Multiplexing
     $\overset{vmm}{\longleftrightarrow}\overset{vm1}{\longleftrightarrow}\overset{vm2}{\longleftrightarrow}\overset{vm3}{\longleftrightarrow}$ (physical memory)

  OS believes that its physical memory starts from zero.

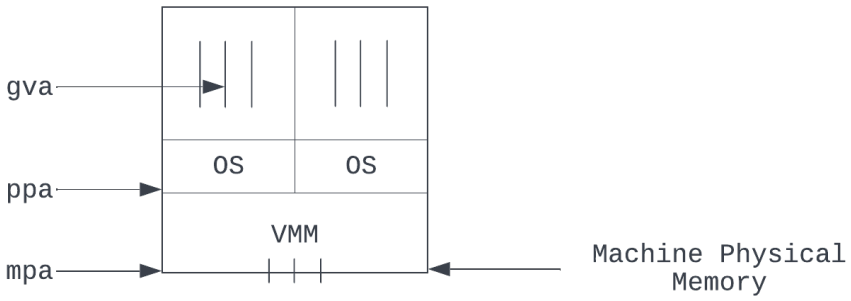

Figure 12: Memory mapping of VMs

gva $\xrightarrow[v2p]{OS-managed}$ ppa $\xrightarrow[p2m]{hypervisor-managed}$ mpa

gva - guest virtual address

ppa - pseudo physical address

mpa - machine physical address

1. mmu! can only do one translation
2. page faults can occur at more than one mapping failure.

3.3. Direct Mapping (Para-Virtualization approach)

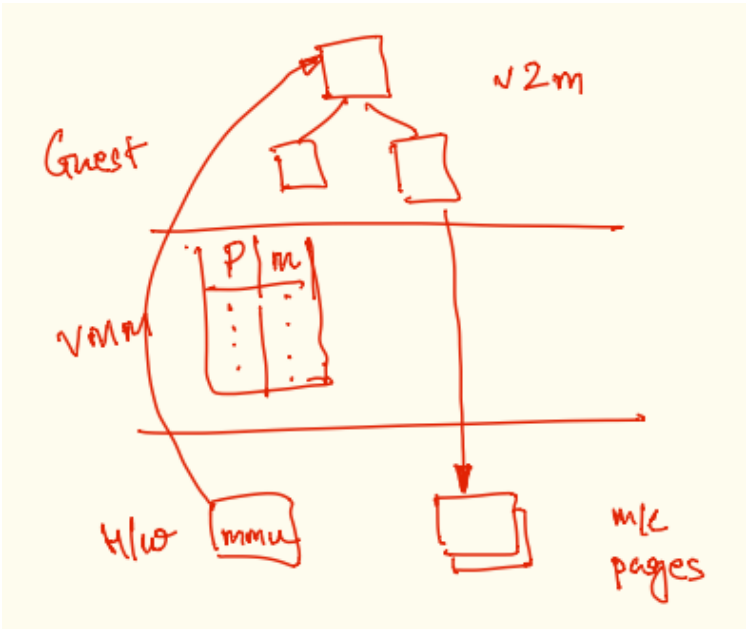

Figure 13: Memory mapping of Direct mapping (PV approach)

v2p, p2m $\xrightarrow[\text{via a single mmu}]{}$ v2m

- How to build v2m mapping?
  - all guest page tables are write-protected.
  - guest update to page tables via the mmu.

  E.g. Through hypercall $update\_pagetable(gva, ppa, pgd)$, ppa $\rightarrow$ mpa.
  Update page table indexed at pgd with gva $\rightarrow$ mpa.
- How does VMM knows which pages in the ppa range are page table pages?
  Guest OS tells the hypervisor! (para-virtualized)
- How to handle page faults?
  - On Fault, VMM handler is invoked.
  - VMM detects that is a guest page fault.
  - "Pass" the handler to the guest(via the CPU CR registers(interrupt state).
  - guest handles the page fault.
  - makes hypercall for pgtable update.
  - Instruction resumes after hypercall return.

  E.g.

Native	VMM
CR3 $\leftarrow$ 1000(ppa)	CR3 $\leftarrow$ 10000
va, pa, pgd $\leftarrow$ 4000, 9000, 1000	$\xrightarrow[\text{Looks up p2m state mapping}]{hypercall}$ 4000, 90000, 10000
	(Update using pages in mpa space)

3.4. Shadow Paging (Full Virtualization, Binary Translation) (Lecture # 7) 27/01/2023

gva $\xrightarrow{v2p}$ ppa $\xrightarrow{p2m}$mpa
Shadow page table – indirection between guest physical and host physical memory
- Every time guest OS changes its page tables, the hypervisor must change shadow page tables as well. Flexible mechanism used by VMM to maintain control
`Problem`: If guest OS remaps virtual page 7 onto what it sees as physical page 100 (instead of 10), the hypervisor has to know about this change. Trouble is that guest OS can change its page tables by just writing to memory. No sensitive operations are required

`Solution:` hypervisor keeps track of which page in the guest's virtual memory contains a top-level page table

- VMM responsible for mapping guest physical memory to actual machine memory
- VMM uses shadow page tables to accelerate mappings
- VMM uses TLB hardware to map virtual memory directly to machine memory to avoid
- Avoid two-level translation on every access
- When guest OS changes virtual memory to physical memory mapping, VMM updates shadow page tables to enable direct lookup
- MMU virtualization creates some overhead for all virtualization approaches, but this is area where 2nd generation hardware-assisted virtualization will offer efficiency gains

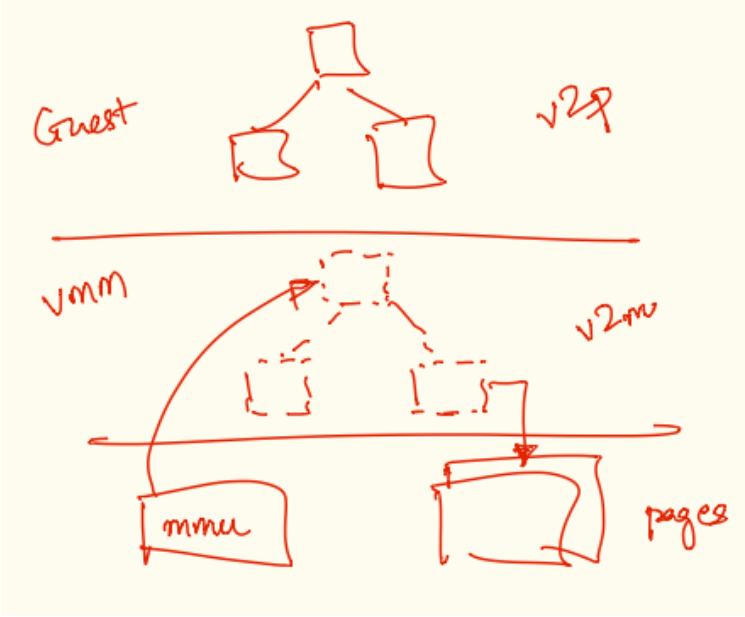

Figure 14: Shadow Paging

CR3 is virtualized. VMM needs to intercept when guest OS modifies the page table and updates the shadow page table accordingly

1. Mark the guest table pages as read-only (in the shadow page table)
2. If guest OS tries to modify its page tables, it triggers a page fault
3. VMM handles the page fault by updating the shadow page table with v2m mapping
4. Every update to a page table ⇒ update to two-page tables.

3.4.1. What if a Guest App Access its Kernel Memory?

How do we selectively allow/deny access to kernel-only pages?
One solution:

- split a shadow page table into two tables
- Two shadow page tables, one for the user, one for the kernel
- When guest OS switches to guest applications, VMM will switch the shadow page table as well, and vice versa

3.5. Hardware-Assisted Memory Virtualization

- Hardware support for memory virtualization
  - Intel EPT (Extended Page Table) and AMD NPT (Nested Page Table)
  - EPT: a per VM table translating PPN → MPN, referenced by EPT base pointer
  - EPT controlled by the hypervisor, guest page table (GPT) controlled by guest OS (both exposed to hardware)
  - Hardware directly walks GPT + EPT (for each PPN access during GPT walk, needs to walk the EPT to determine MPN)
  - No VM exits due to page faults, INVLPG, or CR3 accesses

  TLB: What if two VMs have the same virtual address v2m → process-centric ← Refresh TLB for every process switch.

  Some entries are process-centric and some are not (Global vs Local). ASID - Address specific ID
- Pros
  - Simplified VMM design (all handled by hardware)
  - Guest PT changes do not trap, minimize VM exits
  - Lower memory space overhead (no need for pmap in memory)
- Cons
  - TLB miss is costly: can involve many memory accesses to finish the walk!

More details will be added later maybe...

3.6. Nested Virtualization

We can use hardware-assisted virtualization at one label

3.7. Conclusion

- Software and hardware solutions for memory virtualization both have pros and cons
- More things to take care of besides the basic mechanism of memory virtualization
  - Allocation, sharing, overcommitment and reclamation

§4. Interrupt Handling with VMs and VMM (Lecture # 6) 25/01/2023

Interrupts/traps → External/IO interrupt → By signaling to the cpu
⟶ Exception
⟶ Explicit Interrupt(e.g int)
Exception and Explicit Interrupt comes out from software mechanisms such as Page Fault, NULL deref during instruction execution.

4.1. Native System (int processing)

fork, open → wrappers for system calls → eax, ebx · int 0x80
INT is an assembly language instruction for x86 processors that generates a software interrupt. It takes the interrupt number formatted as a byte value.
The current privilege level with which the x86 executes instructions is stored in %cs register, in the field CPL.
To make a system call on the x86, a program invokes the int n(e.g int 0x80 ← userspace) instruction, where n specifies the index into the IDT. The int instruction performs the following steps:
hardware sequencing

- Fetch the n'th descriptor from the IDT, where n is the argument of int.
- Check PL for integrity. Check that CPL in %cs is <= DPL, where DPL is the privilege level in the descriptor.
- Save %esp and %ss in CPU-internal registers, but only if the target segment selector's PL $<$ CPL.
- Load %ss and %esp (this is kernel stack) from a task segment descriptor.
- Push user %ss
- Push user %esp.
- Push user %eflags.
- Push user %cs.
- Push user %eip.
- Clear some bits of %eflags.
- Set %cs and %eip to the values in the descriptor.

⟵ jump to ISR
(An interrupt service routine (ISR) is a software routine that hardware invokes in response to an interrupt).
S/W sequencing

- Push error code
- Push trap/interrupt no.
  (Trap Information)
- Push ds, gs, es, fs (other segments)
- Push eax (system call no), argument info - bx, cx, dx (general purpose register)
- jmp $idt\_table[trapno]$ ⟵ system call handler (arrays of function)

```
system_call_handler(){
    fn = sys_call_table[eax]; //eax has system call no
    call fn;
    iret;
}
```

Kernel stack

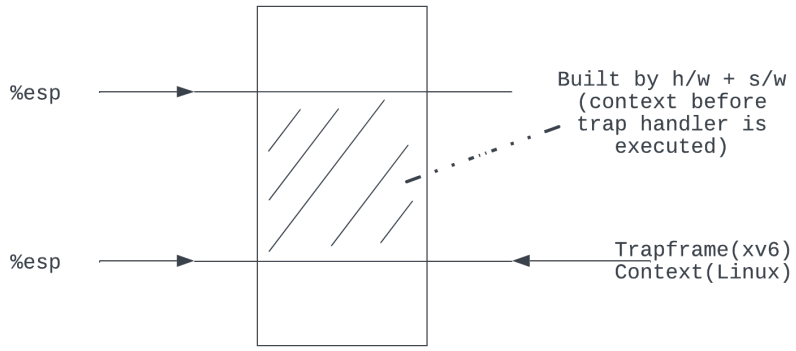

Figure 15: Kernel stack

`return path` is in the ISR

- pop all registers
- pop ds, gs, es, fs (segment registers)
- adjust %esp → to move over error & trap no.
- `iret` → h/w assisted instruction
  → rolls back %esp of the kernel
  → loads user's regs
  → jumps to cs:eip with PL=3

Note: `iret` can't work in ring 1

4.2. What happens to system calls from VMs?

4.2.1. Fully Virtualized VMs

binary translation	HVM
On interrupt - Control in VMM - Context on VMM stack - ISR uses the context to determine that this interrupt should be handled by guest - Emulate an interrupt - copying context from VMM to guest kernel stack ← (updates to %esp and %ss traps to VMM) - switch to guest at cs:eip of guest ISR	On interrupt - Control goes to VMM - VMM needs to pass interrupt to guest - guest executes the handler iret in guest traps to VMs VMM returns to guest userspace

4.2.2. Para-Virtualized VM

(VMM always know the stack pointer of guest OS. because the writing of stack pointer issue trap to vmm).
Similar in PV, but via hypercalls for the critical state.

4.2.3. Interrupts with h/w assisted VMs (diagram)

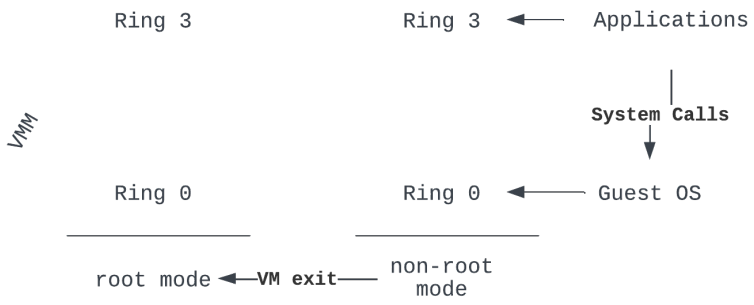

Figure 16: HVM - interrupt

- depends on the exit condition and IDT setup.

§5. IO Virtualization (Lecture # 8) 01/02/2023

5.1. Introduction

- System view of OS includes devices.
- Device (External Devices)

  CPU $\xleftrightarrow{\text{interconnects}}$ device (Device Drivers - Abstraction CPU used to talk with the devices.

  CPU $|_{\text{interface}}$ Device. Device drivers use the interface to communicate with the devices and OS uses the device driver to translate actions from its own subsystem.

There are at least three ways for the OS(including drivers) to communicate with...

1. **PIO** - Programmed IO (very few today use this)
   (a) Special ISA instructions to deal with IO.
   (b) in, out(port index, data) -serial port
2. **MM IO** - Memory Mapped IO
   (a) Physical Memory is mapped with device interface state.
   (b) Read/Write memory $\iff$ Read/Write to device register
3. **DMA**
   (a) where devices read/write directly from/to memory
   (b) setup by instruction on CPU by os.

Eg. Consider a disk with the following interface
reg x: sector number, reg y: R/W, reg z: VA(memory address) and in/out : instruction to state purpose
$x \leftarrow 10$, $y \leftarrow 0$, $z \leftarrow kva$ and in (read from sector 10 and copy to kva specified in kva)
in is a privileged instruction. $fopen \xrightarrow{fd} open \xrightarrow{\text{device block num}} Device\ Driver \xrightarrow{interface} Disk$

5.2. Generalised IO virtualization view with VMs

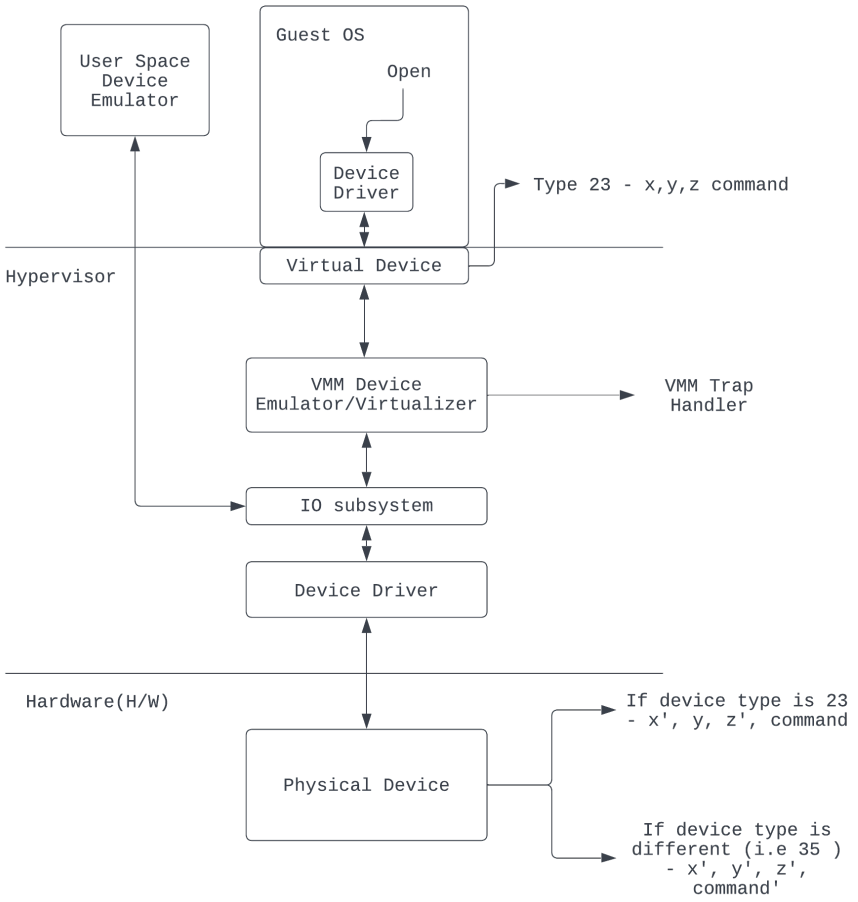

Figure 17: IO Virtualization view

- Types of Devices
  1. Dedicated (Temporal) - IO, Keyboard, Mouse
  2. Partitioned (Spatial) - Disk
  3. Shared - Network
- Virtual device may / may not be the same as the physical device
- Virtual device may be virtualized via different abstractions. E.g. Vdisk (vda - guest OS) ← file (hypervisor)

5.3. Full Device Virtualization (Fully Virtualized VMs)

- Guest OS does not know about vmm
- VMM virtualizes and handles/emulates the complete device interface
  - actual e.g e1000 NIC
  - 2 complete IO stacks
- Device type 23 - x,y,z, cmd

1. Privileged instruction generates a trap
2. Patch instruction to generate a trap
3. Write protect MMIO pages to generate traps

5.4. Paravirtualization (PV) IO virtualization

split device (driver) Model

- Device Type 23
  - $x \leftarrow 10 \xrightarrow[trap]{\text{Memory Mapped IO}}$
  - $y \leftarrow 0 \xrightarrow[trap]{\text{Memory Mapped IO}}$
  - $z \leftarrow kva \xrightarrow[trap]{\text{Memory Mapped IO}}$
  - $cmd\ regs \leftarrow 0/1 \xrightarrow[trap]{\text{Memory Mapped IO}}$

  Four traps but in the PV world - can become a single hyper call (hc-deviceIO(x,y,z, cmd)).
- Change device interface to a virtual/Non-real/Non-Physical interface.
- Send all control/+data(information) in a single hypercall.
- Pack multiple units of IO in a single call.
- Simplify device specification for greater efficiency.

5.4.1. Xen —— Virtio(kvm, hypervisor) —— PV hypervisor

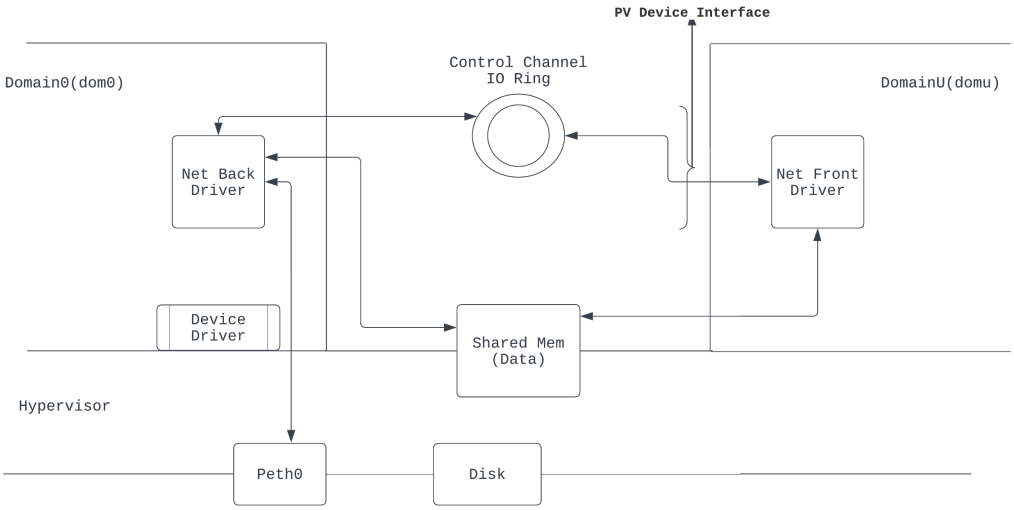

Figure 18: Xen view

1. How deep should the network stack be on the guest OS?
2. How does Dom 0 get/receive packets intended for Dom U?

Lecture #9 03/02/2023

- Dom 0 initializes per device event channel and shared memory.
- Dom u works with a simplified network interface, simplified IO stack and coalesce data to/fro
- Dom 0 contains the device driver for physical devices
- Dom 0 multiplexes all IO traffic
- *Paravirtualized Network Interface Card*(*PV NIC*)
  - Network (N/W) interface: MAC addr, IP addr

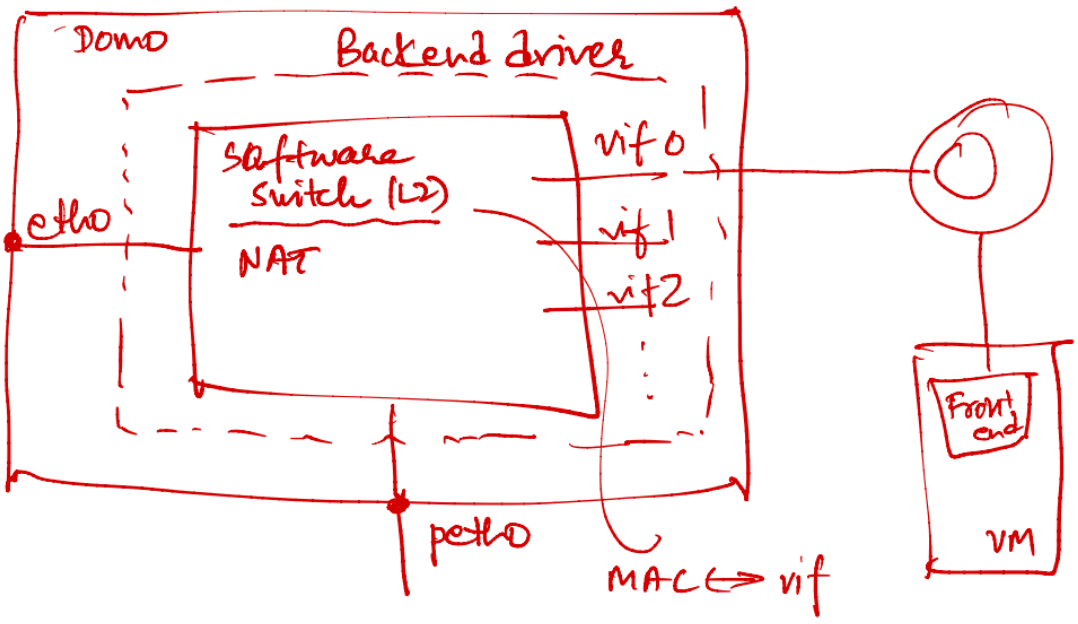

Figure 19: NIC

Lecture #10 08/02/2023 Cancelled

§6. Memory Resource Management in VMware ESX Server Lecture #11 10/02/2023 memmgmt

Abstract

+ Unmodified operating system
+ Ballooning technique reclaims pages considered to be least useful by guest OS
+ what is overcommit Memory? – Memory overcommitment is a concept in computing that covers the assignment of more memory to virtual computing devices (or processes) than the physical machine they are hosted, or running on actually has.

Introduction

+ server consolidation – Server consolidation is the process of combining multiple servers into a single physical server. The goal of server consolidation is to reduce the number of physical servers required to run an application or set of applications.
+ Proliferation - the rapid increase (prasaar)
+ resurgence - the return or growth of something that had stopped.
+ Virtual Machine - illusion of separate physical machine that is protected and isolated
+ Many small servers can be consolidated onto fewer large servers to simplify management and reduce costs.
+ statistical multiplexing - the process of sharing a single resource among multiple users or processes.

+ Virtual machines have been used for decades to allow multiple copies of potentially different operating systems to run concurrently on a single hardware platform
+ VMware ESX Server is a thin software layer designed to multiplex hardware resources efficiently among virtual machines.
+ VMware ESX Server is a type 1 hypervisor, which means that it runs directly on the hardware and is not dependent on any other operating system. ESX Server manages system hardware directly, providing significantly higher I/O performance and complete control over resource management. (Full Virtualization)
+ a background activity exploits opportunities to share identical pages between VMs, reducing overall memory pressure on the system

Memory Virtualization

+ A guest operating system that executes within a virtual machine expects a zero-based physical address space(i.e starting from zero), as provided by real hardware.
+ ESX Server maintains a pmap data structure for each VM to translate "physical" page numbers (PPNs) to machine page numbers (MPNs).
+ Separate shadow page tables, which contain virtual-to-machine page mappings, are maintained for use by the processor and are kept consistent with the physical-to-machine page mappings in the pmap data structure.
+ The server can remap a "physical" page by changing its PPN-to-MPN mapping, in a manner that is completely transparent to the VM.

Reclamation Mechanism

+ Overcommitment means that the total size configured for all running virtual machines exceeds the total amount of actual machine memory
+ Since commodity operating systems do not yet support dynamic changes to physical memory sizes, this size remains constant after booting a guest OS.

Page Replacement Issues

+ The standard approach used by earlier virtual machine systems is to introduce another level of paging, moving some VM "physical" pages to a swap area on disk.
+ Unfortunately, an extra level of paging requires a meta-level page replacement policy: the virtual machine system must choose not only the VM from which to revoke memory, but also which of its particular pages to reclaim.
+ a meta-level page replacement policy must make relatively uninformed resource management decisions, why? Because it has no knowledge of the actual memory usage patterns of the guest operating systems. Because of this limited information, a meta-level page replacement policy must make relatively uninformed resource management decisions. These decisions may not be optimal for every individual page or process, but they are designed to provide the best overall performance for the system as a whole.
+ What is double paging problem? – Double paging is a problem that occurs when a virtual machine system uses a meta-level page replacement policy to reclaim memory from a guest operating system. The problem occurs when the guest OS is using a page that is not currently mapped into the guest OS's page tables. In this case, the guest OS will not be able to access the page, and the virtual machine system will not be able to reclaim it. This situation is called a "double page fault" because the guest OS will fault on the first page fault, and the virtual machine system will fault on the second page fault.

****Ballooning****

+ Ballooning is a technique that allows a guest OS to reclaim pages considered to be least useful by the guest OS. (solution to double paging problem)
+ coaxing - to persuade someone to do something by using gentle or friendly methods.
+ A small balloon module(LKM) is loaded into the guest OS as a pseudo-device driver or kernel service.(No external interface within the guest OS. communicate with the host OS through a private channel)
+ inflate when server is under memory pressure, deflate when server is not under memory pressure(deallocate previously-allocated pages)
+ The balloon driver communicates the physical page number for each allocated page to ESX Server, which may then reclaim the corresponding machine page
+ When a guest PPN is ballooned, the system annotates its pmap entry and deallocates the associated MPN.

+ Hot-pluggable memory cards would enable an additional form of coarse-grained ballooning. What is hot-pluggable memory cards? – Hot-pluggable memory cards are memory modules that can be added or removed from a computer system while the system is running. Hot-pluggable memory cards are also known as hot-swap memory cards.
+ What is cache coloring? – cache coloring (also known as page coloring) is the process of attempting to allocate free pages that are contiguous from the CPU cache's point of view, in order to maximize the total number of pages cached by the processor.
+ synthetic benchmark - a benchmark that is designed to test a specific aspect of a computer system, rather than a real-world application.
+ Disadvantages of ballooning - upper bounds on reasonable balloon sizes may be imposed by various guest OS limitations. - The balloon driver may be uninstalled, disabled explicitly, unavailable while a guest OS is booting, or temporarily unable to reclaim memory quickly enough to satisfy current system demands.

Demand Paging

+ When ballooning is not possible or insufficient, the system falls back to a paging mechanism. Memory is reclaimed by paging out to an ESX Server swap area on disk, without any guest involvement.
+ A randomized page replacement policy is used to prevent the types of pathological interference with native guest OS memory management algorithms

Sharing Memory

+ higher levels of overcommitment can be achieved by sharing identical pages between virtual machines efficiently.
+ guest OSs can share common application code and data, reducing the total amount of memory required to run multiple VMs.

Transparent Page Sharing

+ Disco paper introduces this term as a method for eliminating redundant copies of pages, such as code or read-only data, across virtual machines.
+ Writing to a shared page causes a fault that generates a private copy.

Content-based page sharing

+ Constraint - Modification to guest OS internals are not possible in ESX environment. - Changes to application programming interface(APIs) are not acceptable.
+ Basic Idea: Identify page copies by their contents, regardless of where, when and by whom they were generated. Page with identical contents can be shared.
+ Two key advantages - eliminates the need of modify, hook or even understand the guest OS code. - More opportunities for sharing, all potentially sharable pages can be identified by their contents.
+ Cost is high for comparison of all pages in the system. - $O(n\hat{2})$ time complexity - Hashing is used to identify pages with potentially-identical contents efficiently. - if hash values are identical, the full comparison is performed. - copy on write, reclaimed redundant pages.

Implementation

+ A single global hash table contains frames for all scanned pages, and chaining is used to handle collisions.
+ A shared frame consists of a hash value, the machine page number (MPN) for the shared page, a reference count, and a link for chaining.
+ The current ESX Server page sharing implementation scans guest pages randomly.
+ Configuration options control maximum per-VM and system-wide page scanning rates. (For zero CPU overhead)
+ What is zero pages? – Zero page is a page filled with zeros. You can make a mapping to this page and get wide zeroed virtual region.
+ Page sharing does improve memory locality, and may therefore increase hit rates in physically-indexed caches.
+ These experiments demonstrate that ESX Server is able to exploit sharing opportunities effectively.

Shares vs. Working Sets

+ What is working set in VM? – a virtual machine's working set size is defined as the amount of guest physical memory that is actively being used.

+ ESX Server employs a new allocation algorithm that is able to achieve efficient memory utilization while maintaining memory performance isolation guarantees.

Share Based Allocation

+ Shares are alternatively referred to as tickets or weights in the literature. The term clients is used to abstractly refer to entities such as threads, processes, VMs, users, or groups.
+ A client is entitled to consume resources proportional to its share allocation; it is guaranteed a minimum resource fraction equal to its fraction of the total shares in the system.
+ The shares-per-page ratio can be interpreted as a price; revocation reallocates memory away from clients paying a lower price to those willing to pay a higher price.

Reclaiming Idle Memory

+ A significant limitation of pure proportional-share algorithms is that they do not incorporate any information about active memory usage or working sets.
+ active client with fewer share – suffer from memory pressure while idle client with more share can hoard memory unproductively.
+ ESX Server resolves this problem by introducing an ”idle memory tax”.
+ The basic idea is to charge a client more for an idle page than for one it is actively using. When memory is scarce, pages will be reclaimed preferentially from clients that are not actively using their full allocations.
rho = Share(S) / (Pages(P) (f + k(1-f)) f = fraction of pages in use k = 1 / (1 - tou)

Measuring Idle Memory

+ For the idle memory tax to be effective, the server needs an efficient mechanism to estimate the fraction of memory in active use by each virtual machine.
+ Guest OS monitoring typically relies on access bits associated with page table entries, which are bypassed by DMA for device I/O.
+ ESX Server uses a statistical sampling approach to obtain aggregate VM working set estimates directly, without any guest involvement.
+ minor page faults? – A minor page fault occurs when a page is not in memory, but is present in the swap space (or some other paging file on disk). The page is brought into memory and the process continues as if nothing had happened.
+ major page faults? – A major page fault occurs when a page is not in memory and is not present in the swap space. In this case, the operating system must load the page from disk into memory. This is called a demand-paged page fault.

Allocation Policies
Parameters

+ System administrators use three basic parameters to control the allocation of memory to each VM: a min size, a max size, and memory shares. - min size: the minimum amount of memory that a VM is guaranteed to have available. even when memory is overcommitted. - max size: the maximum amount of memory that a VM can consume. Unless memory is overcommitted, VMs will be allocated their max size. - shares: Memory shares entitle a VM to a fraction of physical memory, based on a proportional-share allocation policy.

Admission Control

+ An admission control policy ensures that sufficient unreserved memory and server swap space is available before a VM is allowed to power on.
+ Machine memory must be reserved for the guaranteed min size, as well as additional overhead memory required for virtualization, for a total of min + overhead.
+ Disk swap space must be reserved for the remaining VM memory; i.e. max min.
+ Memory reservations are used for admission control, actual memory allocations vary dynamically, and unused reservations are not wasted.

Dynamic Reallocation

+ Most operating systems attempt to maintain a minimum amount of free memory.

+ ESX Server employs a similar approach, but uses four thresholds to reflect different reclamation states: high, soft, hard, and low, which default to 6%, 4%, 2%, and 1% of system memory, respectively.

I/O page Remmaping

+ ESX Server maintains statistics to track "hot" pages in high memory that are involved in repeated I/O operations. For example, a software cache of physical-to-machine page mappings (PPN-to-MPN) associated with network transmits is augmented to count the num- ber of times each page has been copied.
+ When the count exceeds a specified threshold, the page is transparently remapped into low memory.
+ This scheme has proved very effective with guest operating systems that use a limited number of pages as network buffers. For some network-intensive workloads, the number of pages copied is reduced by several orders of magnitude

§7. Live Migration of Virtual Machines(Lecture #12) 10/02/2023

Abstract

+ Live migration of virtual machines is a key feature of cloud computing. It allows a clean separation between hardware and software, and facilitates fault management, load balancing, and low-level system maintenance.
+ OS migration built on top of the Xen VMM.
+ Interactive Loads means that the VM is running a GUI, and the user is interacting with it. In this case, the VM is migrated by suspending it, copying its memory state to the destination, and resuming it there. – Examples of interactive workloads in cloud computing could include web applications that require real-time responses to user requests, video conferencing applications that require low-latency communication between multiple users, or online gaming applications that require low latency and high processing power.
+ Writable working sets: written often and so should best be transferred via stop-and- copy – we dub this set of pages the writable working set (WWS) of the operating system by obvious extension of the original working set.

Introduction

+ paravirtualization(i.e Xen) providing better use of physical resources, and better isolation between virtual machines than full virtualization.
+ VM migration is easier as compared to process level migration.
+ In particular the narrow interface between a virtualized OS and the virtual machine monitor (VMM) makes it easy avoid the problem of 'residual dependencies' in which the original host machine must remain available and network-accessible in order to service – residual dependencies 1. shared memory, wait() 2. OS state, tcp control blocks, fs state 3. isolated work
+ Metrics for evaluating migration - Down time - Migration time - CPU overhead (source and destination) - Network IO usage/requests
+ Load balancing
+ Maintenance of the physical machine
+ *PRE COPY* approach - Pages of memory are iteratively copied from the source machine to the destination host, all without ever stopping the execution of the virtual machine being migrated. - Page-level protection hardware is used to ensure a consistent snapshot is transferred, and a rate-adaptive algorithm is used to control the impact of migration traffic on running services.
+ The final phase pauses the virtual machine, copies any remaining pages to the destination, and resumes execution there.
+ eschews: abstain, delibrately avoid
+ Eschews "pull" approach because - relatively poor performance - requires the source machine to remain available and network-accessible in order to service requests from the destination machine for residual dependencies.

Related Work

+ Zaps - Pods : Suspend - copied - resume *Design*
+ Providing live migration of VMs in a clustered server environment leads us to focus on the physical resources used in such environments: specifically on memory, network and disk.

Migrating Memory

+ When a VM is running a live service it is important that this transfer occurs in a manner that balances the requirements of - minimizing both downtime(service interruption - visible to clients) and - total migration time
+ Migrating memory divided into three phases: - Push phase source vm run as normal. Certain pages are pushed across network to new destination vm. To ensure consistency, modified pages are resent. - Stop and copy phase source vm is paused and all remaining pages are copied to destination vm. vm is resumed on destination. - Pull phase The new VM executes and, if it accesses a page that has not yet been copied, this page is faulted in ("pulled") across the network from the source VM.
+ Most practical solution use one or two of these phases.
+ Pure stop-and-copy downtime and migration both are proportional to amount of physical memory.
+ Post Copy (pure demand-migration): short stop-and-copy phase, followed by a long pull phase. - shorter downtime but much longer migration time.
+ PRE COPY: - Iterative pre copy phase followed by a short stop-and-copy phase. - By 'iterative' we mean that pre-copying occurs in rounds, in which the pages to be transferred during round n are those that are modified during round n - 1 (all pages are transferred in the first round). - Bound the number of rounds of pre-copying, based on analysis of the writable working set (WWS) behavior of typical server workloads.

Local Resources

+ Memory can be directly copied to the destination host.
+ Key Challenges: - Device connected with the source physical machine, connections to local devices such as disks and network interfaces demand additional consideration.
+ Two Key problems - Network resources We want a migrated OS to maintain all open network connections without relying on "forwarding mechanisms on the original host". A migrating VM will include all protocol state (e.g. TCP PCBs), and will carry its IP address with it. The network interfaces of the source and destination machines typically exist on a single switched LAN. Our solution for managing migration with respect to network in this environment is to generate an unsolicited "ARP reply"(proactively announce its own IP and MAC address, without waiting for an ARP request.) from the "migrated host"(I think here migrated host refers to destination), advertising that the IP has moved to a new location. small number of in-flight packets may lost. Some routers are configured not to accept broadcast ARP replies (in order to prevent IP spoofing), so an unsolicited ARP may not work in all scenarios. If the operating system is aware of the migration, it can opt to send directed replies only to interfaces listed in its own ARP cache, to remove the need for a broadcast. Alternatively, on a switched network, the migrating OS can keep its original Ethernet MAC address, relying on the network switch to detect its move to a new port - local devices Most modern data centers consolidate their storage requirements using a network-attached storage (NAS) device, in preference to using local disks in individual servers. The problem of migrating local-disk storage is not addressed in this paper

Design Overview

+ View the migration process as a transactional interaction between the two hosts involved: - Stage 0: Pre-Migration Active VM on host A - Stage 1: Reservation Initialize a container on the target host B - Stage 2: Iterative Pre-Copy (Overhead due to copying) Enable shadow Paging Enable page-level protection Copy dirty pages in successive rounds - Stage 3: Stop-and-Copy Suspend VM on host A Generate ARP to redirect traffic to Host B Synchronize all remaining VM state to Host B - Stage 4: Commitment (stage 3 + stage 4: downtime) VM state on host A is released - Stage 5: Activation VM starts on Host B Connects to local devices Resumes normal operation

Writable Working Set

+ Pre-Copy Migration: The drawback is the wasted overhead of transferring memory pages that are subsequently modified, and hence must be transferred again
+ Clearly if the VM being migrated never modifies memory, a single pre-copy of each memory page will suffice to transfer a consistent image to the destination.
+ However, should the VM continuously "dirty pages faster than the rate of copying", then all pre-copy work will be in vain and one should immediately stop and copy.

Measuring WWS

+ All shadow page table are write protected.
+ To trace the writable working set behaviour of a number of representative workloads we used Xen's shadow page tables to track dirtying statistics on all pages used by a particular executing OS.

+ Performing a migration of an operating system will give different results depending on the workload and the precise moment at which migration begins

Implementation Issues

+ Xen has a special management virtual machine used for the administration and control of the machine.
+ Two different methods: - Managed Migration The management VM is responsible for the migration process, and the VM being migrated is unaware of the migration. - Self-Migration Implemented almost entirely within the migratee OS with only a small stub required on the destination machine.

Managed Migration

+ Managed migration is performed by migration daemons running in the management VMs of the source and destination hosts.
+ These are responsible for creating a new VM on the destination machine, and coordinating transfer of live system state over the network.
+ Translation is very simple for dirty logging: - all page-table entries (PTEs) are initially read-only mappings in the shadow tables, regardless of what is permitted by the guest tables. - If the guest tries to modify a page of memory, the resulting page fault is trapped by Xen. If write access is permitted by the relevant guest PTE then this permission is extended to the shadow PTE. At the same time, we set the appropriate bit in the VM's dirty bitmap.

Self-Migration

+ self-migration places the majority of the implementation within the OS being migrated.
+ In this design no modifications are required either to Xen or to the management software running on the source machine,
+ although a migration stub must run on the destination machine to listen for incoming migration requests.
+ Create an appropriate empty VM, and receive the migrated system state.
+ The major implementation difficulty of this scheme is to transfer a "consistent OS checkpoint".
+ In contrast with a managed migration, where we simply suspend the migratee to obtain a consistent checkpoint.
+ self migration is far harder because the OS must continue to run in order to transfer its final state.
+ We solve this difficulty by logically checkpointing the OS on entry to a final two-stage stop-and-copy phase. - The first stage disables all OS activity except for migration and then peforms a final scan of the dirty bitmap, clearing the appropriate bit as each page is transferred. - Any pages that are dirtied during the final scan, and that are still marked as dirty in the bitmap, are copied to a shadow buffer. - The second and final stage then transfers the contents of the shadow buffer — page updates are ignored during this transfer.

Dynamic Rate-Limiting

+ Extended downtime because the hottest pages in the writable working set are not amenable to pre-copy migration.
+ The downtime can be reduced by increasing the bandwidth limit, albeit at the cost of additional network contention.
+ One solution: dynamically adapt the bandwidth limit during each pre-copying round.

Rapid Page Dirtying

+ Author observed that page dirtying is often physically clustered — if a page is dirtied then it is disproportionally likely that a close neighbour will be dirtied soon after.

Paravirtualized Optimizations

+ Stunning Rogue Processes.
+ Freeing Page Cache Pages - When informed a migration is to begin, the OS can simply return some or all of these pages to Xen in the same way it would when using the ballooning mechanism described in. - However should the contents of these pages be needed again, they will need to be faulted back in from disk, incurring a greater overall cost.

Evaluation

+ Simple Web Server - small WWS - relatively easy case for live migration.

Conclusion

+ By integrating live OS migration into the Xen virtual machine monitor authors enable rapid movement of interactive workloads within clusters and data centers.
+ Dynamic network-bandwidth adaptation allows migration to proceed with minimal impact on running services, while reducing total downtime to below discernable thresholds.
+ SPECweb99 can be migrated with just 210ms downtime, while a Quake3 game server is migrated with an imperceptible 60ms outage.

Lecture #13 10/02/2023 sandpiper