Virtualization, Security and RTOS

Rajesh Gupta
Computer Science and Engineering
University of California, San Diego.
Overview

• What is virtualization? Types of virtualization and VMs

• Virtualization and RTOS

• Virtualization techniques

• Virtualization and Security: Trusted Virtual Machines

• Mobile Security: Android
Virtualization

• Briefly, a virtual machine is an abstract system composed of virtual processor(s), virtual devices.
  – Used mainly for portability and/or protection purposes
    • Virtualized processor, virtualized memory, virtualized IO
  – Distinction often made in the computer architecture community between process and system virtualization
• Virtualization implements an isomorphism between a virtual guest and a real host
  – Maps a guest state to a host state and corresponding sequence of operations
  – Similar to an abstraction, though the mapping does not necessarily hide details.
Types of VMs

• Process VM
  – Virtualization software consists of a runtime system that enables an application (guest) to run on top of a host platform (host OS, host CPU)

• System VM
  – Provides a complete system environment to support a guest OS (and its multiple processes)
  – The guest OS has access to hardware resources
  – The virtualization software is called the VMM
VM Uses

- Multiple, replicated VMs can be implemented on a platform
  - A large MP server can implement multiple virtual servers with security and protection benefits
    * And the ability to balance resource usage across VMs
  - In a degenerate sense, one can think of the OS as VM that provides a dedicated machine to each user process

- Provide cross-platform software compatibility
  - Via machine emulation either for the OS or for an application

- On-the-fly optimization of program binaries

- VMs can be used not only for emulating machine but even creating non-existent machines
  - Used by language developers to develop portable ‘binaries’ (or the compiled/machine code). Java VM
Virtualization and RTOS

• Virtualization is a general concept that can be applied in general to the ability to export capabilities across hw/sw platforms

• In the OS context, the capabilities we are interested in are:
  – Multiprogramming
  – System services

• In the RTOS context, the capabilities can be
  – Contextual awareness (location)
    • Contextual information is made available to (remote) applications
  – Security (trust)
    • Security guarantees are made available to (remote) applications
Virtual Machine Operating Systems

• Very rich background of work dating back to IBM 360/367
  – CP-40, later CP-67 system that provided virtual IBM 360/65 on IBM 360/367 machine including IO devices
    • But no MMU virtualization which was added later
    • Still in use today, z/VM runs on zSeries mainframes and supports multiple OS images
  
• Later, ISA support for virtualization was added in mainstream processor architectures
  – Mainly through privilege levels needed to support virtualization

• Currently, a hot bed of commercialization activity
Virtualization Techniques

• VM can be built using virtualization techniques
  – Interpretation, JIT compilation, HW-assisted instruction emulation, etc

• Virtualization by Interpretation
  – Achieve emulation using an interpreter program
  – The interpreter executes guest/source ISA by emulating individual instructions on a host (native) machine
  – Can be a slow process: emulating target ISA at a very low level (fetch, decode, execute etc for each target instr)
  – Low startup overhead, but high overhead at runtime
Virtualization Techniques

• Dynamic Binary Translation
  – Usually better performance by converting blocks of source instructions into target instructions that perform the same functions
  – High startup overhead, but benefits from repeated executions (caches)

• Combine the two approaches: staged emulation via profiling
  – Interpret instruction blocks, binary translate the most frequent ones
Virtualization, State and Memory

• A processor instruction executes in the context of processor state
  – Distinguished by processor mode (privilege level)

• There are two simplified views of the processor that an application sees
  – User-mode view: portion of the processor state that is accessible through unprivileged instructions
  – Privileged-mode view: accessible through privileged instructions

• True processor state is a super set of both views
Virtualization, State and Memory

• The issue becomes more complicated when processor supports multiple levels of privileged modes
  – Nested set through privileged rings
    • Outer view does not have visibility of processor state visible from an inner view
  – Collection of independent views: e.g., some registers can be common to two independent views but some may not

• So if a hardware resource is updated in one view that is independent from another state then that resource is viewed differently in different states
  – E.g., ADD R13, R13, #1
Structure of VM OS

- Two broad components
  - Virtual Machine Monitor (VMM)
  - Guest OS

- VMM is also known as the Control Program (CP) that
  - Provides independent views of the processor state to the applications (OS, programs)
  - Provides mechanisms to switch the physical processor among multiple applications (i.e., implements multiprogramming)
    - Switch not only among different programs but also among different processor views

- Instead of traditional user operations and system calls to OS services, a VMM supports virtual processors
  - Virtual processors can run in user or privileged mode
Virtualization by Instruction Emulation

• VMM maintains a VM Control Block (VMCB) for each VM

• VMCB holds the full processor state, both user and privileged of the virtual processor

• On VM execution, the VMM transfers part of the VMCB into the physical processor state
  – The VM is always executed in the user mode regardless of the virtual processor mode

• With same source and target ISA, there may still be overheads caused by the VMM for
  – Updating physical processor and/or VMCB in case the processor views are different (user versus privileged)
  – Physical processor may trap to the VMM which must then emulate the trap to update VMCB accordingly

• In some cases, this may be problematic if VMM can not detect an unprivileged source instruction that gives different results based on privileged physical state information.
Emulation of privileged instructions

• Since VMs are always executed in the user mode
  – The target execution may be required to produce effect on the virtual privileged states of the processor
  – This is done by the VMM which updates the physical processor state and VMCB accordingly

• Also, when a privileged instruction is attempted in the user mode (of the physical processor), the physical processor takes a ‘privileged instruction trap’

• VMM must be able to receive and handle traps on behalf of a VM
  – E.g., request to emulate a privileged instruction
  – Actual privileged operation is done by the software in the virtual machine running in the privileged mode

• VMM may also need to perform trap handling in cases traps are triggered by other reasons
Trap Handling for Privileged Instructions

• VMM Emulation of ‘legal’ privileged instructions executed by the VM
  – Pure artifact due to the fact that VMs runs in user mode on the physical processor

• Freeze the VM so as to free the physical processor for VMM use
  – Save into the VMCB all registers in the view corresponding to the current virtual processor mode

• Locate and decode the instruction to be emulated in the virtual processor instruction stream
  – Could be expensive operation depending upon what a trap does to the processor state

• Switch the physical processor into the privileged mode for instruction emulation
  – There may be many such privileged modes, so the mapping from virtual processor privilege mode to physical processor privilege may not be the same

• Emulate the instruction using the VMCB as the reference machine state for the emulation
  – The outcome of emulation modifies the VMCB, may even change the privilege state of the virtual processor

• Update the virtual PC in the VMCB to the next instruction in the virtual processor

• Restore the virtual processor state from the updated VMCB and return from the trap.
Exception and Interrupt Handling

• Synchronous exceptions are handled by the VMM
  – And not the guest OS
• However, there may be exceptions that are handled by VMM but interpreted differently by the VM
  – Exceptions generated by a system call instruction

• Asynchronous exceptions
  – Interrupts for the VMM itself:
    • e.g., VMM process scheduler, VMM console actions
  – Interrupts for a single VM:
    • e.g., a disk interrupt for a physical disk assigned to a VM
  – Interrupts synthesized by the VMM and destined to a VM:
    • e.g., a disk interrupt for a virtual disk emulated by the VMM
General Approach

• Save the status of the current VM into corresponding VMCB
• Switch processor to the VMM context and stack
• Select the most privileged mode
• Determine the type of the interrupt request and to which VM it should be dispatched
• Emulate the interrupt processing normally performed by the physical processor in the corresponding VMCB
  – What happens if interrupt is received when VMM was in active execution (e.g., in emulating an instruction on behalf of a VM)?
  – Typically, interrupt handling is deferred to the end of the current emulation
• Return either to the VMM or the VM code that was in progress when interrupt arrived.
VMM

• In general, trap handling is the most elaborate aspect of VMM design
  – Since the VMM may be expected to handle any exception that the processor takes
• VMM also provides means to schedule among multiple VMs
• VMM Process Scheduler
  – Generally tied to the handling of privileged instructions since it is a convenient point for VM scheduling
  – OTOH, interrupt handling must disable process scheduling to ensure interrupt handling is directed to appropriate VMs
• Interrupt handling is deferred to the end of current VMM emulation path
  – Ensures VMM context does not need to be saved
  – VMM is not reentrant since VMM/VMM switch never occurs.
• This affects processor allocation latency which is sum of
  – Longest emulation path in the VMM
  – Longest sequence of instructions in the VMM to be executed with interrupts disabled due to synchronization constraints
  – The scheduling time
  – The VM context save and restore time
Virtualization and Security

- Security in current computing platforms follows the model proposed by TCG
  - Use a Trusted Platform Module (TPM) that provides secured storage and cryptographic functions to the OS
  - A hardware based Root of Trust: TPM enables attestation by signing cryptographic hashes of the platform configuration and software components
    - That the HW, SW are genuine
  - Build a chain of trust by signing keys and verifying certificates by each higher layer
- Enter virtualization
  - Virtualization can make TPM functions available to multiple VMs
  - However, there is a need to distinguish among virtual and physical TPM
    - That runs directly against the VM design philosophy
- How does a trusted machine that the chain of trust is established and maintained in a VM execution environment?