Lecture 4
Device Virtualization
Larry Rudolph
Challenge and Promise of Virtualizing Devices

- Peripheral Device Drivers are major source of system crashes
- Device Drivers written by device manufacturer
- Execute in privileged mode
- Have access to all of memory
- Different upgrade cycle than OS’s
Outline

• **Types of Device Virtualization**
  - Direct Access
  - Emulated
  - Para-virtualized

• **Storage Virtualization**
  - Copy-on-Write Disks
Device Virtualization

• **Goals**
  – Isolation
  – Multiplexing
  – Speed
  – Mobility
  – Interposition

• **Device Virtualization Strategies**
  – Direct Access
  – Emulation
  – Para-virtualization
Direct Access Device

VM
Guest OS

CPU
MMU

Memory
Controller

Local Bus

Interface

High-Speed I/O Bus

NIC
LAN

Controller
Bridge
Frame Buffer

Low-Speed I/O Bus

CD-ROM
USB

USB
CD-ROM
Input-Output MMU

- Allows I/O device to DMA (directly transfer data) to physical memory.
- Device DMA’s to virtual addresses, and the IOMMU translates to a physical address.
Memory Isolation w/ Direct Access Device

VM Guest OS

CPU

Memory

MMU

Controller

MMU

IOMMU

Interface

NIC

Controller

Bridge

Frame Buffer

LAN

CD-ROM

USB

High-Speed I/O Bus

Low-Speed I/O Bus

Local Bus
Virtualization Enabled Device

VM1
Guest OS

CPU

VM2
Guest OS

Local Bus

MMU

Memory

Controller

IOMMU

High-Speed I/O Bus

vNIC 1
vNIC 2

NIC

Bridge

Frame Buffer

Low-Speed I/O Bus

LAN

Controller

Controller

CD-ROM

USB

USB
Direct Access Device Virtualization

- **Allow Guest OS direct access to underlying device**

- **Positives**
  - Fast
  - Simplify monitor
    - Limited device drivers needed

- **Negatives**
  - Need hardware support for safety (IOMMU)
  - Need hardware support for multiplexing
  - Hardware interface visible to guest
    - Limits mobility of VM
  - Interposition hard by definition
Emulated Devices

• **Emulate a device in class**
  – Emulated registers
  – Memory mapped I/O or programmed I/O

• **Convert**
  – Intermediate representation

• **Back-ends per real device**
Serial Port Example

User App

Serial Chip
ABC
Emulation

Generic Serial Layer

Host OS

Serial Chip
p
XYZ

LAN

Monitor

Guest

Serial Chip
ABC
Driver
Emulated Devices

• **Positives**
  – Platform stability
  – Allows interposition
  – No special hardware support needed
    • Isolation, multiplexing implemented by monitor

• **Negatives**
  – Can be slow
  – Drivers needed in monitor or host
I/O Done

- On X86, I/O done with privileged “IN” and “OUT” instructions
- In hosted environment, do world switch
- Sometimes the overhead is too high
  - For keyboard?
  - For USB?
  - For Network Interface Card (NIC)?
Virtualizing a NIC

- Packets arrive too fast for world switches

- Implement partially in the VMM and VM-app (in host)

- Virtual NIC is an AMD Lance, real NIC can be anything
Guest Sending Packet

- Guest OS does OUT to Lance
- Traps to VMM, world switch
- VMDriver calls VMApp (user mode)
- VMApp makes Host OS call
- Host OS sends packet out of real NIC
- World switch back to VMM
- VMM raises IRQ to Guest OS saying packet was sent
Guest Receiving Packet

- Real NIC raises real IRQ
- NIC delivers packet, VMAApp gets it via it’s SELECT system call
- VMAApp puts packet in shared memory and calls VMM
- VMM raises (virtual) IRQ to Guest OS
- Guest OS does IN-OUT ops to its NIC
- World switch Host, to VMAApp, back to VMM
Overheads

- Every real hardware access causes world switch
- I/O interrupt can cause more world switches
- Packet transmission involves two times driver overhead
- Packet is copied extra times
- Result: too much CPU overhead
Experimental Analysis

- 30.65 microsec spent in world switches and host
- If 17.55 spent in driver, then 13.10 is overhead
- 8.9 time in world switch (100x longer than system call)
- IRQ’s real & virtual take up lots of overhead
- Cannot recognize packets rec. for Host or particular guest — overhead grows #VMs
Solution

- Better Virtual NICs
  - Can access virtual NIC memory from user space
  - Single OUT, skips transmit IRQ
- Don’t send Guest packets immediately
  - wait till world switch; combine packets
  - don’t use SELECT, in VMApp — try to go directly to VMM
  - don’t have Guest OS ack every packet
- Avoid host (?) and handle NIC in VMM
  - Must support all the potential NIC devices, and more complications.
Solution (cont)

- Add virtualization to NIC
  - multiple MACs inside
  - allocate to each VM
- NIC now have more functionality
  - Wheel of Reincarnation
Network Receive Processing (today)

1. DMA packet into receive buffers owned by VMkernel
2. Raise physical Interrupt
3. Parse packet to find destination VM
4. Copy packet into guest receive queue
5. Raise virtual NIC interrupt
1. DMA packet into guest receive queue
2. Raise NIC interrupt at guest CPU
3. Raise virtual NIC interrupt
Passthrough I/O

- Guest drives device directly
  - Use case: I/O Appliances, High performance VMs

Requirements:
- I/O MMU for DMA Address Translation and protection (Intel® VT-d, AMD I/O MMU)
- Partitionable I/O device for sharing (PCI-SIG IOV SR/MR specification)

Diagram:
- Guest OS
  - Device Driver

- Virtualization Layer
- Device Manager

- I/O MMU

- I/O Device
  - PF = Physical Function, VF = Virtual Function
Challenges with Passthrough I/O

- Transparent VMotion
  - No simple way to checkpoint device state
  - Inability to write protect pre-copied guest memory
- Hardware independence
  - Checkpoint is probably limited to same HW vendor/model
  - HW errata/workarounds in the driver
- Memory over-commitment
  - No visibility of DMAs to guest memory
- VM Management
  - IHV driver for different vendors, device types, revisions, guest OSes
- Isolation/Security
  - E.g. Promiscuous Mode, SRC MAC address spoofing

Special VMs/Appliances may not have some of these requirements
Wheel of Reincarnation

• Proposed by T. Meyers and I. Sutherland in the 1960’s

  • Display processors: started as dumb things controlled by CPU

  • More and more functionality added to H/W

    • Far away from main CPU, cannot easily share parent computer’s memory, processing

  • Display processors began to be full computers

    • CPU, Memory

    • There are subsystems within display processor that start to get smarter as well

• Similar things happen with all “channels” or peripheral devices.
Virtio: An I/O virtualization framework for Linux

- Provide a standardized interface for emulated devices
- Front-end and back-end device driver
Driver Abstraction
High-Level Architecture
Object Hierarchy

```c
struct virtio_device {
    int index;
    struct device_dev;
    struct virtio_device_id *id_table;
    struct virtio_device_ops *config;
    unsigned long features();
    void *priv;
};

struct virtqueue {
    void (*callback)(struct virtqueue *vq);
    struct virtio_device *vdev;
    struct virtqueue_ops *vq_ops;
    void *priv;
};

struct virtqueue_ops {
    int (*add_buf)(struct virtqueue *vq,
                   struct scatterlist sq[],
                   unsigned int out_num,
                   unsigned int in_num,
                   void *data);
    void (*kick)(struct virtqueue *vq);
    void (*get_buf)(struct virtqueue *vq,
                    unsigned int *len);
    void (*disable_cb)(struct virtqueue *vq);
    bool (*enable_cb)(struct virtqueue *vq);
};
```

```c
struct virtio_config_ops {
    void (*get)(struct virtio_device *vdev,
                unsigned offset,
                void *buf, unsigned len);
    void (*set)(struct virtio_device *vdev,
                unsigned offset,
                const void *buf, unsigned len);
    u8 (*get_status)(struct virtio_device *vdev);
    void (*set_status)(struct virtio_device *vdev, u8 status);
    void (*reset)(struct virtio_device *vdev);
    struct virtqueue *(*find_vq)(struct virtio_device *vdev,
                                   unsigned index,
                                   void (*callback)(struct virtqueue *));
    void (*del_vq)(struct virtqueue *vq);
    u32 (*get_features)(struct virtio_device *vdev);
    void (*finalize_features)(struct virtio_device *vdev);
};
```
Para-Virtualized Devices

• Guest passes requests to Monitor at a higher abstraction level
  – Monitor calls made to initiate requests
  – Buffers shared between guest / monitor

• Positives
  – Simplify monitor
  – Fast

• Negatives
  – Monitor needs to supply guest-specific drivers
  – Bootstrapping issues
Traditional Architecture

Linux
Linux (devel)
Windows 7
Windows 8
MacOS

Virtual Machine Monitor

Hardware
Hosted Monitor Architecture

Host OS (Window 8)

Kernel Module

User App

Guest OS (Linux)

Virtual Machine Monitor

Hardware
VMware ESX 2.0

Figure 1: ESX Server architecture

Hybrid Ex 2 - Xen 3.0

- Para –virtualization
  - Linux Guest
- Hardware-supported virtualization
  - Unmodified Windows
- Isolated Device Drivers

Source: Ottawa Linux Symposium 2006 presentation.  
http://www.cl.cam.ac.uk/netos/papers/