Building a Better Userspace

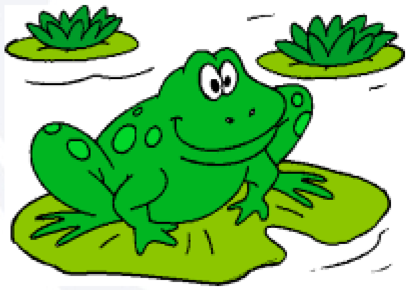
KVM Forum 2008

Anthony Liguori – aliguori@us.ibm.com
Open Virtualization
IBM Linux Technology Center

June 11th 2008

Reflecting on QEMU

We really owe a lot to QEMU



Reflecting on QEMU

- QEMU is a community-driven project
  - No company has sponsored major portions of it's development
- QEMU does a really amazing thing
  - Can emulate 9 target architectures on 13 host architectures
  - Provides full system emulation supporting ~200 distinct devices
- Is the basis of KVM, Xen HVM, and xVM Virtual Box
  - Every Open Source virtualization project uses QEMU
- QEMU receives very little credit for this

But all is not well

Many see QEMU as the last mile for open virtualization

QEMU has a patch problem

- Many patches are ignored
- Security fixes are not applied in a timely manner
- The quality of the patches that are committed is often questionable
- Some sub-systems are effectively unmaintained

How do we fix it?

Fixing the QEMU process

- First step
  - Patch review
  - IBM will be dedicating resources (me) to reviewing patches on qemu-devel
  - This is now happening (although I need help)
- Second step
  - I'll be seeking commit access for virtualization support in QEMU
    - Many patches will still be committed by other people according to subsystem maintainer
    - I'll try to review everything I can
- Third step
  - As things improve, we can hopefully move to a DSCM like git and start taking in pull requests

Patch Review

- We need people reviewing patches on qemu-devel
  - Make use of Reviewed-by, Tested-by, Acked-by
- It's arguable that more committers are needed
  - QEMU has about 9 as it is; KVM just has 1
- We need to jump start a culture of review

The Challenge

- Action is needed from the people in the room
- What can we do to facilitate this?

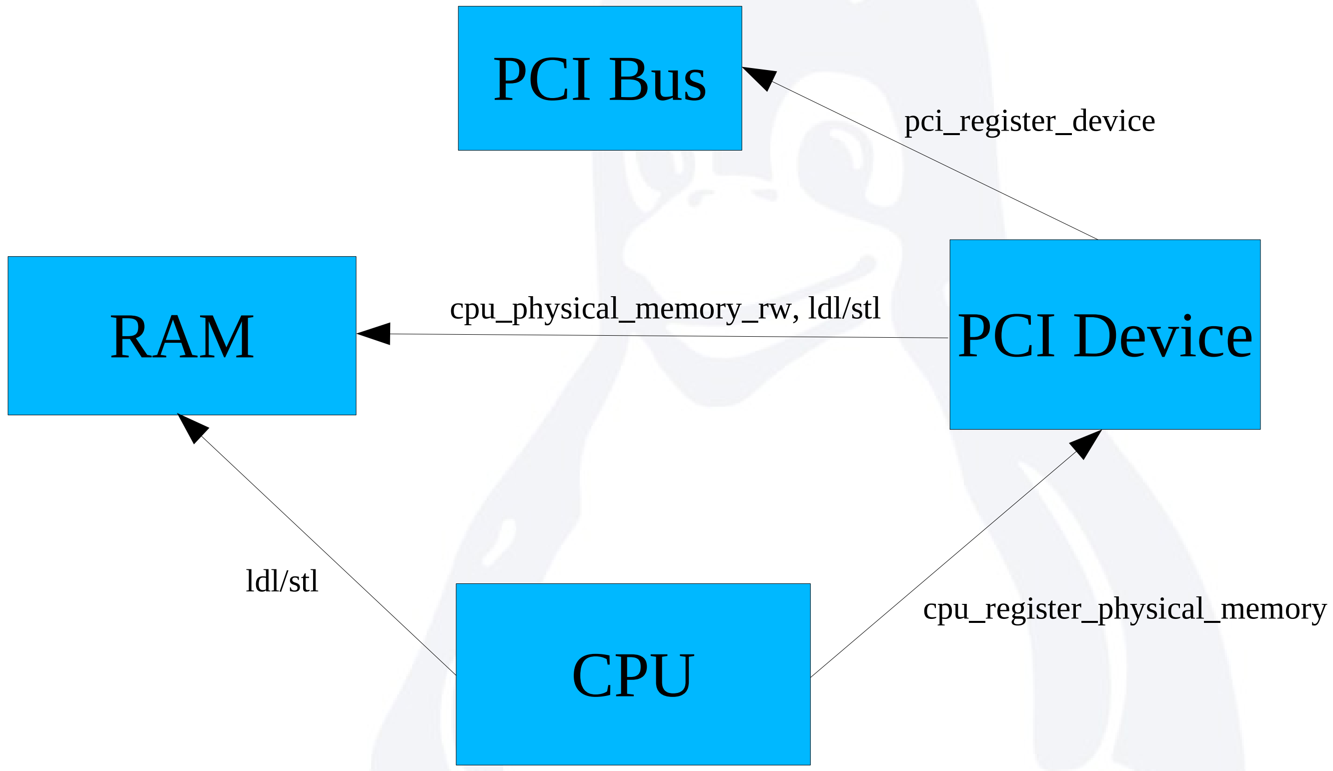
Let's get down to business

Once we fix the QEMU process,
we can start to do the fun stuff

PCI DMA Interface

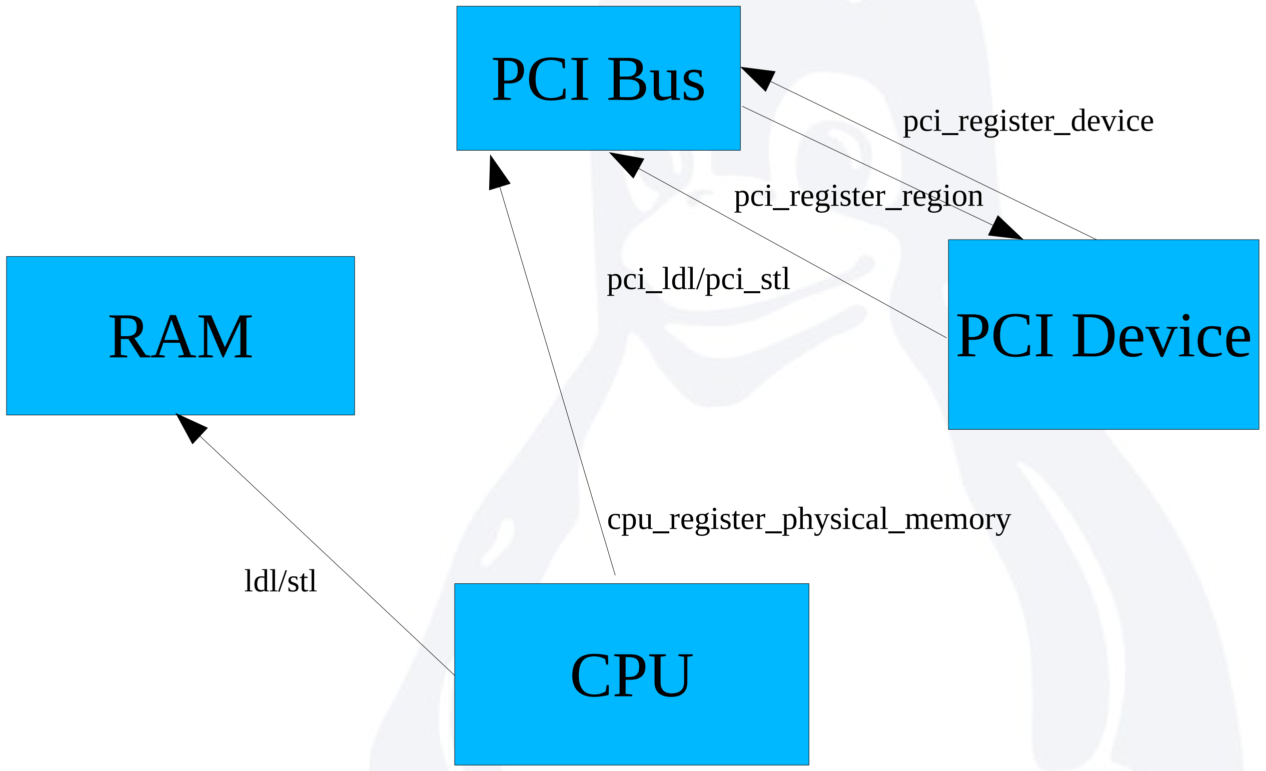
- We want zero copy
  - pci_ram_base + PA is broken, but fast
  - iommu emulation complicates zero copy
    - Some DMA engines can be programmed to perform data transformation (xor)
  - The device drivers need lots of updating
- QEMU uses a multilevel table to lookup pci_ram_base offsets for a given PA
  - Even though there are a small number of contiguous ram regions
    - 0-640K, 1M-3.8GB, 4GB+
  - We should have a fast path for RAM to avoid lookups in the l1_phys_map
- Fabrice is on-board

PCI devices today



- Often broken for > 4GB RAM
- Byte-swapping is FUBAR
- IOMMU support is impossible
- This model is fundamentally broken

DMA API



- Simplify devices; no special PIO/MMIO paths
- Everything goes through PCI bus, allowing zero-copy or IOMMU
- Can finally sanitize byte swapping

Tasklets

- Linux has sucky interfaces
  - O_NONBLOCK still blocks
  - linux-aio can block
  - posix-aio uses threads
- Threads are evil, but are the only way to get around synchronous interfaces
- Introduce a generic thread-pool
  - Tasks are normally run holding a “big qemu lock”
  - Tasks can mark themselves as not requiring the BQL
  - Tasks can also drop the BQL before sleeping
    - Caution must be exercised when doing this!

Introducing re-entrancy

- Introduce a QEMUMutex type
  - Can be used to make existing code re-entrant
  - On platforms without Tasklets, it's simple reference counting
  - Can be used to gradually wean QEMU off of the BQL
- <Refer to QEMUBH code>

Block API

- The current block API is a mix of synchronous/asynchronous behavior
  - Qcow2 does meta-data lookup synchronously but data lookup asynchronously
  - Possible source of latency
- We either need to rewrite Qcow2 to use a state machine and be entirely asynchronous **or**
- Make qcow2 synchronous, but re-entrant
- Use tasklets to parallelize qcow2 requests
- Equally applicable to other disk types (vmdk)
  - Illustrate current API verses tasklet one

Block API

Old API

- bdrv_read()
- bdrv_write()
- bdrv_aio_read()
- bdrv_aio_write()
- bdrv_aio_cancel()
- bdrv_pread()
- bdrv_pwrite()

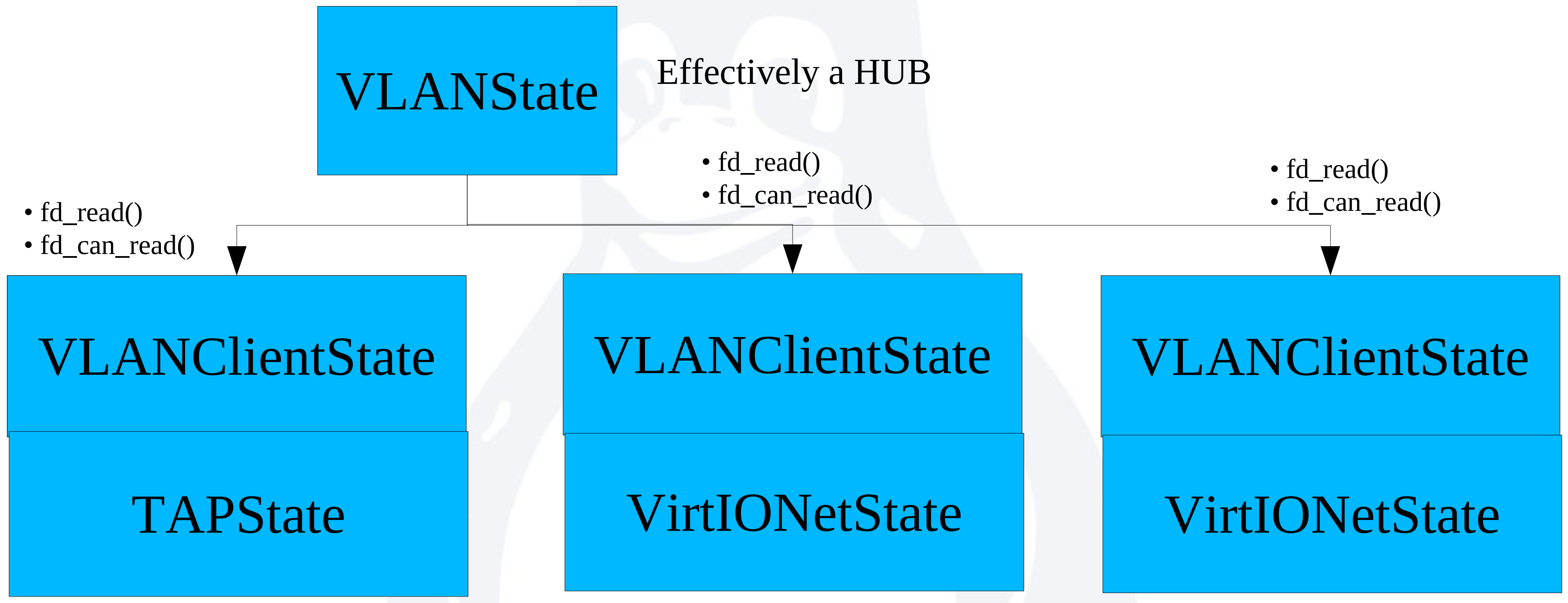
New API

- bdrv_pread()
- bdrv_pwrite()

Networking

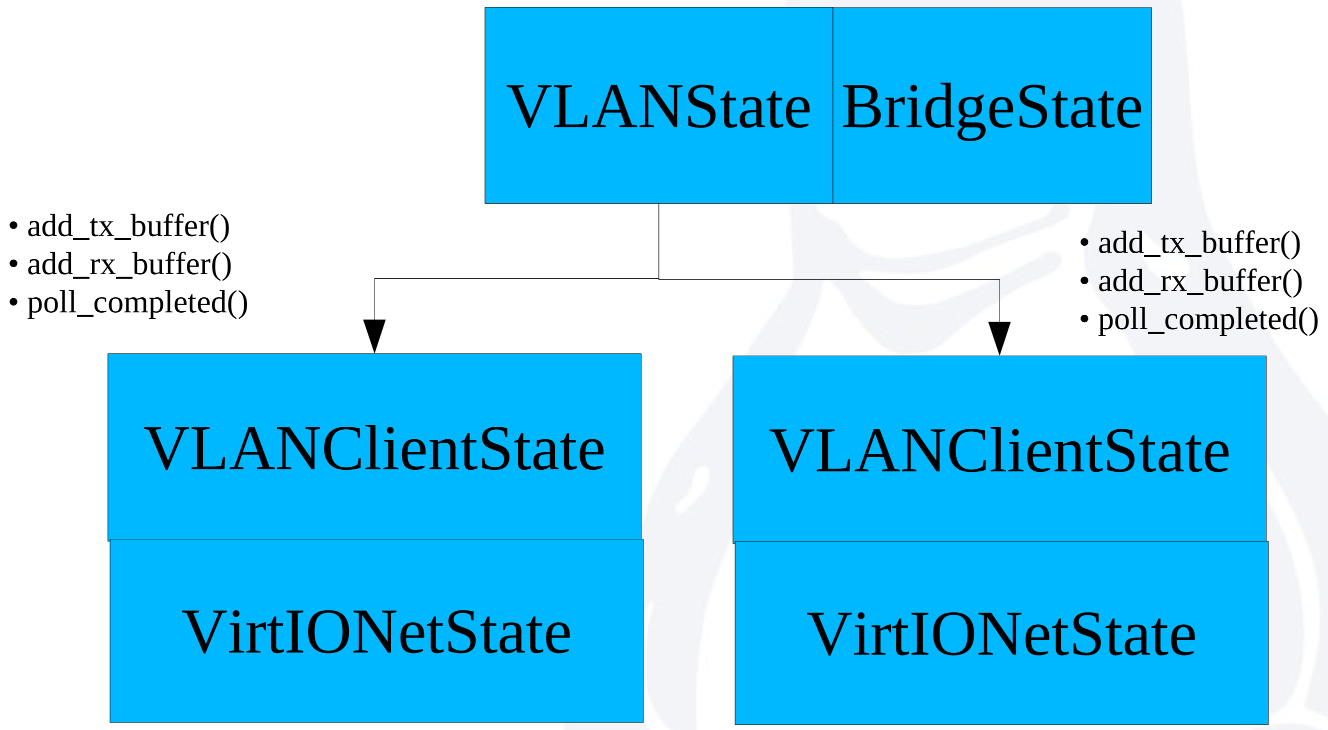
- Current API is good for performance, but bad for zero-copy
- For vringfd, we need an API that pre-registers RX buffers, preferrably with a batching API for RX completion notification
- Can also be used for rate limiting without dropping packets
  - Illustrate current API

Networking API



- Does not scale well for > 1 network device
- Cannot take advantage of DMA engines
- Is confusing to users

Proposed Networking API



- add_tx_buffer()
- add_rx_buffer()
- poll_completed()

- add_tx_buffer()
- add_rx_buffer()
- poll_completed()

- Use Linux bridging for packet copying
- Use vringfd more effectively
- Can make use of DMA offloading and associated trickery

Managability

- The monitor interface is widely used to automate management
  - Almost no error messages
- Need a non-human mode for the monitor
- Need a `select' command for receiving asynchronous operations
- Need to eliminate assumption of “global monitor”
  - Each monitor callback should take an opaque state parameter
  - term_printf() should take that state
  - Monitor commands should be dynamically registerable

General code quality

- Splitting up vl.c
  - Logically divisible into net, chardev, option parsing, etc.
- Organizing some of the existing code into directories would be a good idea
  - Block is long overdue
  - Some better organization in hw/
- This must be approach gradually
  - There is such a thing as too much cosmetic churn

Upstream support for KVM

- Need to get rid of some if (kvm_enabled())
- Need to clean-up qemu-kvm*.c
- Need to fork device models for in-kernel KVM devices
  - Already do this for PIT, need to do it for APIC
- Get migration, virtio, and PCI hot-plug upstream
  - What to do about BIOS changes?
- Introduce a target-kvm
  - Useful for embedded PPC, Itanium, and s390
  - Would produce a qemu-kvm executable

Where can we get to

A QEMU that we're happy with by KVM Forum 2009

Questions

- Comments?