Virtualisation

COSC349—Cloud Computing Architecture

David Eyers

Learning objectives

- Define **virtualisation**
- Give examples of virtualisation of many **different types of resource** (*e.g.*, CPU, memory, disk, network, *etc*.)
- Explain **challenges** in virtualising different resources
- Illustrate **useful capabilities** of virtual machines (VMs)
- Describe **key techniques** virtualisation engines use to virtualise x86/x64 OSs (*e.g.*, Linux, Windows, macOS, …)

Defining virtualisation

- Simulation and emulation often involve ‘pretending’ to be some different sort of hardware
- **Virtualisation** is about adding a **layer for manageability**
    - CPUs can be virtualised, but so can many other resources
    - Virtualisation support is increasingly **built into devices**
- FYI: Virtualisation first appeared in **mainframe technology**
    - Mainframes still used, but we’re highly unlikely to code on them
    - Mainframes typically have tight hardware+software binding

What resources commonly get virtualised?

- **CPU**—isolate and contain different environments
- **Memory**—many types of virtualisation abstraction
- **Storage**—‘hard disk’ in a file; directory subtree into VM
- **Networks**—map guest network needs onto host
- **Displays**—contain guest display within host display
- Other **peripherals**—*e.g.*, USB stack in VirtualBox

Virtual machines—VMs

- A set of virtualised resources can work together to provide a complete **virtual machine**, or **VM**.
- VirtualBox effects what's termed **hardware virtualisation**
  - Explore VirtualBox GUI to see what can be configured
  - CPU; RAM; storage all need to be set for a new VM
  - GUI offers configuration of many other parameters
    - Some options are quite obscure, but the documentation is good!
- Generally, VM needs (1) CPU, (2) memory, and (3) I/O

Some key, useful capabilities of VMs

- Ability to **pause** and **resume** VMs
  - Potential device interactions make this a non-trivial task!
- Can **snapshot** VMs' state and **restore** from that state
  - Handy to protect virtual resources such as hard disks
- Ability to **clone** new VMs from snapshots
  - However making useful copies of machines needs further work:
    - Windows SIDs need regeneration, or uniqueness fails
    - Normally MAC addresses on network cards will be different
- With above, can **migrate** VMs from one host to another

CPU virt. must protect VMs' OS kernels

- Need to understand what CPU does during a **system call**
  - Control must pass from **user space** to **kernel space**
  - Not as simple as executing a function call…
  - … but usually languages wrap syscalls in functions, *e.g.* `printf()`

- Often involves causing a software trap / exception
  - CPU goes into **protected mode** (AKA supervisor mode, …)
  - CPU saves program state of the caller
  - Jumps to privileged exception handler code
  - Eventually reverses protected mode, and **restores CPU state**

Fast virtualisation of CPUs

- **Goal**: run guest machine code mostly on the host CPU
- **Challenge**: must isolate host & guests from each other
  - Guest OS kernel needs to **believe** it has CPU protected mode
  - … but this can't safely be the **actual** CPU protected mode
- **Existing abstraction**: Intel CPUs support four "rings"
  - Rings isolate resources and define levels of privilege
  - Ring 0: runs operating system kernel
  - Ring 3: runs application code
  - Other rings stay largely unused in most typical OSs

Fast virtualisation of CPUs

- Typical Intel x86/x64 virtualisation **remaps protection rings**
  - Host kernel runs on **host CPU ring 0**
  - Guest OS kernel expects CPU ring 0 but is run on **host CPU ring 1**
  - Guest OS userspace is run on **host CPU ring 3**
  - Thus get "cheap" isolation of the desired sort… up to a point…

- Some operations can only actually be run from ring 0
  - *e.g.*, CPU instructions to interact with real hardware devices
  - A solution: **apply just-in-time re-compilation** to guest's code to avoid directly hitting these cases (this is expensive)

Virtual Memory

- RAM already has many levels of abstraction
- **Physical addresses** relate to RAM chips
- **Virtual addresses** get mapped into physical addresses
- Also, **paging** divides up memory into blocks

- 'Virtual memory' or 'paging' in older OSs was all about swapping processes' memory between RAM and disk
  - RAM / disk swapping is just one potential use of virtual memory

Fast virtualisation of memory

- **Goal**: guest memory use is host memory use
- **Challenge**: need to ensure protection of host memory
- **Existing abstraction**: virtual addresses; memory paging

- **Solution**: context switch to VM as you would a process
  - CPU helps facilitate switching processes with large RAM use
  - Prevent VM from seeing real host's memory management

- FYI: 32-bit versus 64-bit guests handled very differently

Fast virtualisation of disk

- **Goal**: guest has manageable ‘hard disks’
- **Challenge**: can’t safely share actual host hard disk

- Technically simple solution: **guest HD is huge file** on host
  - Map requests for guest HD read/write (sectors) into file on host
  - **Wasteful**: guest’s pointless management of non-real resources
  - Host space can be optimised to extend HD file on-demand
- Ideally pass through capabilities from host better…

Fast virtualisation of network cards (NICs)

- **Goal**: support guest networking as directly as possible
- **Existing abstractions**: plenty, including bridges, NAT, …

- Higher-end NICs offload work from CPU
  - Checksum calculations
  - IP fragmentation handling

- Ensure guest OS delegates functions to its (virtual) NIC
  - … since then virtualisation engine can support functions easily

Fast virtualisation of graphics

- **Goal**: get highest-level requests from guest
- **Existing abstractions**: e.g., OpenGL, DirectX, …

- OpenGL allows virtualisation host to avoid emulating graphics hardware—can largely pass through OpenGL
  - Do not want host intercepting per-pixel operations!
  - Need to avoid graphics ‘breaking out’ of guest though

- Alternative: no gfx. card—Use RDP or VNC+framebuffer