Instruction caching for bhyve

Mihai Carabas, Neel Natu
{mihai,neel}@freebsd.org

AsiaBSDCon 2015
Tokyo University of Science
Tokyo, Japan
March 12 – 15, 2015

Who we are?

- Mihai Carabas
  - PhD Student and Teaching Assistant at the University POLITEHNICA of Bucharest, Romania
  - DragonFly BSD (SMT aware scheduler - 2012 / Intel EPT for vkernels - 2013)
  - FreeBSD - bhyve (instruction caching - 2014 / coordinating students in bhyve projects - current)

Who we are?

- Mihai Carabas
  - PhD Student and Teaching Assistant at the University POLITEHNICA of Bucharest, Romania
  - DragonFly BSD (SMT aware scheduler - 2012 / Intel EPT for vkernels - 2013)
  - FreeBSD - bhyve (instruction caching - 2014 / coordinating students in bhyve projects - current)
- Neel Natu
  - principal contributor for the bhyve project (together with Peter Grehan)
  - started as a FreeBSD/mips committer

Context

- Hardware Assisted Virtualization
  - a new CPU privilege level
  - memory virtualization (EPT / NPT)

Context

- Hardware Assisted Virtualization
  - a new CPU privilege level
  - memory virtualization (EPT / NPT)
- What about controlling the APIC from the VM?
  - each control register access traps in the hypervisor
  - the hypervisor needs to emulate that access

Steps for handling a trap in the hypervisor

- Fetch the instruction
  - manually walking the Guest OS page table to find the physical address
  - map the address in the hypervisor address space and copy the instruction

Steps for handling a trap in the hypervisor

- Fetch the instruction
  - manually walking the Guest OS page table to find the physical address
  - map the address in the hypervisor address space and copy the instruction
- Decode the instruction
  - variable length instructions for x86 platforms

Steps for handling a trap in the hypervisor

- Fetch the instruction
  - manually walking the Guest OS page table to find the physical address
  - map the address in the hypervisor address space and copy the instruction
- Decode the instruction
  - variable length instructions for x86 platforms
- Emulate the instruction
  - execute the instruction in the name of the VM

Steps for handling a trap in the hypervisor

- Fetch the instruction
  - manually walking the Guest OS page table to find the physical address
  - map the address in the hypervisor address space and copy the instruction
- Decode the instruction
  - variable length instructions for x86 platforms
- Emulate the instruction
  - execute the instruction in the name of the VM
- Any solution to jump over some of them?

Identify an instruction for caching

- Cached object: `struct vie`
- Unique identifier (key)

Identify an instruction for caching

- Cached object: `struct vie`
- Unique identifier (key)
  - VM ID: `struct vm *`
  - instruction address (RIP)
  - pointer to the page table (CR3)
- Stored in `struct vie_cached`

Integrating caching mechanism in the emulation code

- New interface provided by `vmm_instruction_cache.h`

Integrating caching mechanism in the emulation code

- New interface provided by `vmm_instruction_cache.h`
- `vm_inst_cache_add`
  - adds the instruction to the cache
  - mark as read-only the pages related to the instruction

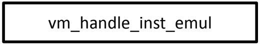
Integrating caching mechanism in the emulation code

- New interface provided by `vmm_instruction_cache.h`
- `vm_inst_cache_add`
  - adds the instruction to the cache
  - mark as read-only the pages related to the instruction
- `vm_inst_cache_delete`
  - removes an instruction from cache
  - solves the write page fault

Integrating caching mechanism in the emulation code

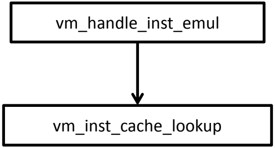
- New interface provided by `vmm_instruction_cache.h`
- `vm_inst_cache_add`
  - adds the instruction to the cache
  - mark as read-only the pages related to the instruction
- `vm_inst_cache_delete`
  - removes an instruction from cache
  - solves the write page fault
- `vm_inst_cache_lookup`

Caching flow



vm_handle_inst_emul

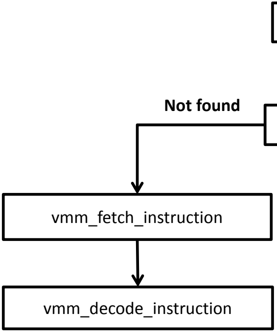
Caching flow



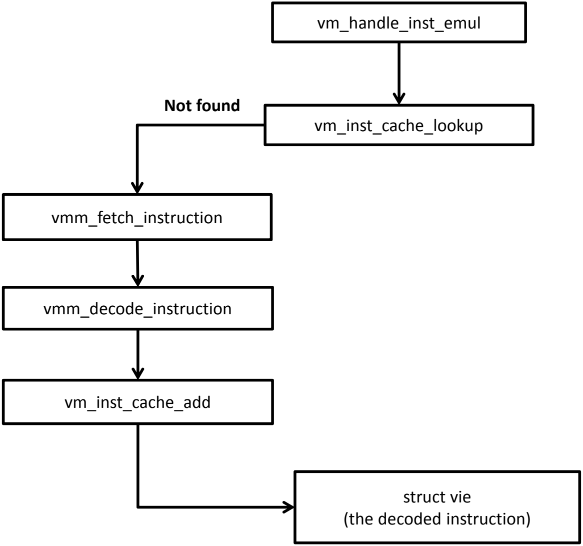
vm_handle_inst_emul

vm_inst_cache_lookup

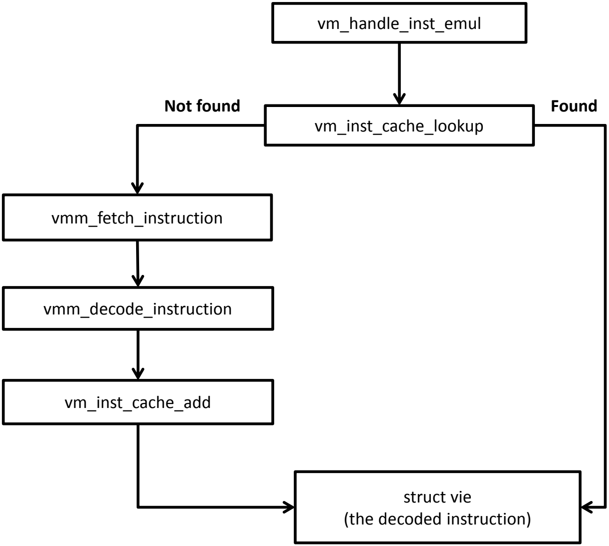
Caching flow



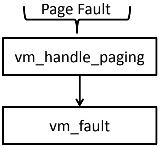
Caching flow



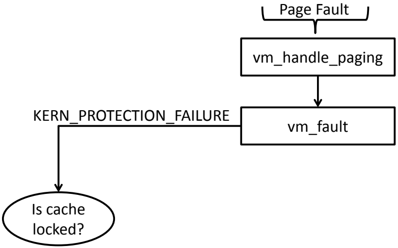
Caching flow



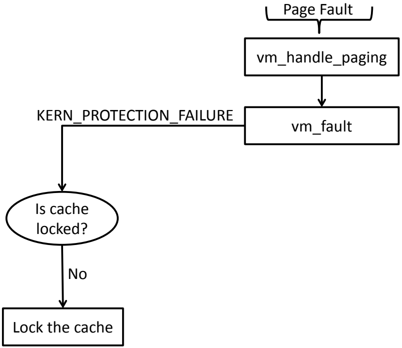
Cache invalidation flow



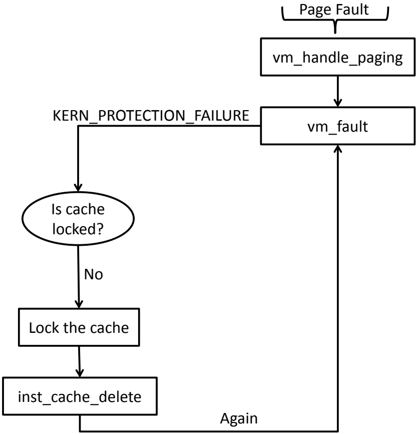
Cache invalidation flow



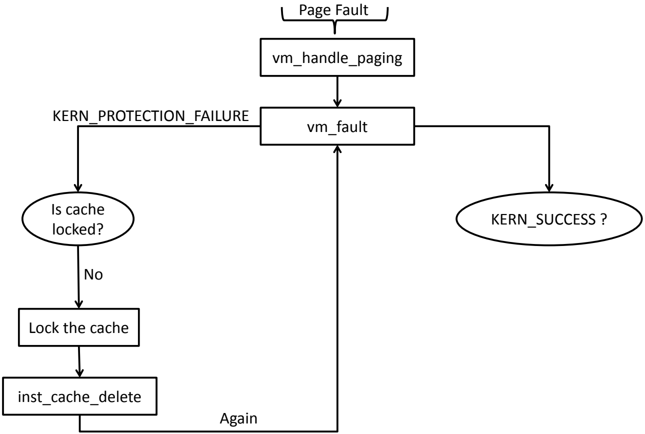
Cache invalidation flow



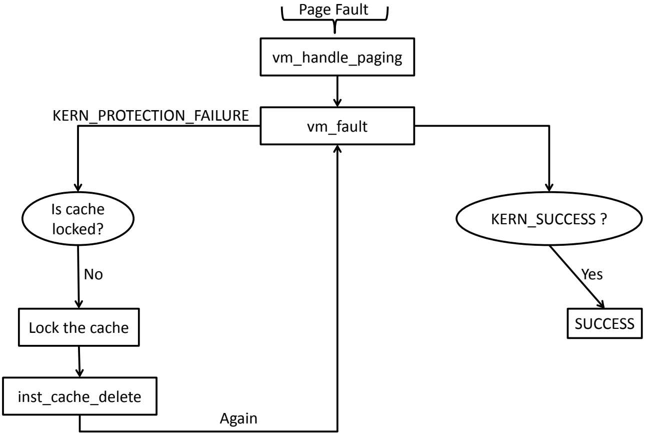
Cache invalidation flow



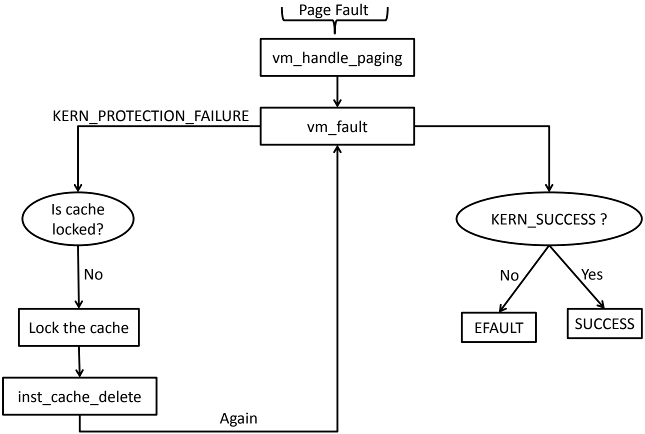
Cache invalidation flow



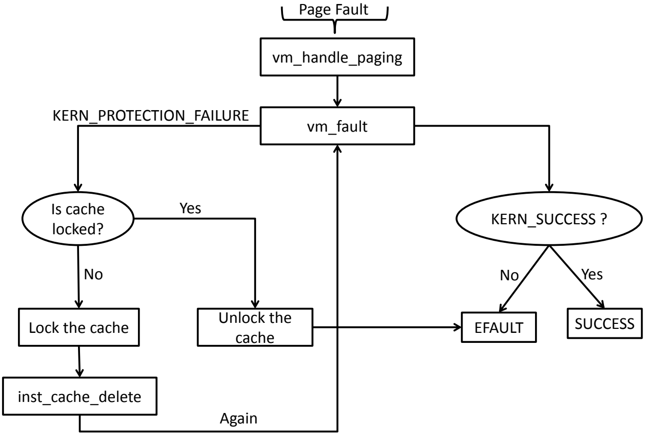
Cache invalidation flow



Cache invalidation flow



Cache invalidation flow



Efficiency evaluation

- Micro-benchmarking
  - kernel module accessing the LAPIC ID in a tight loop
  - measure the average access time
  - 10500 ticks without instruction caching
  - 6700 ticks with it (30% improvement)

Efficiency evaluation

- Micro-benchmarking
  - kernel module accessing the LAPIC ID in a tight loop
  - measure the average access time
  - 10500 ticks without instruction caching
  - 6700 ticks with it (30% improvement)
- Real world workloads
  - simple loop running in user space and `make buildworld` in VM
  - measure the time that needs to finish the workload (`time` command)
  - measure the cache efficiency (hits, misses) (`VMM_STAT_*` custom counters)

Real world cache efficiency

Table: CPU intensive bash script

Number of instruction cache	vCPU0	vCPU1
hits	699.519	840,485
insertions	10.395	5,743
evictions[0]	7.139	8.926
evictions[1]	0	0
evictions[2]	0	0
evictions[3]	0	0

Table: make buildworld -j2

Number of instruction cache	vCPU0	vCPU1
hits	19.204.630	12.930.500
insertions	8.688.733	9.051.295
evictions[0]	8.563.694	9.173.381
evictions[1]	1.131	1.457
evictions[2]	0	0
evictions[3]	0	0

Speed-up for running time

Table: CPU intensive bash script

hw.vmm.instruction_cache	time spent in execution (s)
1	225
0	230

Table: make buildworld -j2

hw.vmm.instruction_cache	time spent in execution (s)
1	13900
0	13938

Related work

- KVM driver isn't using any caching technique
- there exists something in the fetch part (pre-fetch the instructions bytes in advanced)

Related work

- KVM driver isn't using any caching technique
- there exists something in the fetch part (pre-fetch the instructions bytes in advanced)
- KVM community opinion as stated in a KVM-Intel presentation from 2012
  - they want to rely on the hardware only
  - all the interrupt handling in hardware (virtualize the APIC without VM exists)
  - a VM exit is too expensive

Related work

- KVM driver isn't using any caching technique
- there exists something in the fetch part (pre-fetch the instructions bytes in advanced)
- KVM community opinion as stated in a KVM-Intel presentation from 2012
  - they want to rely on the hardware only
  - all the interrupt handling in hardware (virtualize the APIC without VM exists)
  - a VM exit is too expensive
- instruction emulation will still be used for other devices models (e.g. HPET, AHCI)

Conclusions

- Cache the emulated instructions in order to decrease the time spent in the hypervisor
- Handled corner cases like contention on the VM page table without using a big lock
- Theoretical good results (e.g. 30% improvement of the average access time)
- Didn't find a real world workload to benefit from this mechanism

Thank you for your attention!

ask questions