CS 3204
Operating Systems

Lecture 27

Godmar Back

Announcements

- Project 4 due Dec 10
- Ribbon-Cutting Ceremony/Demo for extra credit
  - Dec 11, Reading Day: 1:30pm-3:00pm
  - McB 124
- Final Exam Announcement
- Tuesday:
  - wrap-up
  - Quiz (voluntary)
  - Demo preparation
  - Q&A for final exam
  - Teaching evaluations

Virtualization

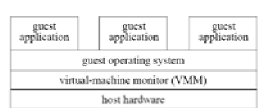
Definitions

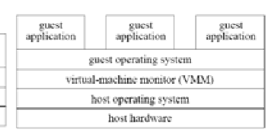
- Terms is somewhat ill-defined, generally
  - A machine that's implemented in software, rather than hardware
  - A self-contained environment that acts like a computer
  - An abstract specification for a computing device (instruction set, etc.)
- Common distinction:
  - (language-based) virtual machines
    - Instruction set usually does not resemble any existing architecture
    - Java VM, .Net CLR, many others
  - virtual machine monitors (VMM)
    - instruction set fully or partially taken from a real architecture

Use of Virtual Machines

- Test applications
- Program / debug OS
- Simulate networks
- Isolate applications
- Monitor for intrusions
- Inject faults
- Resource sharing/hosting

Types of Virtual Machines





- Type I
- Type II

VMM Classification

Paravirtualized guest drivers

Unmodified Guest ↔ Ported Guest

	Guest OS sees true hardware interface	Guest OS sees (almost) hardware interface, has some awareness of virtualization	Guest OS sees virtualized hardware interface	
Hypervisor runs directly on host hardware	VMware ESX MS Virtual Server	Xen Windows 7		Type I
Hypervisor runs on host OS	qemu, VMware Workstation, VMware GSX		UML	Type II

Kernel Support for VMM: skas3, UMLinux, vmware.ko

Virtualizing the CPU

- Basic mode: direct execution
- Requires *Deprivileging*
  - (Code designed to run in supervisor mode will be run in user mode)
- Hardware vs. Software Virtualization
  - Hardware: "trap-and-emulate"
    - Not possible on x86 prior to introduction of Intel/VT & AMD/Pacifica
  - Software:
    - Either require cooperation of guests to not rely on traps for safe deprivileging
    - Or binary translation to avoid running unmodified guest OS code (note: guest user code is always safe to run!)

Binary translation vs trap-and-emulate

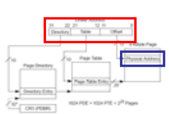
- Adams [ASPLOS 2006] asked:
  - How fast is binary translation?
  - Is it always slower than trap-and-emulate?
- Surprising result: binary translation usually beats trap-and-emulate. Why?
  - Binary translation is highly optimized:
    - most instructions are translated as IDENT (identical), preserving most compiler optimizations and only slightly increasing code size
    - binary translation can be *adaptive*: if you know an instruction is going to trap, inline part of all of trap handler. Way cheaper than actually trapping.

Virtualizing MMU

- Guest OS programs page table mapping virtual -> physical
- Hypervisor must *trace* guest's page tables, apply additional step from physical -> hardware
- Shadow page tables: hypervisor makes a copy of page table, installs copy in MMU
  - This approach is used both in ESX & full virtualization via Intel/VT
- Xen paravirtualization: guest's page table are directly rewritten to map virtual -> hardware

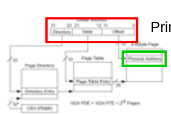
MMU Paravirtualization

- Paravirtualized MMU
- Shadow Page Table

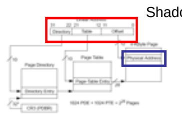


Virtual → Physical → Hardware

Primary



Shadow

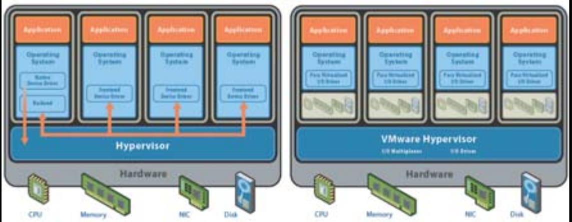


How much do shadow page tables hurt?

- Recall: a primary argument for paravirtualization was avoiding shadow page tables
- Turns out that shadow page tables can be implemented very efficiently
  - They are created on demand (only if guest code actually faults), and only needed translation range is created (e.g., single 2nd level page table in 32bit model)
  - Cost of tracing updates by guest is minimized via adaptive binary translation
- In practice, seems to be a non-issue!

Virtualizing I/O

- Xen
- ESX



Source: VMware white paper on virtualization considerations.

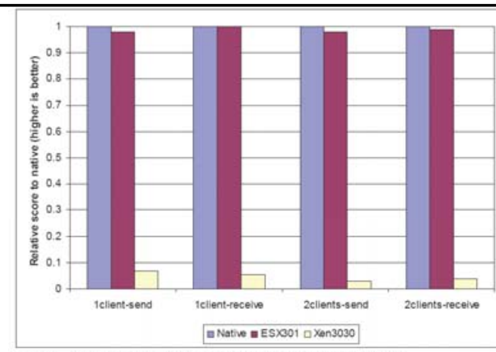


Figure 7 — Netperf results compared to native (higher values are better)

Source: VMware paper on hypervisor performance

Performance Impact of I/O Virtualization

- ESX mainly outperforms Xen because
  - Costs of CPU & MMU virtualization are (relatively small)
  - It uses native drivers in hypervisor (like Xen 1.0 did, really)
    - Hardware vendors port their Linux drivers to Xen
  - Thus avoids inter-domain communication
  - Caveat: Xen is being continuously improved (previous slide is 3.0.* version); I/O performance still remains challenging
- Note: guest drivers are simple, and can be paravirtualized
  - Most OS have an interface for 3rd party drivers; but no interface to have core modules (e.g. memory management) replaced!

Memory Management in ESX

- Have so far discussed how VMM achieves isolation
  - By ensuring proper *translation*
- But VMM must also make resource management decisions:
  - Which guest gets to use which memory, and for how long
- Challenges:
  - OS generally not (yet) designed to have (physical memory) taken out/put in.
  - Assume (more or less contiguous) physical memory starting at 0
  - Assume they can always use all physical memory at no cost (for file caching, etc.)
  - Unaware that they may share actual machine with other guests
  - Already perform page replacement for their processes based on these assumptions

Goals for memory management

- Performance
  - Is key. Recall that
    - avg access = hit rate * hit latency + miss rate * miss penalty
    - Miss penalty is *huge* for virtual memory
- Overcommiting
  - Want to announce more physical memory to guests that is present, in sum
  - Needs a page replacement policy
- Sharing
  - If guests are running the same code/OS, or process the same data, keep one copy and use copy-on-write

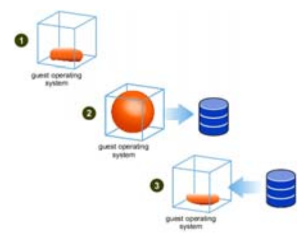
Page Replacement

- Must be able to swap guest pages to disk
  - Question is: which one?
  - VMM has little knowledge about what's going on inside guest. For instance, it doesn't know about guest's internal LRU lists (e.g., Linux page cache)
- Potential problem: Double Paging
  - VMM swaps page out (maybe based on hardware access bit)
  - Guest (observing the same fact) – also wants to "swap it out" – then VMM must bring in the page from disk just so guest can write it out
- Need a better solution

Ballooning

- What if we could trick guest into reducing its memory footprint?
- Download balloon driver into guest kernel
  - Balloon driver allocates pages, possibly triggering guest's replacement policies.
  - Balloon driver pins page (as far as guest is concerned) and (secretly to guest) tells VMM that it can use that memory for other guests
  - Deflating the balloon increases guest's free page pool
- Relies on existing memory in-kernel allocators (e.g., Linux's get_free_page()
- If not enough memory is freed up by ballooning, do random page replacement

Ballooning

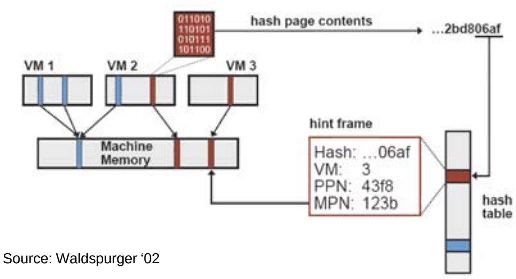


Source: VMware

Sharing Memory

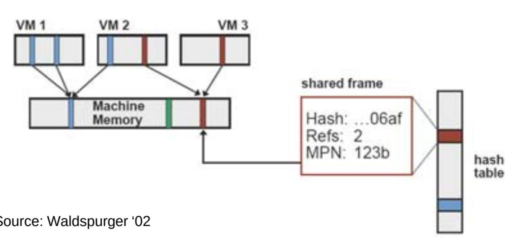
- Content-based sharing memory
- Idea: scan pages, compute a hash. If hash is different, page content is different. If hash matches, compare content.
- If match, map COW
- Aside: most frequently shared page is the all-zero page (why?)

Page Sharing (1)



Source: Waldspurger '02

Page Sharing (2)



Source: Waldspurger '02

Sharing Efficiency

- Ideal conditions: (all guests equal, run same workload) – up to 60% savings
- Realistic conditions (measured in production system)

		Total	Saved	
Workload	Guest Types	MB	MB	%
Corporate IT	10 Windows	2048	673	32.9
Nonprofit Org	9 Linux	1846	345	18.7
VMware	5 Linux	1658	120	7.2

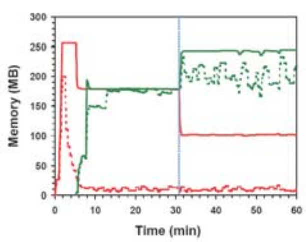
Allocation

- How should machine memory be divvied up among guest?
- Observation: not all are equally important
- (Traditional OS approach of maximizing system-wide utility – as, for instance, global replacement algorithm would do - is not applicable)
- Use share-based approach
  - Graceful degradation under overload
  - Work conserving under underload

Proportional Sharing of Memory

- Could simple proportional split be applied?
  - As is done in CPU algorithms (VTRR, etc.)?
- Answer appears to be no:
  - It doesn't take into account if memory is actually used (that is, accessed) by clients
- Idea: develop scheme that takes access into account
  - Tax idle memory higher (a "progressive" task on unused, in a way)
- Determine degree of idleness by sampling

Idle Memory Tax: 75%



- Experiment
  - 2 VMs, 256 MB, same shares
  - VM1: Windows boot+idle
  - VM2: Linux boot+dbench
  - Solid: usage, Dotted: active
- Change tax rate
- After: high tax
  - Redistribute VM1 → VM2
  - VM1 reduced to min size
  - VM2 throughput improves 30%

Source: Waldspurger '02