I/O virtualization

Jiang, Yunhong <yunhong.jiang@intel.com>
Yang, Xiaowei <xiaowei.yang@intel.com>
Legal Disclaimer

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL’S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL® PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. INTEL PRODUCTS ARE NOT INTENDED FOR USE IN MEDICAL, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS.

Intel may make changes to specifications and product descriptions at any time, without notice. All products, computer systems, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.

Intel and the Intel logo are trademarks of Intel Corporation in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2009 Intel Corporation.
Agenda

- Overview
- Software Emulation
- Para-virtualization
- Direct I/O
- SR-IOV
A Retrospect on Physical Platform

![Diagram of physical platform components]

- Processor
- Memory
- SATA Disk
- USB host controller
- USB Device
- Display
- NIC
- Legacy Device
- USB host controller
- ICH 10
- Intel QuickPath Interconnect
- PCIe
- LPC
Device Semantic

- **Software communicate with device**
  - Port I/O
  - MMIO

- **Device transfer data to and from system memory**
  - DMA access

- **Events notification from device**
  - Interrupt

- **Device discovery and configuration**
  - Configuration space access in PCI device
I/O Virtualization Overview

• Present virtual I/O to VM
  ▪ Isolation
  ▪ Performance
  ▪ Scalability
  ▪ Reliability

• Approaches
  ▪ Software Emulation
  ▪ Para-virtualization
  ▪ Direct I/O
Agenda

- Overview
- Software Emulation
- Para-virtualization
- Direct I/O
- SR-IOV
Overview

- **Emulate existing hardware completely in software components (aka. Device Model)**
  - Must maintain same SW interface as native device
    - E.g. I/O, MMIO, Interrupt, DMA
  - But, could use arbitrary media to emulate the virtualized device
    - E.g. VMM can use a SATA disk (or a partition, a file) to emulate a virtualized IDE disk

- **Transparent to VM software stack**
I/O Access Emulation

- Device Model
  - State machine
  - Handlers
  - Driver
- Apps
  - Native Driver
  - VM Entry
  - VM Exit
- Device
- VMM
- HW
I/O Access Emulation (II)

- VMCS IO bitmap decides which IO port access will cause VM Exit
- VMM setup shadow page table/EPT table to trap MMIO access
  - In shadow model, corresponding shadow L1 entry will be set non-present
  - In EPT model, the EPT table will not setup the mapping for the MMIO range

MMIO trap in shadow model
DMA Emulation

Device Model

State machine

0x70, w, 1
...
...
0x90, w, 2

Handlers

Driver

Native Driver

VMM

Apps

DMA target address

HW

Device

2 1

0x0

0x1 0x2
Interrupt Emulation

Native Interrupt Injection

Virtual Interrupt Injection

Interrupt Emulation

CPU

IDTR

IDT

ICH

PIC

PIRQA

INTA#

Device

Virtual CPU

vIDTR

vIDT

Device Model

Device API Virtual PIC

VM Entry Interrupt Injection
Case Study – IDE sector read

- Driver allocate the PRDT
- Driver write register BMIDTPP to setup the PRDT address
  - VMM trap the write operation and notify DM (Device Model)
  - Device model map the guest memory pointed by PRDT
- Driver set the buffer address/length in the PRDT
- Driver write register to set the target IDE vector
  - VMM trap the write operation and notify DM
  - DM keep this information in state machine
- Driver write the BMICX command register to trigger a DMA read
  - VMM trap the write operation and notify DM
  - DM translate the IDE sector to offset in the image file
  - DM read corresponding data
  - DM copy the data to guest’s memory
- A interrupt is injected after DMA read
  - DM inject a virtual IDE interrupt to guest
- OS acknowledge the interrupt and consume the data in the buffer
PCI Device Discovery & Configuration in Native

• PCI device is presented as (BUS, Device, Function) in PCI hierarchy

• Software enumerate devices through configuration space
  ▪ Configuration space access return 0xFF for non-exist device
  ▪ Device ID/Vendor ID/Revision ID to identify devices

• Device’s Port IO/MMIO base address is configurable
  ▪ BAR (Base Address Register) in configuration space
PCI Device Discovery in Software Emulation

- **Device model emulates a virtual PCI hierarchy**
  - A host-PCI bridge is emulated
  - PCI device’s configuration space access is emulated, like
    - Device ID/Vendor ID for emulated device
    - Port IO/MMIO BAR
Device Emulation – Pros & Cons

• Pros
  ▪ Transparent to VM software stack
  ▪ Agnostic to physical device in the platform. Thus
    ▪ Legacy SW can still run, even after HW upgrade
    ▪ Smooth VM migration across different platforms
  ▪ Good physical device sharing

• Cons
  ▪ Un-optimum performance
  ▪ Cannot enjoy latest & greatest HW
    ▪ Lack of modern device emulation, since too complex
  ▪ Poor scalability
  ▪ Isolation and stability depends on implementation
Agenda

- Overview
- Software Emulation
- Para-virtualization
- Direct I/O
- SR-IOV
Overview

• **Goal -- Improve I/O performance through SW approach**
  - VMM presents a specialized virtual device to VM
    * A new & efficient interface between VMM & VM driver
      - Usually high-level abstraction
      - Requires a specialized driver in VM
  - Tactics to further reduce overhead caused by VMM
    * Batched I/O
    * Shared memory
Case Study – PV block device read

- FE (front-end) driver receives an OS standard read request
- FE grants the write permission of read buffer to the domain where BE (backend) driver is in
- FE produces a corresponding request (buffer’s index, offset, length) in shared data structure, then notifies BE
- Upon notification, BE consumes the requests by
  - mapping the remote domain’s buffer,
  - reassembling it in a OS standard format
  - delivering downwards
- When the request is complete, BE callback function assembles the response in the shared data structure and notifies FE
- FE removes the buffer’s granted permission and completes the read request finally
Para-virtualization – Pros & Cons

• **Pros**
  - Better performance
  - Agnostic to physical device in the platform.
    - Benefit smooth VM migration

• **Cons**
  - Need install specialized driver in VM
  - High CPU utilization for the I/O interface and memory copy
  - Not so good scalability because of CPU utilization
  - Isolation and stability depends on implementation
Agenda

• Overview
• Software Emulation
• Para-virtualization
• Direct I/O
• SR-IOV
Device I/O - Overview

- Assign a physical device to a VM directly
- VM access the device directly, w/ VMM intervention reduced to minimum
- Physical device access guest’s memory directly with help of Intel’s VT-d technology
- Interrupt will be injected to guest through hypervisor
- VMM emulates PCI configuration space (may through device model)
**Case Study – IDE sector read**

- **Driver allocate the PRDT**
- **Driver write register BMIDTPP to setup the PRDT address**
  - The access is passed-through to hardware
- **Driver set the buffer address/length in the PRDT**
- **Driver write register to set the target IDE vector**
  - The access is passed-through to hardware
- **Driver write the BMICX command register to trigger a DMA read**
  - The access is passed-through to hardware
  - IDE disk perform DMA access to the memory through VT-d
- **A interrupt is injected after DMA read**
  - DM inject a virtual IDE interrupt to guest
- **OS acknowledge the interrupt and consume the data in the buffer**
DMA Operation

• Driver in guest use guest physical address as DMA target

• Device issue DMA access using address setup by guest

• Also, driver in VM can set bogus address as DMA target
Intel VT-d Technology – DMA Remapping

• All DMA transaction is captured by chipset
• Intel VT-d technology provides DMA remapping, to translate ....
Intel VT-d Technology – DMA Remapping

Memory-resident Partitioning & Translation Structures
DMA Remapping – Translation Flow

- Context entry points to the translation structure
- Multiple-level page table is used for address translation
DMA Remapping – PCIe ATS

- PCIe ATS (Address Translation Services) specification enable I/O devices to participate in DMA remapping
  - Device can request a translation explicitly before DMA access
  - A special flag to indicate if DMA access’s target address is translated or not
  - VT-d will only translate DMA request that isn’t translated by device
I/O Access Pass-through

- **Guest may change device’s Port IO/MMIO base address**
  - Will not impact physical MMIO BAR through configuration space emulation
  - Guest has different port IO/MMIO base address with physical one

- **MMIO access pass-through**
  - For shadow mode, setup shadow page table according to physical address
  - For EPT, setup the EPT table for the translation

- **Port I/O access is pass-through only if guest doesn’t change Port IO BAR**
  - Otherwise need trap-and-emulation
Device Initial State

- Device should have clean state before assigned to guest
  - To avoid information leak when device detach
  - To avoid in-flight device action

- VMM need quiescent and reset a PCIe function before device assignment
Device Direct Assignment – Pros & Cons

- **Pros**
  - Near-to-native performance
  - Minimum VMM intervention, thus low CPU utilization
  - Good isolation

- **Cons**
  - Exclusive device access
  - PCI slots in system is limited. Thus not a very scalable solution.
Agenda

• I/O Virtualization Overview
• Device Emulation
• Device Para-virtualization
• Device Direct Assignment
• SR-IOV
SR-IOV - Overview

• A standard defined by PCI SIG, to provide “virtualization friendly” device
  ▪ Device can be shared by VMs, while still enjoying the benefits of device direct assignment
SR-IOV - Overview

• A PCI standard to provide "virtualization friendly" device
  ▪ Device can be shared by VMs, while still enjoying the benefits of device direct assignment

• Start with a single function device
  ▪ HW under the control of privileged software (VMM)
  ▪ Includes a SR-IOV capability
  ▪ Physical Function

• Replicate the resources needed by a VM
  ▪ MMIO for direct communication
  ▪ RID for tag DMA traffic
  ▪ Minimal configuration space
  ▪ Virtual Function (VF)
SR-IOV – Runtime Operation

- **Guest’s run-time operation to VF is same to Device Direct assignment**
  - Same MMIO pass-through mechanism
  - No Port IO supported in VF
  - DMA through DMA remapping
  - Interrupt emulation through hypervisor

- **The only difference is device configuration**
  - How to enable the VF
    - VF does not exists by default
  - How to locate the VF’s resource, like MMIO base address
Network Performance and Scalability with SR-IOV

Xen network scalability with software virtualization

Xen network scalability with Intel VT-c (SR-IOV)
SR-IOV - Summary

- **Pros**
  - Good performance
  - Good Scalability
  - A cost effective I/O virtualization solution
Agenda

• I/O Virtualization Overview
• Device Emulation
• Device Para-virtualization
• Device Direct Assignment
• SR-IOV
Backup
Interrupt Isolation Challenges

• In existing interrupt architecture on x86 Platform
  ▪ Interrupt is an DMA write to a specific address (0xFEEX_XXXX)
  ▪ Interrupt attribute is contained in the data/address
  ▪ Device issue DMA write according to guest driver setup

• Guest can trigger arbitrary interrupt through bogus DMA access
  ▪ it is not page grained, DMA remapping can’t cover this range
• **Interrupt remapping redefines the interrupt message format**
  - The DMA write include only an index to a remapping table
  - The Remapping table is setup in memory by VMM and includes the interrupt attribute
  - Interrupt Remapping Engine will translate the index to interrupt attribute
  - VMM need intercept guest’s setup to interrupt attribute (Message Interrupt setup)