# Virtualization Support

Advanced Operating Systems and Virtualization

Alessandro Pellegrini

A.Y. 2017/2018

# System Virtualization

- Virtualization allows to show resources different from the physical ones
- More operating systems can be run on the same hardware
- A Virtual Machine is a mixure of software- and hardware-based facilities
- The software component is the Hypervisor or VMM (Virtual Machine Monitor).

- Advantages:
  - Isolation of different execution environments (on the same hardware)
  - Reduction of hardware and administration costs

# Hypervisor

- *Host system*: the real system where (software implemented) virtual machines run
- *Guest system*: the system that runs on top of a (software implemented) virtual machine

- Hypervisor:
  - It manages hardware resources available to the *host system*
  - It makes virtualized resources available to the guest system in a correct and secure way

  - *Native Hypervisor:* runs with full capabilities on the native host hardware. It resembles a lightweight virtualization kernel operating on top of the harware.
  - *Hosted Hypervisor:* it runs as an applicaiton, which accesses the actual host services via system calls

# Software-based Virtualization

- Instructions are executed by the native physical CPU in the host platform
- We need to emulate a subset of the instruction set
- No particular  hardware component playes a role in virtualiztion (as instead for the case of Intel VT-x o AMD-V).

- *The main issue:*
  - What if RING 0 is required for the guest system tasks?
  - Risk to bypass the VMM resource management policy in case of actual RING 0 access

- *The solution:* ring deprivileging.

# Ring Deprivileging

- A technique to let the guest kernel run at privilege level that simulates 0

- Two main strategies:
  1. 0 / 1 / 3 Model:
     - VMM runs at ring 0.
     - Kernel guest runs at ring 1 (which is typically not used by native kernels)
     - Applications still run at ring 3.
     - This is the most used approach.
  2. 0 / 3 / 3 Model :
     - VMM runs at ring 0.
     - Kernel guest and applications run at ring 3.
     - Too close to emulation, too high costs.

# 0/1/3 Model

- The application layer (running at ring 3) cannot damage the guest operating system state (which runs at ring 1).
- The guest system cannot access to the hadware priviledged facilities bypassing the VMM, so we still guarantee the isolation of guest systems' execution
- Any exception must be trapped by the VMM (at ring 0) and must be properly handled (e.g. by reflecting it into ring 1 tasks)

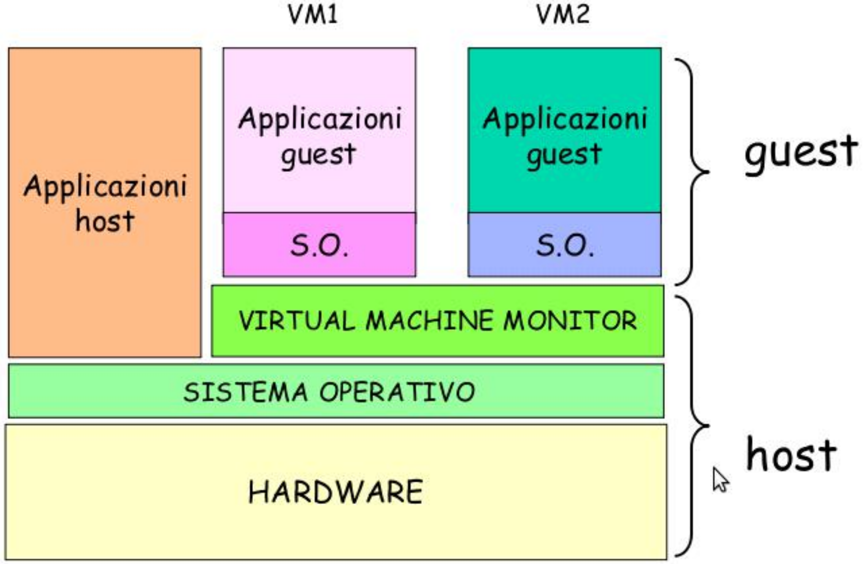
- Issues to cope with:
    - Ring aliasing
    - Virtualization of the interrupts
    - Frequent access to privileged resources

# Ring Aliasing

- The kernel is designed to run at ring 0, while it is actually being run at ring 1 for guest systems
- Privileged instructions generate an exception is not run at CPL 0:
  - hlt
  - lidt
  - lgdt
  - invd
  - mov %crx
- *I/0 sensistive instructions*: they generate a trap if executed when CPL > IOPL (I/O Privilege Level). Classical examples are:
  - cli
  - sti
- The generated trap (*general protection fault*) must be handled by VMM,  so as to finally determine how to handle it (emulation vs interpretation)

# The VirtualBox Example

- Based on hosted hypervisor with ad-hoc kernel facilities, via classical special devices.




- Pure software virtualization is supported for x86
  - Fast Binary Translation (code patching): the kernel code is analysed and modified before being executed so as to replace privileged instructions with semantically equivalent blocks of code
- Based on the 0/1/3 model

# Execution Modes and Context

- Guest context (GC): execution context for the *guest system*. It is bsed on two modes:
  - Raw mode: native guest code runs at ring level 3 or ring level 1
  - Hypervisor: VirtualBox code is run at the maximum privilege level (ring 0)

- Host context (HC): execution context for userspace portions of VirtualBox (ring 3):
  - The running thread implementing the VM lives in this context upon a mode change
  - REM mode: execution mode for emulating critical/privileged instructions

# Virtual Box GDT

- Introduction of gate descriptors for kernel code/data segments with DPL=1. These segments are accessible with CPL=1
- New TSSD pointing to the TSS wrapper which keeps info on stack positioning at ring 1 (ss1,esp1) and ring 0 (ss0,esp0).
- 2 new segments for the Hypervisor are addedd with DPL=0

| DESCRIPTION | OFFSET | DPL | BASE |
|---|---|---|---|
| Entry 0 | $(0000)_H$ | - | null |
| ... | ... | ... | ... |
| **KERNEL CODE SEGMENT** | $(0060)_H$ | 1 | |
| **KERNEL DATA SEGMENT** | $(0068)_H$ | 1 | |
| ... | ... | ... | ... |
| **VIRTUALBOX TSSD** | $(FFE0)_H$ | 0 | |
| ... | ... | ... | ... |
| **HYPERVISOR DATA SEGMENT** | $(FFF0)_H$ | 0 | |
| **HYPERVISOR CODE SEGMENT** | $(FFF8)_H$ | 0 | |

ORIGINAL TSS

| | |
|---|---|
| ... | ... |
| esp0 | ... |
| ss0 | $(0068)_H$ |
| esp1 | unused |
| ss1 | unused |
| ... | |

VBOXTSS

| | |
|---|---|
| ... | ... |
| esp0 | $(FE557000)_H$ |
| Ss0 | **$(FFF0)_H$** |
| esp1 | $(F70D3FF8)_H$ |
| ss1 | $(0069)_H$ |
| ... | |

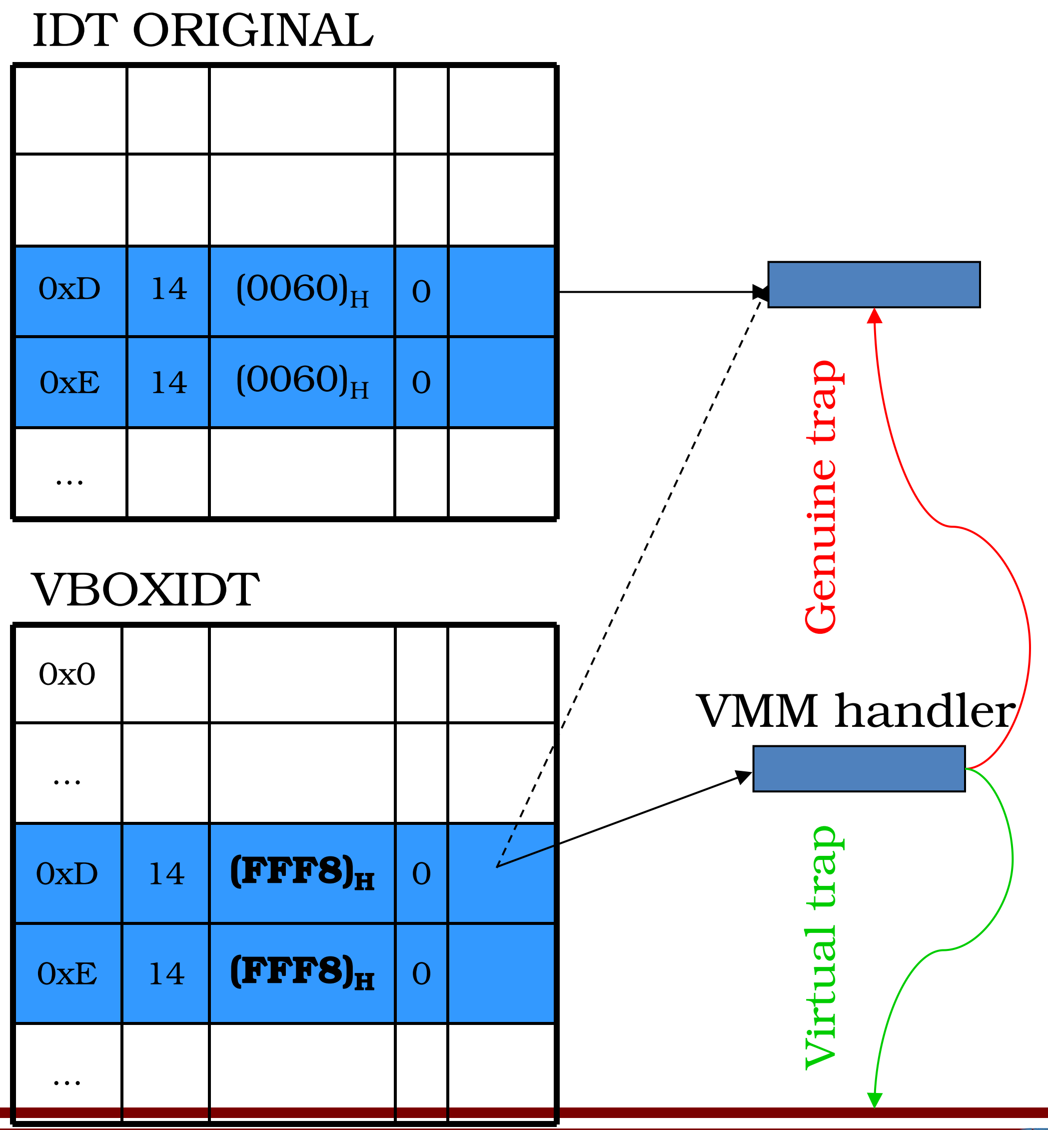
ss1=ss0 | 1

CPL = Current Privilege Level

DPL = Descriptor Privilege Level

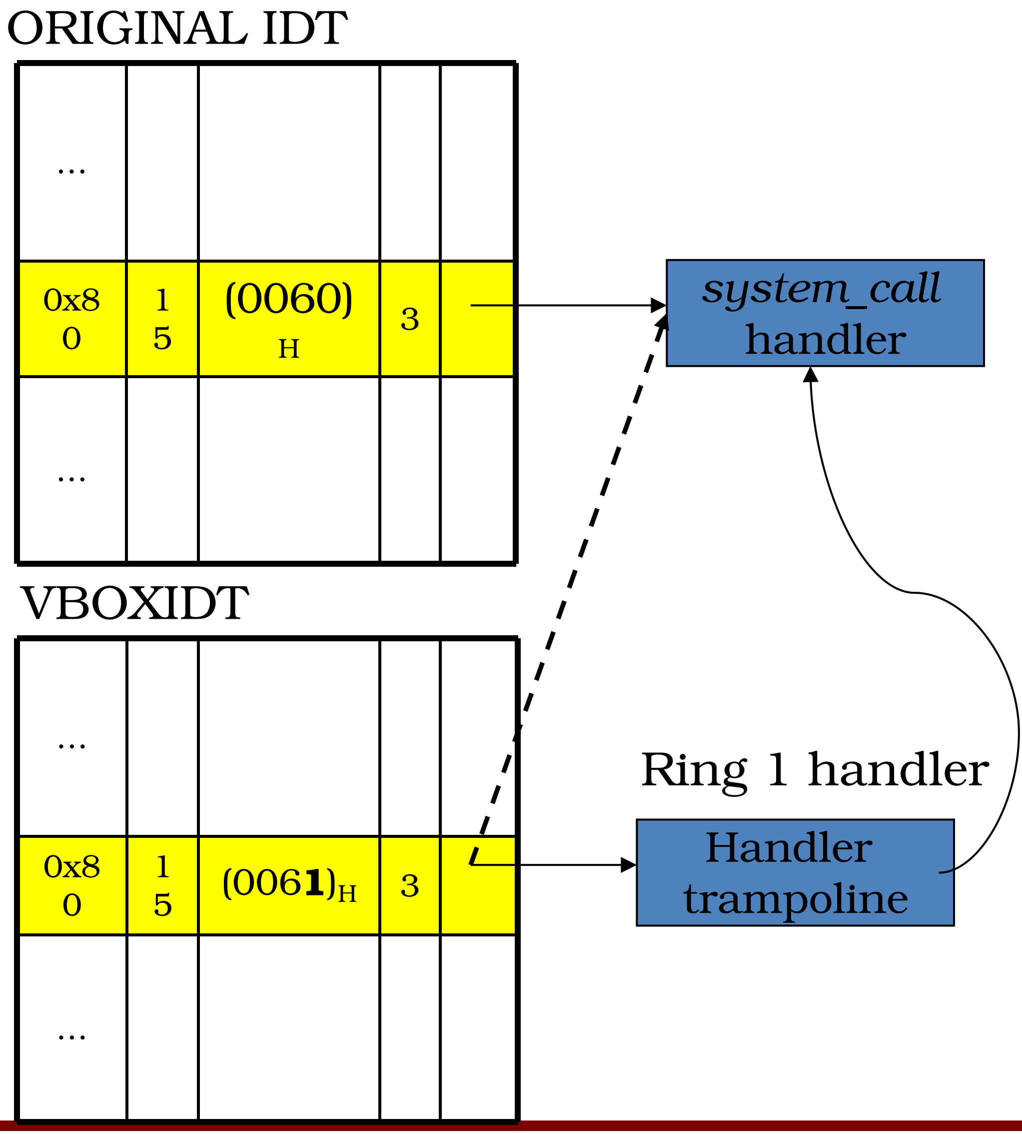
# VBOXIDT: interrupt gate

- Interrupt must be managed by the VMM.
- To this end, a wrapper for the IDT is generated
- Proper handlers are instantiated, which get executed by the Hypervisor upon traps. VMM can take control thanks to the ad-hoc segment selector (at the GDT offset for the *hypervisor code segment*).
- In case of a "genuine" trap, the control goes to the native kernel, otherwise the virtual handler is executed



# VBOXIDT: gate 0x80

- INT 0x80 has an ad-hoc management
- The syscall gate is modified so as to provide a segment selector with RPL = 1
- It indicates the GDT offset for the code segment (at ring 1).
- Hence calling a system call does not require interaction with the Hypervisor
- The trampoline handler is then used to launch the actual syscall handler



# Access to raw mode

- This is used for privileged instructions
  - LIDT -> idtr points to VBOXIDT
  - LGDT -> gdtr poiunts to VBOXGDT
  - LTR -> trpoints to VBOXTSS

- The guest system can then take back control by returning from the trap (iret), with the following registers saved on the stack
  - SS
  - ESP
  - EFLAGS
  - CS
  - EIP

# Privileged instructions: patching

- Privileged instructions may hamper performane given that the Hypervisor needs to take back control for handling any of them
- A way to cope with this is patching of these instructions

- An example: the cliinstruction
- Trap if CPL<=IOPL → VMM sets IOPL=0 upon entering raw mode
- Problem: if IF=0, then VMM cannot handle interrupts anymore.

- The solution: the code block cli…sti is replaced with a functionally-equivalent one
  - Interrupts are disabled only for the guest system
  - The Hypevisor will take care of finally delivering it.

# REM mode

- It does not use runtime patching due to efficiency issues
- Actually executed in host context at ring 3.
- It relies on QEMU.
- Emulation process can be slow, since we need to keep track of processor state changes to be restored upon reentering raw mode
- Typically, at each emulation step, it is checked whether native code execution can be restored

# Kernel Samepage Merging

- COW is used by the kernel to share physical frames with different virtual mappings
- If the kernel has no knowledge on the usage of memory, a similar behaviour is difficult to put in place
- KSM exposes the `/dev/ksm` pseudofile
- By means of `ioctl()` calls, programs can register portions of their address spaces
- An additional `ioctl()` call enables the page sharing mechanism, and the kernel starts looking for pages to share

# Kernel Samepage Merging

- The KSM driver (in a kernel thread) picks one registered region and starts scanning for it
  - A SHA1 hash is used to compare frames
  - If a similarity is found, all processes "sharing" the page will point to the same frame (in COW mode)
- A host running several guest Windows machines can overcommit its memory 300% without affecting performance
  - Windows zeroes all free'd memory