Virtual Machines

David E. Culler
CS162 – Operating Systems and Systems Programming
http://cs162.eecs.berkeley.edu/
Lecture 15
October 22, 2019
Mark your calendar

- November 14, Guest Lecture by Dr. Eric Brewer, Google VP of Infrastructure
  - Former UCB Faculty
  - Founded Inktomi – first fast massive search engine
    » inktomi.Berkeley.edu => inktomi.com
  - Responsible for re-architecting Google’s internals
  - Including Kubernetes.

- Let’s fill the room
  - Great chance to engage with one of the field’s leading thinkers
Paging Review

• Page Table provides a virtual address space for each process
  – Serves as the basis of protection

• Allows creation of the illusion of a very large memory dedicated to each
  – TLB translates every reference from Virtual to Physical
    » Fast, High associativity, Small, OS manages consistency

• Also basis for sharing (MMap), fast fork (copy—on-write), fast exec (load by swap)

• Key policy issue is page replacement policy
  – Approximation to LRU

• Clock algorithm (or N-th chance)
  – Scan through physical frames, replace first one that has not been accesses in last N passes (clock: N==1)
  – Reverse map: frame => PTE, accessed bit in PTE
Management of the Memory Hierarchy

Managed in Hardware

Processor
- TLB
- Registers
- L1 Cache
- L2 Cache
- L3 Cache (shared)

Accessed in Hardware
- Speed (ns): 0.3, 1, 3, 10-30
- Size (bytes): 100Bs, 10kBs, 100kBs, MBs

Managed in Software - OS

- Main Memory (DRAM)
- Secondary Storage (SSD)
- Secondary Storage (Disk)

- PT
- Managed in Hardware
- Managed in Software - OS

- TLB
- Registers

- Speed (ms): 10,000,000, 100,000 (0.1 ms), 10,000,000 (10 ms)
- Size (bytes): 100Bs, 100GBs, TBs
Demand Paging Mechanisms

- **PTE makes demand paging implementable**
  - Valid ⇒ Page in memory, PTE points at physical page
  - Not Valid ⇒ Page not in memory; use info in PTE to find it on disk when necessary

- **Suppose user references page with invalid PTE?**
  - Memory Management Unit (MMU) traps to OS
    » Resulting trap is a “Page Fault”
  - What does OS do on a Page Fault?:
    » Choose an old page to replace
    » If old page modified (“D=1”), write contents back to disk
    » Change its PTE and any cached TLB to be invalid
    » Load new page into memory from disk
    » Update page table entry, invalidate TLB for new entry
    » Continue thread from original faulting location
  - TLB for new page will be loaded when thread continued!
  - While pulling pages off disk for one process, OS runs another process from ready queue
    » Suspended process sits on wait queue
Recall: Clock Algorithm (aka Not Recently Used)

- Which bits of a PTE entry are useful to us?
  - **Use**: Set when page is referenced; cleared by clock algorithm
  - **Modified**: set when page is modified, cleared when page written to disk
  - **Valid**: ok for program to reference this page
  - **Read-only**: ok for program to read page, but not modify
    » For example for catching modifications to code pages!

- **Clock Algorithm**: pages arranged in a ring
  - On page fault (or background reaper):
    » Advance clock hand (not real time)
    » Check **use bit**: 1→used recently; clear and leave alone
    0→selected candidate for replacement
  - Crude partitioning of pages into two groups: young and old

Single Clock Hand:
Advances through memory frames
Check for pages not used recently
Best candidates for eviction
Recall: Clock, Free List, 2nd Chance

- Keep set of free pages ready for use in demand paging
  - Freelist filled in background by Clock algorithm or other technique ("Pageout demon")
  - Dirty pages start copying back to disk when enter list
- Like VAX second-chance list
  - If page needed before reused, just return to active set
- Advantage: faster for page fault
  - Can always use page (or pages) immediately on fault
Reverse Page Mapping (Sometimes called “Coremap”)

- Physical page frames often shared by many different address spaces/page tables
  - All children forked from given process
  - Shared memory pages between processes

- Whatever reverse mapping mechanism that is in place must be fast
  - Must hunt down all page tables pointing at given page frame when freeing a page
  - Must hunt down all PTEs when seeing if pages “active”

- Implementation options:
  - For every page descriptor, keep linked list of page table entries that point to it
    » Management nightmare – expensive
  - Linux: Object-based reverse mapping
    » Link together memory region descriptors instead (much coarser granularity)
Allocation of Page Frames (Memory Pages)

• How do we allocate memory among different processes?
  – Does every process get the same fraction of memory? Different fractions?
  – Should we completely swap some processes out of memory?

• Each process needs minimum number of pages
  – Want to make sure that all processes that are loaded into memory can make forward progress
  – Example: IBM 370 – 6 pages to handle SS MOVE instruction:
    » instruction is 6 bytes, might span 2 pages
    » 2 pages to handle from
    » 2 pages to handle to

• Possible Replacement Scopes:
  – Global replacement – process selects replacement frame from set of all frames; one process can take a frame from another
  – Local replacement – each process selects from only its own set of allocated frames
Fixed/Priority Allocation

• **Equal allocation (Fixed Scheme):**
  – Every process gets same amount of memory
  – Example: 100 frames, 5 processes → process gets 20 frames

• **Proportional allocation (Fixed Scheme)**
  – Allocate according to the size of process
  – Computation proceeds as follows:
    \[ s_i = \text{size of process } p_i \text{ and } S = \sum s_i \]
    \[ m = \text{total number of physical frames in the system} \]
    \[ a_i = (\text{allocation for } p_i) = \frac{s_i}{S} \times m \]

• **Priority Allocation:**
  – Proportional scheme using priorities rather than size
    » Same type of computation as previous scheme
  – Possible behavior: If process \( p_i \) generates a page fault, select for replacement a frame from a process with lower priority number

• Perhaps we should use an adaptive scheme instead???
  – What if some application just needs more memory?
What about Compulsory Misses?

• Recall that compulsory misses are misses that occur the first time that a page is seen
  – Pages that are touched for the first time
  – Pages that are touched after process is swapped out/swapped back in

• Clustering:
  – On a page-fault, bring in multiple pages “around” the faulting page
  – Since efficiency of disk reads increases with sequential reads, makes sense to read several sequential pages

• Working Set Tracking:
  – Use algorithm to try to track working set of application
  – When swapping process back in, swap in working set
On to Virtual Machines …
OS software supports user level software

- Illusion: virtualizes hardware resources and provides convenient high level user abstractions & services
- Referee: provide isolation & protection, allocates resources to user processes
- Glue: efficient access, sharing, resource management
Today: turn system / user inside out
Recall (L1) Virtual Machines

• Virtualize every detail of a hardware configuration so perfectly that you can run an operating system (and many applications) on top of it.
  – VMWare Fusion, Virtual box, Parallels Desktop, Xen, Vagrant

• Provides isolation

• Complete insulation from change

• The norm in the Cloud (server consolidation)

• Long history (60’s in IBM OS development)

• All our work will take place INSIDE a VM
  – Vagrant (new image just for you)
System Virtual Machines: Layers of OSs

- Useful for OS development
  - When OS crashes, restricted to one VM
  - Can aid testing programs on other OSs
Containers virtualize the OS

• Roots in OS developments to provide protected systems abstraction, not just application abstraction
  – User-level file system (route syscalls to user process)
  – Cgroups – predictable, bounded resources (CPU, Mem, BW)
Motivation

• An “Application” is no longer a stand-alone executable or composition of vendor-supplied software, it is often a complex platform utilizing several deep software stacks, many processes, shared libraries and services, …
  – All of which evolve fairly rapidly

• To stand up a viable instance requires “all of it”, so need to wrap up the “entire stack”
  – The OS and all its user-level daemons, the software installed in its file system
  – The hardware itself

• Want to isolate this entire ensemble from other ensembles, which may be on the same physical machine (server consolidation)

• Want to be able to allocate resources predictably
Our “Host Operating System”

Hardware
- Processor
- Memory
- I/O Devices

System Software
- scheduler
- syscall tbl
- intr tbl
- intrpt handlers
- syscall handlers
- subsysystems
- Page Tables
- regs

User Software
User-level “Guest” Operating System

- User Software
  - appln process

- System Software
  - scheduler
  - process & thread mgmt
  - VM system
  - Page Tables
  - syscall tbl
  - intr tbl
  - syscall handlers
  - intrpt handlers
  - file systems
  - Drivers
  - VM system

- Hardware
  - Processor
  - Memory
  - I/O Devices

[Diagram showing the structure and components of a user-level guest operating system.]
Example: my Macbook Pro / OS-X

```
(base) CullerMac19:~ culler$ vm_stat
Mach Virtual Memory Statistics: (page size of 4096 bytes)
Pages free: 1374276.
Pages active: 887789.
Pages inactive: 634340.
Pages speculative: 254822.
Pages throttled: 0.
Pages wired down: 703546.
Pages purgeable: 63546.
"Translation faults": 2069239178.
Pages copy-on-write: 54900953.
Pages zero filled: 1485741210.
Pages reactivated: 1061165.
Pages purged: 2294989.
File-backed pages: 573432.
Anonymous pages: 1203439.
Pages stored in compressor: 1645906.
Pages occupied by compressor: 338775.
Decompressions: 68562947.
Compressions: 84641004.
Pageins: 67082753.
Pageouts: 10069.
Swapins: 71555118.
Swapouts: 74956494.
```

```
(base) CullerMac19:~ culler$ hostinfo
Mach kernel version:
Darwin Kernel Version 18.7.0: Thu Jun 20 18:12:21 PDT 2019; root:xnu-4903.278.47~4/RELEASE_X86_64
Kernel configured for up to 12 processors. 6 processors are physically available.
12 processors are logically available. Processor type: x86_64h (Intel x86-64 Haswell)
Procesor active: 0 1 2 3 4 5 6 7 8 9 10 11
Primary memory available: 16.08 gigabytes
Default processor set: 480 tasks, 1875 threads, 12 processors
Load average: 0.53, Mach factor: 11.46
```
Up that Virtual Machine

(base) CullerMac19:cs162-vm culler$ vagrant up
/opt/vagrant/embedded/gems/2.2.4/gems/vagrant-2.2.4/lib/vagrant/util/which.rb:37: warning: Insecure world writable dir /Users/culler/devel in PATH, mode 040777
Bringing machine 'default' up with 'virtualbox' provider...

- default: Checking if box 'cs162/fall2019' version '1.0.0' is up to date...
- default: Clearing any previously set forwarded ports...
- default: Clearing any previously set network interfaces...
- default: Preparing network interfaces based on configuration...
  default: Adapter 1: nat
  default: Adapter 2: hostonly
- default: Forwarding ports...
  default: 22 (guest) => 2222 (host) (adapter 1)
- default: Running 'pre-boot' VM customizations...
- default: Booting VM...
- default: Waiting for machine to boot. This may take a few minutes...
  default: SSH address: 127.0.0.1:2222
  default: SSH username: vagrant
  default: SSH auth method: private key
  default: Warning: Connection reset. Retrying...
  default: Warning: Remote connection disconnect. Retrying...
- default: Machine booted and ready!
- default: Checking for guest additions in VM...
  default: The guest additions on this VM do not match the
  default: VirtualBox! In most cases this is fine, but ...
  default: prevent things such as shared folders from wo
  default: shared folder errors, please make sure the gu
  default: virtual machine match the version of VirtualB
  default: your host and reload your VM.
  default: default: Guest Additions Version: 5.2.32
  default: VirtualBox Version: 6.0
- default: Setting hostname...
- default: Configuring and enabling network interfaces..
- default: Mounting shared folders...
  default: /vagrant => /Users/culler
- default: Machine already provisioned. Run `vagrant pro
- default: flag to force provisioning. Provisioners mark

(base) CullerMac19:cs162-vm culler$ vagrant ssh
/opt/vagrant/embedded/gems/2.2.4/gems/vagrant-2.2.4/lib/vagrant/util/which.rb:37: warning: Insecure world writable dir /Users/culler/devel in PATH, mode 040777
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-65-generic x86_64)

* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage

System information as of Thu Oct 17 16:18:56 UTC 2019

System load: 0.21 Processes: 104
Usage of /: 29.4% of 9.63GB Users logged in: 0
Memory usage: 14% IP address for enp0s3: 10.0.2.15
Swap usage: 0% IP address for enp0s8: 192.168.162.162

* Kata Containers are now fully integrated in Charmed Kubernetes 1.16!
Yes, charms take the Krazy out of K8s Kata Kluster Konstruktion.

https://ubuntu.com/kubernetes/docs/release-notes

64 packages can be updated.
0 updates are security updates.
In the Host OS

Processes: 409 total, 3 running, 406 sleeping, 1932 threads
Load Avg: 1.06, 1.16, 1.09  CPU usage: 0.83% user, 1.52% sys, 97.64% idle
SharedLibs: 316M resident, 53M data, 49M linkedit.
MemRegions: 118007 total, 3151M resident, 118M private, 1283M shared.
PhysMem: 10G used (3581M wired), 5770M unused.
VM: 2354G vsize, 1373M framework vsize, 7156460(0) swapins, 74956494(0) swapouts.

<table>
<thead>
<tr>
<th>PID</th>
<th>COMMAND</th>
<th>%CPU</th>
<th>TIME</th>
<th>#TH</th>
<th>#WQ</th>
<th>#PORTS</th>
<th>MEM</th>
<th>PERG</th>
<th>CMPS</th>
<th>PGPR</th>
<th>PPD</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>89197</td>
<td>siriknowled</td>
<td>0.0</td>
<td>00:00:00.92</td>
<td>2</td>
<td>2</td>
<td>47</td>
<td>2668K</td>
<td>0B</td>
<td>1724K</td>
<td>89197</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>89193</td>
<td>com.apple.DF</td>
<td>0.0</td>
<td>00:18:97</td>
<td>2</td>
<td>1</td>
<td>67</td>
<td>2684K</td>
<td>0B</td>
<td>1852K</td>
<td>89193</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>82453</td>
<td>PAH_Extensio</td>
<td>0.0</td>
<td>00:28:51</td>
<td>3</td>
<td>1</td>
<td>235</td>
<td>16M</td>
<td>0B</td>
<td>7216K</td>
<td>82453</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>75819</td>
<td>tzlinkd</td>
<td>0.0</td>
<td>00:00:01</td>
<td>2</td>
<td>1</td>
<td>14</td>
<td>452K</td>
<td>0B</td>
<td>448K</td>
<td>75819</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>75787</td>
<td>MTLCompilerS</td>
<td>0.0</td>
<td>00:00:10</td>
<td>2</td>
<td>2</td>
<td>24</td>
<td>9032K</td>
<td>0B</td>
<td>9024K</td>
<td>75787</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>75767</td>
<td>secd</td>
<td>0.0</td>
<td>00:00:84</td>
<td>2</td>
<td>1</td>
<td>36</td>
<td>3188K</td>
<td>0B</td>
<td>2552K</td>
<td>75776</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>75098</td>
<td>DiskUnmountW</td>
<td>0.0</td>
<td>00:00:51</td>
<td>2</td>
<td>1</td>
<td>36</td>
<td>1412K</td>
<td>0B</td>
<td>1128K</td>
<td>75098</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>75093</td>
<td>MTLCompilerS</td>
<td>0.0</td>
<td>00:00:06</td>
<td>2</td>
<td>1</td>
<td>21</td>
<td>5924K</td>
<td>0B</td>
<td>5916K</td>
<td>75093</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>74938</td>
<td>ssh-agent</td>
<td>0.0</td>
<td>00:00:11</td>
<td>1</td>
<td>0</td>
<td>21</td>
<td>908K</td>
<td>0B</td>
<td>772K</td>
<td>74938</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>56389</td>
<td>usbmuxd</td>
<td>0.0</td>
<td>00:02:39</td>
<td>3</td>
<td>3</td>
<td>38</td>
<td>1592K</td>
<td>0B</td>
<td>1008K</td>
<td>56389</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>55576</td>
<td>SimulatorTra</td>
<td>0.0</td>
<td>00:27:65</td>
<td>3</td>
<td>1</td>
<td>137</td>
<td>9252K</td>
<td>0B</td>
<td>5212K</td>
<td>55576</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>55575</td>
<td>com.apple.C0</td>
<td>0.0</td>
<td>00:28:03</td>
<td>3</td>
<td>1</td>
<td>143</td>
<td>10M</td>
<td>0B</td>
<td>6316K</td>
<td>55575</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>54313</td>
<td>com.apple.C0</td>
<td>0.0</td>
<td>00:02:01</td>
<td>2</td>
<td>1</td>
<td>14</td>
<td>908K</td>
<td>0B</td>
<td>9024K</td>
<td>54313</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>53416</td>
<td>kproxy</td>
<td>0.0</td>
<td>00:01:53</td>
<td>2</td>
<td>1</td>
<td>44</td>
<td>3488K</td>
<td>0B</td>
<td>2604K</td>
<td>53416</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>16871</td>
<td>screencaptur</td>
<td>0.2</td>
<td>00:00:32</td>
<td>11</td>
<td>9</td>
<td>211</td>
<td>13M</td>
<td>84K</td>
<td>0B</td>
<td>16871</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>16870</td>
<td>screencaptur</td>
<td>1.9</td>
<td>00:00:19</td>
<td>3</td>
<td>2</td>
<td>57</td>
<td>3048K+ 620K</td>
<td>0B</td>
<td>383</td>
<td>383</td>
<td>sleeping</td>
<td></td>
</tr>
<tr>
<td>16858</td>
<td>ssh</td>
<td>0.0</td>
<td>00:01:53</td>
<td>1</td>
<td>0</td>
<td>21</td>
<td>1448K</td>
<td>0B</td>
<td>16857</td>
<td>16857</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>16857</td>
<td>vagrant</td>
<td>0.0</td>
<td>00:00:00</td>
<td>5</td>
<td>0</td>
<td>17</td>
<td>612K</td>
<td>0B</td>
<td>16857</td>
<td>16856</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>16856</td>
<td>CoreServices</td>
<td>0.0</td>
<td>00:00:07</td>
<td>3</td>
<td>2</td>
<td>163</td>
<td>4544K</td>
<td>0B</td>
<td>0B</td>
<td>16856</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>16690</td>
<td>worker_sha</td>
<td>0.0</td>
<td>00:00:02</td>
<td>3</td>
<td>1</td>
<td>50</td>
<td>3068K</td>
<td>0B</td>
<td>16690</td>
<td>1</td>
<td>sleeping</td>
<td></td>
</tr>
<tr>
<td>16689</td>
<td>worker_sha</td>
<td>0.0</td>
<td>00:00:02</td>
<td>3</td>
<td>1</td>
<td>50</td>
<td>3192K</td>
<td>0B</td>
<td>16689</td>
<td>1</td>
<td>sleeping</td>
<td></td>
</tr>
<tr>
<td>16688</td>
<td>worker_sha</td>
<td>0.0</td>
<td>00:00:02</td>
<td>3</td>
<td>1</td>
<td>50</td>
<td>3156K</td>
<td>0B</td>
<td>16688</td>
<td>1</td>
<td>sleeping</td>
<td></td>
</tr>
<tr>
<td>16687</td>
<td>worker_sha</td>
<td>0.0</td>
<td>00:00:03</td>
<td>3</td>
<td>1</td>
<td>50</td>
<td>3188K</td>
<td>0B</td>
<td>16687</td>
<td>1</td>
<td>sleeping</td>
<td></td>
</tr>
<tr>
<td>16678</td>
<td>worker_sha</td>
<td>0.0</td>
<td>00:00:04</td>
<td>4</td>
<td>1</td>
<td>50</td>
<td>5192K</td>
<td>0B</td>
<td>16678</td>
<td>1</td>
<td>sleeping</td>
<td></td>
</tr>
<tr>
<td>16675</td>
<td>VBoxNetDHCP</td>
<td>0.0</td>
<td>00:00:02</td>
<td>2</td>
<td>1</td>
<td>58</td>
<td>4812K</td>
<td>0B</td>
<td>16675</td>
<td>16627</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>16674</td>
<td>VBoxHeadless</td>
<td>1.2</td>
<td>02:30:35</td>
<td>28</td>
<td>1</td>
<td>293</td>
<td>840M</td>
<td>0B</td>
<td>16674</td>
<td>16627</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>16645</td>
<td>writeconfig</td>
<td>0.0</td>
<td>00:00:01</td>
<td>2</td>
<td>2</td>
<td>18</td>
<td>980K</td>
<td>0B</td>
<td>964K</td>
<td>16645</td>
<td>1</td>
<td>sleeping</td>
</tr>
<tr>
<td>16627</td>
<td>VBoxSVC</td>
<td>0.0</td>
<td>00:06:52</td>
<td>15</td>
<td>2</td>
<td>100</td>
<td>7496K</td>
<td>0B</td>
<td>16627</td>
<td>1</td>
<td>sleeping</td>
<td></td>
</tr>
<tr>
<td>16625</td>
<td>VBoxXPCOMIPC</td>
<td>0.0</td>
<td>00:02:84</td>
<td>1</td>
<td>0</td>
<td>21</td>
<td>2544K</td>
<td>0B</td>
<td>16625</td>
<td>1</td>
<td>sleeping</td>
<td></td>
</tr>
<tr>
<td>16566</td>
<td>bash</td>
<td>0.0</td>
<td>00:00:03</td>
<td>1</td>
<td>0</td>
<td>21</td>
<td>924K</td>
<td>0B</td>
<td>16566</td>
<td>16565</td>
<td>1</td>
<td>sleeping</td>
</tr>
</tbody>
</table>
In the vbox / vagrant

```bash
vagrant@development [16:17:00] ~ $ lshw --short
WARNING: you should run this program as super-user.
H/W path  Device  Class  Description
==================================================================================================
 system   Computer
 bus      Motherboard
/0        memory     985MiB System memory
/0/0      bridge     440FX - 82441FX PM [Natoma]
/0/0/1    bridge     82371SB PIIX3 ISA [Natoma/Triton II]
/0/0/1.1  storage    82371AB/EB/MB PIIX4 IDE
/0/0/2    display    VirtualBox Graphics Adapter
/0/0/3    enp0s3    network  82540EM Gigabit Ethernet Controller
/0/0/4    generic    VirtualBox Guest Service
/0/0/5    multimedia 82801AA AC'97 Audio Controller
/0/0/7    bridge     82371AB/EB/MB PIIX4 ACPI
/0/0/8    enp0s8    network  82540EM Gigabit Ethernet Controller
/0/0/14   scsi2      storage  53c1030 PCI-X Fusion-MPT Dual Ultra320 SCSI
```

```bash
vagrant@development [16:17:42] ~ $ sudo lshw --short
WARNING: output may be incomplete or inaccurate, you should run this program as super-user.
vagrant@development [16:17:42] ~ $ 
```

```bash
vagrant@development [16:17:59] ~ $
```
Guest ubuntu Linux

top - 16:20:20 up 11 min, 1 user, load average: 0.00, 0.01, 0.00
Tasks: 98 total, 1 running, 54 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.2 us, 0.0 sy, 0.0 ni, 99.8 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem: 1008948 total, 482096 free, 94156 used, 432696 buff/cache
KiB Swap: 0 total, 0 free, 0 used. 769064 avail Mem

<table>
<thead>
<tr>
<th>PID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>%CPU</th>
<th>%MEM</th>
<th>TIME+</th>
<th>COMMAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>77816</td>
<td>9016</td>
<td>6672</td>
<td>S</td>
<td>0.0</td>
<td>0.9</td>
<td>0:42.31</td>
<td>systemd</td>
</tr>
<tr>
<td>2</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>kthread</td>
</tr>
<tr>
<td>3</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.01</td>
<td>kworker/0:0</td>
</tr>
<tr>
<td>4</td>
<td>root</td>
<td>0</td>
<td>-20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>kworker/0:0H</td>
</tr>
<tr>
<td>5</td>
<td>root</td>
<td>0</td>
<td>-20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>mm_percpu_wq</td>
</tr>
<tr>
<td>7</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.15</td>
<td>ksoftirqd/0</td>
</tr>
<tr>
<td>8</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.07</td>
<td>rcu_sched</td>
</tr>
<tr>
<td>9</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>rcu_bh</td>
</tr>
<tr>
<td>10</td>
<td>root</td>
<td>rt</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>migration/0</td>
</tr>
<tr>
<td>11</td>
<td>root</td>
<td>rt</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>watchdog/0</td>
</tr>
<tr>
<td>12</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>cpuhp/0</td>
</tr>
<tr>
<td>13</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>cpuhp/1</td>
</tr>
<tr>
<td>14</td>
<td>root</td>
<td>rt</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>watchdog/1</td>
</tr>
<tr>
<td>15</td>
<td>root</td>
<td>rt</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>migration/1</td>
</tr>
<tr>
<td>16</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.03</td>
<td>ksoftirqd/1</td>
</tr>
<tr>
<td>18</td>
<td>root</td>
<td>0</td>
<td>-20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>kworker/1:0H</td>
</tr>
<tr>
<td>19</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>kdevtmpfs</td>
</tr>
<tr>
<td>20</td>
<td>root</td>
<td>0</td>
<td>-20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>I</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>netns</td>
</tr>
<tr>
<td>21</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.00</td>
<td>rcu_tasks_kthre</td>
</tr>
<tr>
<td>22</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
<td>0:00.01</td>
<td>kauditd</td>
</tr>
</tbody>
</table>
And networking

(base) CullerMac19:~ culler$ ifconfig
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
options=1203<RXCSUM,TXCSUM,TXSTATUS,SW_TIMESTAMP>
inet 127.0.0.1 netmask 0xff000000
inet6 ::1 prefixlen 128
inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
nd6 options=201<PERFORMNUD,DAD>
status: active
glo0: flags=8018<POINTOPOINT,MULTICAST> mtu 1280
stf0: flags=0<-> mtu 1280
XHC20: flags=0<-> mtu 0
VHC128: flags=0<-> mtu 0
XHC0: flags=0<-> mtu 0
XHC1: flags=0<-> mtu 0
en5: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
ether 68:55:1f:6c:2e:41
inet6 fe80::e0:68ff:fe00:1122/en5 prefixlen 64 scopeid 0x8
nd6 options=201<PERFORMNUD,DAD>
media: autoselect (100baseTX <full-duplex>)
status: active
ap1: flags=8002<BROADCAST,SIMPLEX,MULTICAST> mtu 1500
ether f2:18:98:09:f1:9e
media: autoselect
status: inactive
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
ether 68:55:1f:6c:2e:41
inet6 fe80::13:d3b:8f93:c347/en0 prefixlen 64 secured scopeid 0x8
inet6 2607:f140:000:0:a09:c5f:ca5:e9c0:d90 prefixlen 64 autoconf secure
inet6 2607:f140:000:0:a09:9010:14f:1815:0020 prefixlen 64 autoconf temporary
inet 10.142.39.255 netmask 0xfffff00 broadcast 10.142.39.255
nd6 options=201<PERFORMNUD,DAD>
media: autoselect
status: inactive

prun0: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 2000
inet6 fe80::c655:5dfe:1be1:cd7/utun0 prefixlen 64 scopeid 0x12
nd6 options=201<PERFORMNUD,DAD>

utun0: flags=8051<UP,POINTOPOINT,RUNNING,MULTICAST> mtu 2000
inet6 fe80::c655:5dfe:1be1:cd7/utun0 prefixlen 64 scopeid 0x12
nd6 options=201<PERFORMNUD,DAD>

vboxnet0: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
ether 0a:00:27:00:00:00
inet 192.168.162.1 netmask 0xfffff00 broadcast 192.168.162.255
vboxnet1: flags=8842<BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
ether 0a:00:27:00:00:01

vagrant@development [16:23:20] ~ $ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.162.162 netmask 255.255.255.0 broadcast 192.168.162.255
inet6 fe80::a00:27ff:fed8:b072 prefixlen 64 scopeid 0x20<link>
ether 08:00:27:d8:b0:72 txqueuelen 1000 (Ethernet)
RX packets 312 bytes 28704 (28.7 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 292 bytes 41012 (41.0 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
inet 192.168.162.162 netmask 255.255.255.0 broadcast 192.168.162.255
inet6 fe80::a00:27ff:fed8:b072 prefixlen 64 scopeid 0x20<link>
ether 08:00:27:d8:b0:72 txqueuelen 1000 (Ethernet)
RX packets 312 bytes 28704 (28.7 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 292 bytes 41012 (41.0 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x10<host>
loop txqueuelen 1000 (Local Loopback)
RX packets 52 bytes 4447 (4.4 KB)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 52 bytes 4447 (4.4 KB)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

management [16:23:27] ~ $
User-level “Guest” Operating System

User Software
- appln process

System Software
- scheduler
- process & thread mgmt
- VM system
- Page Tables
- syscall tbl
- intr tbl
- intrpt handlers
- syscall handlers
- file systems
- Drivers
- file systems
- syscall tbl
- intr tbl
- intrpt handlers
- syscall handlers
- file systems
- Drivers

Hardware
- Processor
- Memory
- I/O Devices

“virtual hardware”
Software Emulation of Hardware

• Example: QEMU for x86
  – Used to do MIPS R3000 (subset) emulation as 61C project

• User software emulates the behavior of every single instruction
  – Data structure for Processor, Memory, I/O, etc
  – Code for Instruction Cycle: Fetch Instruction, Decode, Fetch Operands, Perform Operation, Store Results, Update PC
  – Emulate privilege levels, interrupts, MMU, . . . too
  – Load (i.e., boot) image of the operating system
  – Initializes (virtual) HW, loads pgms, schedules threads, etc.

• Popular in the 90’s
  – run Windows (x86) on your MAC (M68000), …
  – Software Fault Isolation

• Lots of dynamic translation optimizations to reduce overhead from 1000-fold to 2-10 fold
  – Want more like 10-20% overhead
Guest Virtual Hardware: proc regs

Hardware
- Processor
- Memory
- I/O Devices

System Software
- System calls
- Process & thread mgmt
- VM system
- Page Tables
- Intrpt handlers
- Syscall handlers
- File systems
- Drivers
- Scheduler

Guest OS
- appln process
- appln process
- syscall tbl
- syscall handlers
- Intr tbl
- Intrpt handlers
- Drivers
- “virtual hardware”

Unprivileged Instructions

Privileged Instructions

User Software
- appln process
Virtual Hardware: “physical mem”

- **User Software**
  - `appln process`
  - `mmap <file>`

- **Hardware**
  - `Processor`
  - `Memory`
  - `I/O Devices`

- **System Software**
  - `scheduler`
  - `process & thread mgmt`
  - `VM system`
  - `Page Tables`
  - `regs`

- **System Software**
  - `Guest OS`
  - `Guest OS`
  - `Unprivileged Instructions`
  - `Privileged Instructions`
  - `VM system`
  - `syscall handlers`
  - `intrpt handlers`
  - `syscall tbl`
  - `intr tbl`
  - `Drivers`
  - `file systems`

- **Virtual Hardware**
  - `virtual hardware`
  - `appln process`
  - `appln process`
  - `appln process`
Virtual Hardware: “I/O Device”

User Software
- appln process
- mmap <file>

VM process VAS
- host file

System Software
- scheduler
- process & thread mgmt
- VM system
- Page Tables
- syscall tbl
- intr tbl
- intrpt handlers
- syscall handlers
- file systems

Hardware
- Processor
- Memory
- I/O Devices

Guest OS
- “virtual hardware”
- mem
- scsi disk

Unprivileged Instructions
- Privileged Instructions
Virtual Hardware: “I/O Devices”

- **User Software**
  - appln process
  - mmap <file>
  - window

- **VM process VAS**
  - host file

- **VM-X**
  - appln process
  - syscall
  - intr

- **Guest OS**
  - syscall handlers
  - intrpt handlers
  - file systems

- **System Software**
  - syscall tbl
  - intr tbl
  - Page Tables
  - process & thread mgmt
  - VM system
  - file systems

- **Hardware**
  - Processor
  - Memory
  - I/O Devices
  - sched الحر
  - process & thread mgmt
  - VM system

- **Virtual Hardware**
  - mmap <file>
  - window
  - appln process
  - syscall
  - intr

- **Drivers**
  - syscal handlers
  - intrpt handlers
  - file systems

- **Guest OS**
  - syscall handlers
  - intrpt handlers
  - file systems

- **VM system**
  - Page Tables

- **File Systems**
  - file systems

- **Drivers**
  - Drivers

- **VM process VAS**
  - host file

- **Virtual Hardware**
  - virtual hardware
Virtual Hardware: “I/O Devices”

User Software
- appln process
- mmap <file>
- window
- nw intf

Hardware
- Processor
- Memory
- I/O Devices

System Software
- scheduler
- process & thread mgmt
- VM system
- Page Tables
- syscall tbl
- intr tbl
- intrpt handlers

Guest OS
- VM process VAS
- VM-X
- appln process
- appln
- file systems
- syscall handlers
- file systems
- intrpt handlers
- Drivers
- mem
- scsi disk
- graphics
- nic
- "virtual hardware"
Challenges

• How do we provide virtual address spaces for each of the guest processes?
  – Host OS has the REAL page tables

• When Guest process does syscall (trap), how does it vector through the Guest OS Syscall Table?
  – Interrupts go through the physical interrupt vector

• How does the guest OS protect itself from its guest application processes & each other?

• Guest OS executes as Unprivileged
  – How does it set up its page table? Disable interrupts? …
  – Driver Read/Write to I/O devices? Handle interrupts?

• How do interrupts get delivered to Guest OS?

• Deal with the huge diversity of I/O devices that might be on the host machine?
I/O diversity (1st Try)

• Guest OS is configured with drivers for a few particular (virtual) I/O devices
  – They claim to be these (often old) devices (SCSI disk, …)
• Driver accesses its device through:
• Programmed I/O to memory-mapped registers
  – Read status registers, Write Control Registers
  – Page Table translates to specific physical addresses of I/O device registers
• Direct Memory Access (DMA)
  – Bulk transfer to/from a memory buffer & device
• Device raises interrupt line when operations complete
• => Catch these operations and emulate the device on the host object (file, NW, window)
Guest User Level Memory Access

• Guest User Process accesses a virtual address space set up by the Guest OS

• But hardware translates the Virtual to Physical Address through the REAL Page Table
  – Not the Guest OS Page Table

• ???
The Two Translations

• Guest process operates on virtual addresses
  – It can’t see what physical address these translate into

• Guest OS set up mapping to “guest mem”
  – It thinks these are physical addresses
  – They are actually within a mapped region of the VM process

• Host OS maps VM process VAS to physical mem
VMM Shadow Page Tables

Guest OS Page Tables

Guest VAS pages

Guest Physical Frames

Host VM Process Pages

Host Physical Frames

Host OS Page Table for VM Process

VMM Page Tables for VM

Shadow VAS pages

Host Physical Frames
VMM Shadow Page Tables (cont)

Guest OS Page Tables

Guest VAS pages

Guest Physical Frames

Guest OS Page Table for VM Process

Host VM Process Pages

Host Physical Frames

Host OS Page Table for VM Process

VMM Page Tables for VM

Shadow VAS pages

Host Physical Frames
Virtual Machine Monitor

User Software
- appln process

System Software
- scheduler
- process & thread mgmt
- VM system
- Page Tables
- syscall tbl
- intr tbl
- intrpt handlers
- syscall handlers
- file systems
- drivers

Hardware
- Processor
- Memory
- I/O Devices

Virtual Machine Monitor
- Kernel module
- Driver

Unprivileged Instructions

Privileged Instructions

“virtual hardware”
Virtual Machine Monitor

• Extends the host operating system using established dynamic mechanisms
  – Kernel module / Driver
  – Requires administrative privileges – but not reboot
  – Has visibility into page table management, interrupt control, hardware configuration

• Maintains shadow page tables for every page table in the guest OS
  – Mapping from Guest OS Process VAS to Host Physical Mem

• How do these shadow page tables get used?
Virtual Machine Monitor

User Software
- appln process

System Software
- scheduler
- process & thread mgmt
- VM system
- Page Tables
- syscall tbl
- intr tbl
- intrpt handlers
- syscall handlers
- file systems
- process & thread mgmt
- VM system

Hardware
- Processor
- Memory
- I/O Devices

Guest OS
- syscall tbl
- intr tbl
- intrpt handlers
- syscall handlers
- file systems
- process & thread mgmt
- VM system

“virtual hardware”

Guest OS
- syscall handlers
- file systems
- process & thread mgmt
- VM system

Unprivileged Instructions

Privileged Instructions

Virtual Machine Monitor
- Kernel module
- Driver
VMM – VM – VMM extension process

User Software
- appln process
  - host file
  - window
  - nw intf

Guest OS
- syscall tbl
- intr tbl
- intrpt handlers
- syscall handlers
- file systems
- Drivers

Unprivileged Instructions

Privileged Instructions

Hardware
- Processor
- Memory
- I/O Devices
- VMM
- Scheduler
- Process & thread mgmt
- VM system
- Page Tables
- syscall tbl
- intr tbl
- intrpt handlers
- syscall handlers

System Software
- Drivers
- VMM-X
- syscall
- tbl
- intr
- tbl
- Page Tables
- VM system
- scheduler
- process & thread mgmt
- host file
- window
- nw intf

“virtual hardware”
VM Guest OS Processes

- Host OS schedules VMM threads
- VMM switches to shadow page table for guest OS’s current process
- Resumes Guest OS Thread & Process through return-from-interrupt
- Guest OS process runs with composite translation: Guest VA => VMMX VA => PA
  - Behaves as if Guest OS page table was in use
  - Guest OS region inaccessible to its user processes (protected)
- What happens if there is an interrupt?
- How does it make a SYSCALL
- What about the Guest OS?
  - It is not trusted by Host OS, Unprivileged!
Interrupts while in VM ???

User Software
- appln process
  - host file
  - window
  - nw intf

VMM-X

System Software
- scheduler
- process & thread mgmt
- VM system
- Page Tables

Guest OS
- syscall tbl
- intr tbl
- syscall handlers
- intrpt handlers
- file systems
- Drivers

"virtual hardware"

Hardware
- Processor
- Memory
- I/O Devices
- regs
- int:
- ptbr:
Interrupts

• Most interrupts have to do with hardware devices and need to be handled by Host OS
  – Regardless of whether Host process, Guest OS process or Guest OS is running

• But when Guest processes fault/trap it needs to be handled by the Guest OS
  – SYSCALL, Divide by zero, …
  – Page Fault ???

• Virtual Machine Monitor needs to be involved in Interrupt handling when VM is “running”
VMM: Interrupt Handling

Unprivileged Instructions

Privileged Instructions
Guest Process SYSCALL int

User Software
- appln process
  - host file
  - window
  - nw intf

System Software
- scheduler
- process & thread mgmt
- VM system
  - Page Tables
- syscall tbl

Hardware
- Processor
  - regs
  - int:
- Memory
- I/O Devices

Guest OS
- syscall handlers
- intrpt handlers
- file systems
- Drivers

"virtual hardware"

Unprivileged Instructions

Privileged Instructions

VMM
- v. PTs
- v. intr tbl

VMM-X
- VMM
- VMM-X

10/22/19
UCB CS162 Fa19 L15
Guest Process Syscall

• Traps to VMM
• VMM can dispatch the exception to the Guest OS syscall hander (through its interrupt vector), which will process args, dispatch to particular system call, etc.
• The Guest OS return-from-interrupt resumes the VMM
• The VMM can then resume the Guest OS process with the syscall result

• VMM serves as intermediary between the Guest Processes and the Guest OS
Guest Process SYSCALL int return

Hardware
- Processor
- Memory
- I/O Devices

System Software
- Scheduler
- Process & thread mgmt
- VM system
- Page Tables

User Software
- appln process
- window
- nw intf

VMM-X
- host file

Guest OS
- syscall tbl
- intr tbl
- intrpt handlers
- syscall handlers
- file systems
- Drivers
- "virtual hardware"

Unprivileged Instructions

Privileged Instructions

VMM
- v_PTs
- v_intr tbl

appln process
- regs

Guest Process SYSCALL int return
Host Interrupt during VM

Unprivileged Instructions

Privileged Instructions
VMM Interrupt Processing

• During VM
  • VMM catches interrupts and either redirects them back to the Host OS or up to the Guest OS
  • Otherwise, the Host OS receives interrupts
  • Those that involve the VMM will be delivered to it like other drivers

• But, Guest OS is not privileged
  – It is running at user level (some VMMs run it at PL=1)
• What happens when it executes privileged instructions?
  – Access to kernel region? Disable/Enable Interrupts? …
Virtualizable Instruction Set Arch.

• An instruction set is *virtualizable* if all “sensitive” instructions cause a trap if executed in unprivileged mode...

• x86 ISAs (till last several years) were not virtualizable in a strict sense and hugely complex
  – See VMware paper for incredible work arounds to get effective virtualizations
  – Recent generations of x86 have improved support for virtualization

• We’ll assume all virtualization sensitive actions by the Guest OS cause a trap to the VMM
  – Access to kernel region, Update PTBR, CLI/STI, …
  – Change its page tables, …
Host Interrupt during VM

User Software
- appln process
- host file
- window
- nw intf

Hardware
- Processor
- Memory
- I/O Devices

System Software
- scheduler
- process & thread mgmt
- VM system
- Page Tables
- syscall tbl
- intr tbl
- intrpt handlers
- syscall handlers

System Software (Guest OS)
- syscall process
- syscall process
- regs
- syscall handlers
- file systems
- Drivers

“virtual hardware”

VMM
- VMM-X
- Trap & Emulate
- v_PTs
- v_inr tbl
- Drivers

Unprivileged Instructions

Privileged Instructions

10/22/19
Trap and Emulate

• When Guest OS tries to access it kernel region it traps to VMM
  – Updates to page tables, scheduling threads, switching processes, interrupts,
• The VMM decodes what Guest OS is doing (basic blocks of multiple instructions) and emulates them, updating the Guest OS data structures as if it had done it itself.
• The VMM “sees” everything the Guest OS tries to do and can take action
  – If Guest OS updates a PTE, the VMM updates the shadow PTE
  – If it switches PTs, the VMM switches shadows
  – When it irets to process thread, VMM gives that process the right shadow
Guest OS Drivers

• Drivers interact with “their device” through Programmed I/O (PIO), Direct Memory Access (DMA), and Interrupts

• Device has a set of “registers” that are configured to appear at specific physical addresses
  – e.g, Read Data, Write Data, Operation, Status, Address
  – Mapped into kernel region of virtual address space

• Driver reads/writes these to access device
  – DMA allows blocks of data to be written to / read from the device

• All actions Guest OS drivers take on virtual I/O devices cause traps to the VMM
  – It then emulates these operations
Host Interrupt during VM

Unprivileged Instructions

Privileged Instructions

VMM-X

appln process

host file

window

nw intf

Guest OS

“virtual hardware”

file systems

drivers

I/O Devices

RPC

trap

Unpriviledged
Instructions

VMM

Trap & Emulate

appln process

regs

scheduler

process & thread mgmt

VM system

Page Tables

s

calls

intr tbl

intrpt handlers

syscall handlers

Drivers

Privileged
Instructions

Scheduler

Process & Thread mgmt

VM System

Page Tables

scall tbl

syscall handlers

Drivers

Unprivileged
Instructions

Scheduler

Process & Thread mgmt

VM System

Page Tables

syscall tbl

syscall handlers

Drivers

Unprivileged
Instructions

Processor

I/O Devices

Memory

Host Interrupt during VM

Hardware

System Software

User Software

appln process

host file

window

nw intf

appln process

regs

ptbr

int:

host file

window

nw intf

Guest OS

“virtual hardware”

file systems

Drivers

Unpriviledged
Instructions

VMM

Trap & Emulate

appln process

regs

scheduler

process & thread mgmt

VM system

Page Tables

s

calls

intr tbl

intrpt handlers

syscall handlers

Drivers

Privileged
Instructions

Scheduler

Process & Thread mgmt

VM System

Page Tables

scall tbl

syscall handlers

Drivers

Unprivileged
Instructions

Scheduler

Process & Thread mgmt

VM System

Page Tables

syscall tbl

syscall handlers

Drivers

Unprivileged
Instructions

Processor

I/O Devices

Memory

10/22/19

UCB CS162 Fa19 L15
Other forms of Virtual Machines

- We have described Hosted Virtual Machines
  - VMware fusion, Virtual Box, KVM, ...

- Para-virtualization: Guest OS is modified to work collaboratively with the Host VMM
  - VMM presents simplified machine to Guest OS

- Hypervisor: VMM resides under all the “Guest” OS – peers, none is the special host
  - Much cleaner relationship between VMM and OS’s

![Diagram of virtual machine architecture with hypervisor and multiple OS layers]
Stepping Back

• In creating a virtual machine we configured a set of resources within which all of its activities – its OS and all its processes – would operate
  – Total amount of physical memory (the MMAP)
  – Total size of all its disk storage (the file backing its disk)
  – Total network bandwidth

• These constraints are valuable even without a distinct (possibly heterogenous) Guest OS

• Modern operating systems provide performance isolation, in addition to traditional protection isolation
  – Without the additional machinery and capability of VMs
Performance Isolation: CGroups

Example:
- "production"
- "testing"
- "dev"

Hardware
- Processor
- Memory
- Net BW
- Files

System Software
- Page Tables
- schedulers
- intr tbl
- syscall tbl
- syscall handlers
- intrpt handlers

User Software
- Drivers
- subsystems

"production"
- System Software
- User Software

"testing"
- System Software
- User Software

"dev"
- System Software
- User Software

Example:
CGroups

• Identify collections of processes that will be treated as a *group* for resource allocation
  – Groups can have hierarchical structure
  – i.e., a Group can be comprised of subgroups
  – Process parent-child relationship defines a hierarchy
    » Generalize this beyond `fork()`

• Set of key resource dimensions
  – Processor share (CPU), CPU set (bound to particular cores)
  – Physical memory share,
  – block IO, net priority, net class
  – Namespace (ns), i.e., containers

• Resource limiting, prioritization, accounting and control

• Containers define a collection of libraries and executables that should be a group
How is all the cgroups info recorded?

• Unix-based systems use `/proc` to represent information about processes
  – `/proc/<pid>` describes process `<pid>`

• `/proc/cgroups/*`
  – Describes what controllers are implemented

• Each cgroup controller has a directory under `/sys/fs/cgroup/⟨controller⟩`
  – `/sys/fs/cgroup/cpu/production`
  – `/sys/fs/cgroup/memory/foo/memory.limit_in_bytes`

• Kernel monitors and controls processes/threads in accordance with cgroup controllers
Summary

• For 50+ years operating systems focused on virtualizing resources to provide clean, protected services to user processes
  – Illusion of near infinite resources, despite constraints
  – Protection and isolation, Glue (accounting was there too)

• Past 15+ years, with ever more resources, focus on *predictable* share (proportions) of resources (as part of virtualization)
  – Predictability: Service level agreements focus on 95\textsuperscript{th} or 99\textsuperscript{th} percentile
  – Metering and monitoring to ”groups” of processes, as defined by system administrative goals (also CFS, …)

• Virtual machines bring the illusion down to the lowest level and provide an extreme form of resource partitioning
  – Re-purpose the mechanisms used for protection, isolation and system extension (drivers) to permit near-perfect emulation
  – Intricate interplay of (in kernel) Virtual Machine Monitor and (user level) VMM-X process to allow Guest OS as if on Physical Machine and Guest OS User Processes as if on Guest OS
    » VMM interposes between hardware/Host OS and Guest OS (Page Table & Interrupt Tbl)

• Shadow Page Table (composition of two PTs) to translate Guest OS process VAS => MMAP region => Physical

• Trap and Emulate

• Control Groups & Containers provide resource limiting w/o VM
If you want to dig further

  – Extremely detailed look at what it takes to pull off VM in older x86
  – Good set of references
  – Newer systems have better hardware support for virtualization (e.g., KVM)

• **Xen and the Art of Virtualization,** 2003 (paravirtualization)

• **Disco: Running Commodity Operating Systems on Scalable Multiprocessors** (1997) – revived concept of VMMs largely lost since the 70s