Private vs. Common
Reflections on Cross-Architecture Commonality

Christian Bornträger
Maintainer KVM on IBM Z
borntraeger@de.ibm.com
IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries.

The following are trademarks or registered trademarks of other companies.
• Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
• Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.
Motivation

Maintainer for s390 KVM (IBM Z)
Mostly focused on that platform
Also try to keep up2date with kvm in general
I have seen many opportunities for improvement regarding cross-architecture work
Often challenging due to the details
Some history and some examples
Outlook and Future
The following patchset adds a driver for Intel’s hardware virtualization extensions to the x86 architecture. The driver adds a character device (/dev/kvm) that exposes the virtualization capabilities to userspace. Using this driver, a process can run a virtual machine (a "guest") in a fully virtualized PC containing its own virtual hard disks, network adapters, and display.

Using this driver, one can start multiple virtual machines on a host. Each virtual machine is a process on the host; a virtual cpu is a thread in that process. kill(1), nice(1), top(1) work as expected.

In effect, the driver adds a third execution mode to the existing two: we now have kernel mode, user mode, and guest mode. Guest mode has its own address space mapping guest physical memory (which is accessible to user mode by mmap()ing /dev/kvm). Guest mode has no access to any I/O devices; any such access is intercepted and directed to user mode for emulation.

The driver supports i386 and x86_64 hosts and guests. All combinations are allowed except x86_64 guest on i386 host. For i386 guests and hosts, both pae and non-pae paging modes are supported.

SMP hosts and UP guests are supported. At the moment only Intel hardware is supported, but AMD virtualization support is being worked on.

Performance currently is non-stellar due to the naive implementation of the mmu virtualization, which throws away most of the shadow page table entries every context switch. We plan to address this in two ways:

- cache shadow page tables across page faults
- wait until AMD and Intel release processors with nested page tables
Enabling Other Architectures

Refactoring of code end of 2007
Intel: IA64
IBM: POWER, s390

$ git log -p --grep "KVM: Portability" | diffstat -p1
    drivers/kvm/i8259.c              |    1
    drivers/kvm/ioapic.c             |   16
    drivers/kvm/iodev.h              |   63
    drivers/kvm/irq.c                |    1
    drivers/kvm/irq.h                |   32
    drivers/kvm/kvm.h                |  771 +--------
    drivers/kvm/kvm_main.c           | 2632 +-------------------------------
    drivers/kvm/lapic.c              |   97 -
    drivers/kvm/mmu.c                |  202 +-
    drivers/kvm/mmu.h                 |   46
    drivers/kvm/paging_tmpl.h        |   16
    drivers/kvm/segment_descriptor.h |   12
    drivers/kvm/svm.c                |  129 -
    drivers/kvm/types.h               |   54
    drivers/kvm/vmx.c                |  231 +-!
    drivers/kvm/x86.c                 | 3202 +++++++++++++++++++++++++++++++++++----
    drivers/kvm/x86.h                 |  590 ++++++-
    drivers/kvm/x86_emulate.c         |   25
    include/asm-x86/kvm.h             |  170 ++
    include/linux/kvm.h               |  154 -
20 files changed, 4421 insertions(+), 4023 deletions(-)

$ git shortlog -s --no-merges --grep "KVM: Portability"
   8  Carsten Otte
   1  Christian Ehrhardt
   8  Hollis Blanchard
   7  Jerone Young
  29  Zhang Xiantao

20 files changed, 4421 insertions(+), 4023 deletions(-)
Comparing code size

Starting with Linux 2.6.26, kvm supports four different machine architectures: x86, s390 (System Z, or mainframes), ia64 (Intel's Itanium), and embedded PowerPC processors. It is interesting to compare the size of the code supporting each architecture:

<table>
<thead>
<tr>
<th>arch</th>
<th>lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86</td>
<td>17442</td>
</tr>
<tr>
<td>ia64</td>
<td>8154</td>
</tr>
<tr>
<td>s390</td>
<td>2509</td>
</tr>
<tr>
<td>ppc</td>
<td>2229</td>
</tr>
</tbody>
</table>

x86 is old and cruffy; it supports three instruction sets and four paging modes; its long and successful history means that it needs the most kvm support code. There are two different virtualization extensions that kvm supports on x86 (Intel's VT and AMD's SVM). It is also the architecture that has been supported by kvm for the longest time. It is no surprise that it leads the pack by a significant amount.

ia64 is a newer architecture, but a quite complex one. The mechanism by which it supports virtualization, with a module loaded into the host kernel and a second module loaded into the guest address space, also adds complexity. So it comes in second, though far behind x86.

s390 is older (and probably far crufter) than x86. But on the other hand, its hardware virtualization support is so mature and complete that a complete hypervisor fits in a fraction of the lines required for x86. Indeed, it will take a while until x86 can support 64-way guests.

ppc 44x, the embedded PowerPC variant targeted by kvm, has a simple software-managed tlb model, and the regular instruction set encoding favored by RISC processors, so it gets by with just a seventh of the amount of code required by x86.

As we add more features, kvm code size will continue to grow slowly, but the relative comparison will no doubt remain valid. And kvm will likely remain the smallest full virtualization solution available.

http://avikivity.blogspot.com/2008/05/comparing-code-size.html
Components
Structure of a KVM Installation

**KVM kernel module**
Closest to the hardware → does that mean least commonality?

**QEMU userspace**
Also close to hardware, but uses abstract system call interface → worse or better?

**Libvirt**
Far away from hardware → should be easy, no?

**Management stack beyond libvirt**
Is there anything platform specific at all?
QEMU vs Kernel

target/*/kvm* vs accel/kvm/
As expected: more commonality
Unexpected: similar size across architecture

libvirt
Not even arch-specific folders/files (apart from test)
KVM Kernel Module

Lines of Code (sloccount) 4.18
arch/arm64/kvm/    = 4,792
arch/arm/kvm/      = 2,458
arch/mips/kvm/     = 9,005
arch/s390/kvm/     = 10,984
arch/powerpc/kvm/  = 32,651
arch/x86/kvm/      = 41,439
virt/kvm/          = 16,327  (- virt/kvm/arm/ = 11,776) == 4551

arm 16.17%  mips 7.65%  powerpc 27.75%  s390 9.34%  x86 35.22%  kvm 3.87%

Move device assignment to x86
KVM Kernel Module

**What is common?**
- Basic IOCTLs
- VM lifecycle
- VCPU lifecycle
- Eventfd/IRQFD handling
- Debugfs
- `kvm_stat` base functions

**What is partially common?**
- RCU handling
- Wait state
- Cpu request handling
- Polling
- Time accounting
- Vfio kvm bridge

**What is arch specific?**
- Hardware setup
- Nested virtualization
- MMU emulation
- Instruction emulation
- `kvm_stat` counters
- Arch tracepoints
- Guest state interfaces
- Basically any hardware specific guest feature enablement

....
What Kind of Problems?

Kernel perspective
Halt Polling

Wait a bit when guest CPU is going idle
Lots of heuristics: Uses hysteresis to adapt, check for system load...
Does one size fits all?

Problem 1: Local and floating interrupts
If there is an interrupt, consider the poll successful
I/O interrupts on s390 are floating (pending for all CPU, first come first serve)
Polling was overly optimistic
Solution: Add arch callback to filter wakeups

Problem 2: SMT variants
Solution: Private implementation for power

commit 0cda69dd7cd64fdd54bdf584b5d6ba53767ba422
KVM: PPC: Book3S HV: Implement halt polling

This patch introduces new halt polling functionality into the kvm_hv kernel module. When a vcore is idle it will poll for some period of time before scheduling itself out.

[...]
There exists generic halt_polling code in virt/kvm_main.c, however on powerpc the polling conditions are different to the generic case. It would be nice if we could just implement an arch specific kvm_check_block() function, but there is still other arch specific things which need to be done for kvm_hv (for example manipulating vcore states) which means that a separate implementation is the best option.
So what about cross architecture review?
Reviews

Reviews are often inside the silo

```
# git log arch/arm*/kvm/ | grep Reviewed | sort | uniq -c | sort -n -r | head -n 5
158 Reviewed-by: Christoffer Dall <christoffer.dall@linaro.org>
85  Reviewed-by: Marc Zyngier <marc.zyngier@arm.com>
48  Reviewed-by: Andrew Jones <drjones@redhat.com>
29  Reviewed-by: Will Deacon <will.deacon@arm.com>
27  Reviewed-by: Christoffer Dall <cdall@linaro.org>

# git log arch/s390/kvm/ | grep Reviewed | sort | uniq -c | sort -n -r | head -n 5
116 Reviewed-by: Cornelia Huck <cornelia.huck@de.ibm.com>
115 Reviewed-by: Christian Borntraeger <borntraeger@de.ibm.com>
85  Reviewed-by: David Hildenbrand <dahi@linux.vnet.ibm.com>
66  Reviewed-by: Cornelia Huck <cohuck@redhat.com>
50  Reviewed-by: David Hildenbrand <david@redhat.com>

# git log arch/powerpc/kvm/ | grep Reviewed | sort | uniq -c | sort -n -r | head -n 5
43  Reviewed-by: David Gibson <david@gibson.dropbear.id.au>
9   Reviewed-by: Thomas Huth <thuth@redhat.com>
9   Reviewed-by: Alexander Graf <agraf@suse.de>
8   Reviewed-by: David Hildenbrand <david@redhat.com>
8   Reviewed-by: Cornelia Huck <cohuck@redhat.com>

# git log arch/mips/kvm/ | grep Reviewed | sort | uniq -c | sort -n -r | head -n 5
13  Reviewed-by: James Hogan <james.hogan@imgtec.com>
4   Reviewed-by: David Hildenbrand <david@redhat.com>
4   Reviewed-by: Cornelia Huck <cohuck@redhat.com>
3   Reviewed-by: Radim Krcmar <rkrcmar@redhat.com>
2   Reviewed-by: Paolo Bonzini <pbonzini@redhat.com>
```

Redhat as the exception (and overall KVM maintainer)
More Cross-Team Education and Review

There have been several talks about educating about architecture specifics
Nested Virtualization on ARM - Christoffer Dall - KVM Forum 2017
To EL2, and Beyond! - Christoffer Dall – KVM Forum 2017
Nesting KVM on s390x - David Hildenbrand – KVM Forum 2016
KVM on System z: The Good, the Bad and the Weird - Cornelia Huck – KVM Forum 2016
QEMU Hotplug Infrastructure and Implementing PCI Hotplug for PowerKVM - Michael Roth – KVM Forum 2015
KVM on IBM POWER8 Machines - Paul Mackerras – KVM Forum 2014

[...
There are more talks this time
Arm Timers; and Fire! - Christoffer Dall
Secure Virtual Machines on Power - Ram Pai & Guerney Hunt
Protect Data of Virtual Machines with Memory Encryption on KVM - Kai Huang
s390 KVM Memory Management and its Pitfalls - Janosch Frank

Do we need more discussion across architecture teams?
Each architecture has its own pile of oddities
Most HW company have some non-yet-public in-house developments
BOF Session proposal
(see https://www.linux-kvm.org/page/KVM_Forum_2018_BOF)
Successful Changes
Linux systems have the notion of system time, user time, idle time, softirq time etc
For KVM a guest time was added
Support available in sysstat and others

<table>
<thead>
<tr>
<th>Average:</th>
<th>CPU</th>
<th>%usr</th>
<th>%nice</th>
<th>%sys</th>
<th>%iowait</th>
<th>%steal</th>
<th>%irq</th>
<th>%soft</th>
<th>%guest</th>
<th>%gnice</th>
<th>%idle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average:</td>
<td>all</td>
<td>0.06</td>
<td>0.00</td>
<td>0.05</td>
<td>0.00</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>1.13</td>
<td>0.00</td>
<td>98.75</td>
</tr>
</tbody>
</table>

Generic callbacks that handle this across all architectures
Now wrapped in guest_enter/exit_irqoff
Does work for all variants of virt cpu accounting
Can we build upon this to influence and enhance others?
No support in top (procps-ng)

Many more successful common parts
Idea sharing
Are we good in sharing ideas?
Lazy FPU

Registers are often shared between guest and host
For example the hypervisor must reload the floating point registers on guest entry/exit

Does it?
The kernel does not use floating point (most of the time)

First stage
preempt notifier
Light-weight exits (e.g. a virtio notify bound to eventd) have no need of fp reloading

Second stage
On reschedule: save guesta, load hosta → schedule → save hosta, load hostb → schedule → save hostb, load hosta → save hosta, load guesta
Do the guest/host reloading in ioctl path instead
Now on reschedule: save guesta, load hostb → schedule → save hostb, load guesta

Last year I talked about that with Rik van Riel
See what happened afterwards

commit f775b13e6e2ef7f3c6fdd4e90fb79090ce5d339
  x86,kvm: move qemu/guest FPU switching out to vcpu_run

Signed-off-by: Rik van Riel <riel@redhat.com>
Suggested-by: Christian Borntraeger <borntraeger@de.ibm.com>
IBM Z had asynchronous page fault for a long time in z/VM
KVM on z tried to implement the same on KVM
Meanwhile KVM on x has implemented a similar scheme
Sometimes it needs review feedback to state the obvious

Implementation was not fully identical but we were able to factor out things
Protected by CONFIG_KVM_ASYNC_PF_SYNC
Interestingly enough there is no sharing in the guest handlers
Guest Page Hinting

**Tell the host, which guest pages are no longer necessary**
Implemented for IBM Z with z/VM since ~2005 as CMMA (collaborative memory management assist)
Available for KVM on Z since ~2014
Guest has callbacks in arch_alloc_page and arch_free_page
Callbacks use hardware instructions that sets special bits in the host page table

**Several possible optimization**
On swapout: discard unused pages
On swapin: discard swap slot and use empty zero page
During runtime: drop existing mappings to avoid page out

**For x86 under discussion for a while**
Similar scheme – guest page hinting
See Guest Free Page Hinting - Nitesh Narayan Lal, Red Hat, Inc., Friday, October 26 • 16:15 – 16:45
Any chance to collaborate? Are the implementations too different?
Interfaces
VIRTIO

Devices are platform specific
SATA, IDE, SCSI, DASD....
Must be platform specific

**Virtio defines fully virtual I/O interface**
This can clearly be made platform independent!
First implementation allowed anything (virtio over pigeon carriers)

**Unfortunately, no so:**
During SPECing things became more specific
A Windows driver was considered important
It was much easier to add a PCI driver than to add a new access method
So it was PCI

Now, **s390 had no PCI back then...**
S390 implemented an lguest like scheme (with queues above main memory)
Refactoring started
Later virtio-ccw was added → hotplug, hot unplug up to 256k devices etc.
Later virtio-mmio was added

**Does that cause problem?**
Virtio in Libvirt

Implementation start on one platform
Usually PCI
**Other platforms add thing later**
CCW, MMIO and previously S390
It becomes obvious that we can refactor

commit 8dcac770f1f5bd966968962de905ab37eb17488a
AuthorDate: Mon Feb 27 17:16:20 2012 +0800
  qemu: add virtio-scsi controller model

[...]

commit 4c1d1497e2b6c27be66bb58d9c44e6958c202a94
AuthorDate: Thu Mar 14 19:32:25 2013 +0100
  S390: Enable virtio-scsi and virtio-rng

[...]

commit 04eb7479fc0d9196e07f7db227deb3a8003e8964
AuthorDate: Wed Sep 5 18:24:55 2018 +0200
  qemu: Unify generation of command line for virtio devices
Future Development
Upcoming Guest User Interfaces

Every architecture has ways of exposing that you run a hypervisor

```
s390sys # cat /proc/sysinfo
VM00 Name: s38kvm21
VM00 Control Program: KVM/Linux
VM00 Adjustment: 1000
VM00 CPUs Total: 2
VM00 CPUs Configured: 2
VM00 CPUs Standby: 0
VM00 CPUs Reserved: 0
VM00 Extended Name: s38kvm210
VM00 UUID: ca9eb12e-5a01-4dcf-b952-32db704ec364
```

Dmidecode
etc

**New patch for /sys/hypervisor**

Commonality with XEN

X86 specific

Can be fixed to be cross-architecture

[PATCH] KVM: Start populating /sys/hypervisor with KVM entries

Start populating /sys/hypervisor with KVM entries when we're running on KVM. This is to replicate functionality that's available when we're running on Xen.

Let's start with /sys/hypervisor/uuid, which users prefer over /sys/devices/virtual/dmi/id/product_uuid as a way to recognize a virtual machine, since it's also available when running on Xen HVM and on Xen PV and, on top of that doesn't require root privileges by default.
Upcoming Guest User Interfaces

Read Only Enforcement patches V4 [VMM based kernel hardening]
X86 only
Any chance of making this generic?

A steady flow of such new interfaces
Summary

**Improve inter-architectural discussion**
That implies understanding what this is all about

**Do not assume that anything is x86**
Most people know ARM, but there are others: mips, POWER,s390

**Sometimes it even possible to share interfaces with other hypervisors**
XEN guest interfaces, z/VM interfaces, PowerVM?

**Sometimes sharing hurts and private implementations are better**
Try, but do not go too far

**Interested in doing something? See my BoF Session**
THANK YOU!
borntraeger@de.ibm.com
What else if I had more time?
Problems when doing common code
Over-Generalization

Sometimes people do the right thing and implement things across architecture
This can go wrong as well
People makes assumptions and base their implementation on those

Example
Postcopy transfers empty pages to the target and marks them as transferred
Second requests for that page are being ignored
Should not fault
Guest page hinting might now throw away this page
And then the guest reuses that page and accesses it
Fault!
The host will then resolve the fault.....
Userfaultd is responsible.....
Postcopy thinks: already transmitted – return and do nothing
Fault!

Everything was fixed by disabling the problematic cases during postcopy
Something to learn from both sides, sometimes assumptions break
Before enabling a feature on a platform
How do handle host paging and similar thing
Ideally just allow everything that a normal process has
Provide callbacks for page table primitives
mm gets a callback structure for primitives:
  ptep_test_and_clear_young → ptep_clear_young_notify
[...]
  ptep_test_and_clear_young
  mmu_notifier_clear_young
KVM will register for operations

S390 does not use mmu notifier
Storage keys as a per physical page entity
Update needs to be in sync for host and guest
Solution: not use mmu_notifier, do the guest part in the base architecture primitives
Other challenges: need to implement large page support separately
See “s390 KVM Memory Management and its Pitfalls - Janosch Frank, IBM” at this KVM Forum