Quo Vadis Virtio?

2016

Michael S. Tsirkin
Red Hat

Uses material from https://lwn.net/Kernel/LDD3/
Gcompris, tuxpaint, childplay
Distributed under the Creative commons license, except logos which are C/TM respective owners.
1.0 - towards an OASIS standard

GUEST

FIRMWARE

DPDK: vhost

VHOST: net, scsi
Standartization

• Next: v1.1

• Devices
  – Virtio-input
  – Virtio-gpu
  – Virtio-vsock
  – Virtio-9p

• Transport
  - IOMMU / Guest PMD
What to expect?

- **Devices**
  - Virtio-crypto
  - Virtio-pstore
  - Virtio-sdm
  - Virtio-peer

- **Transport**
  - Vhost-pci

- **Features**
  - Balloon page hints

- **Enabling performance optimizations**
Request processing time

![Graph showing request processing time for Virtio 1.0]
Why does batching help?

- `batch=1`
Why does batching help?

- batch=2: pipelining increases throughput
CPU caching

- Communicating through a shared memory location can cause cache misses.
- Number of these impacts latency.

![Diagram showing cache misses]
Virtio 1.0: no batching

- Access = cache miss → 5 cache misses per request
CPU caching

• Virtio 1.0 queue layout: batching

• Batch=4 → 5 misses per batch
  1.25 misses per request

1 Miss per batch
1 Miss per batch
1 Miss per up to 32 reqs
1 Miss per up to 4 reqs
1 Miss per up to 16 reqs
Estimating caching effects: Hyperthreading

- Shared cache
- Pipelining effects still in force

- Not a clean experiment: HTs can conflict on CPU
- Still interesting
Request processing: comparison

![Graph comparing request processing times for Virtio 1.0 and Virtio 1.0 HT. The x-axis represents batch numbers, and the y-axis represents nanoseconds (ns). The graph shows a comparison of performance across different batch sizes.]
Virtio 1.0 vs 1.1 (partial)

- **1.0**: 26 byte 3 bit
  - addr
  - len
  - flags
  - next

- **1.1**: 14 byte 6 bit
  - addr
  - idx
  - id
  - len
  - flags
  - V/F/M/L
Virtio 1.1: read/write descriptors

- Guest: produced 9
- Host: consumed 4

- V=0 – OK for guest to produce
- V=1 – OK for host to consume
Host: pseudo code (in-order)

while(!desc[idx].v) ← miss?
    relax();
    process(&desc[idx]);
    desc[idx].v = 0; ← miss?
    Idx = (idx + 1) % size;

• Write access can trigger miss
CPU caching

- Both host and guest incur misses on access
- No batching: 2 to 4 misses per descriptor
- Batch=4:
  2 to 4 misses per batch
  4 descriptors per cache line → 0.5 to 1 misses per descriptor
- Better than virtio 1.0 even in the worst case
Request processing: comparison

![Graph showing request processing comparison for Virtio 1.0, Virtio 1.0 HT, Virtio 1.1, and Virtio 1.1 HT across different batch sizes.](image)
Virtio 1.0: mergeable buffers

- Small packet
  - `num_buffers = 1`

- Large packet
  - `num_buffers = 3`

- Forwarding guest: no access necessary
  - `seg_num = header->num_buffers;`
  - `//seg_num = header->num_buffers;`

- Small packet throughput +15% (Andrew Theurer)
Virtio 1.1: potential gains

- Small packet
  - seg_num = header->num_buffers;
  + while (!(desc.fml & L)) {...}

- Large packet

- Avoid 1 miss per packet. Performance - TBD
Parallel ring processing?

- Virtio 1.0: workers contend on idx cache line
- Virtio 1.1: can host or guest parallelize?
- If order does not matter (e.g. network TX completion):
  - Each worker polls and handles its own descriptors
IO kick / interrupt mitigation

- event index mechanism
  - Similar to avail/used idx
  - Miss when enabling interrupts/IO

- flags mechanism
  - keep interrupts/IO enabled under light load

- first/middle/last to get interrupt per batch
  - Linux: batching using skb->xmit_more
Research

- Rings are RW
  - security issue?
  - Virtio-peer proposal?
- Test on different CPUs
  - AMD (MOESI)
  - ARM
  - Power
- Integrate in existing virtio implementations
VIRTIO_F_IOMMU_PLATFORM

• Legacy: virtio bypasses the vIOMMU if any
  - Host can access anywhere in Guest memory
  - Good for performance, bad for security
• New: Host obeys the platform vIOMMU rules
• Guest will program the IOMMU for the device
• Legacy guests enabling IOMMU will fail 😞
  - Luckily not the default on KVM/x86 😊
• Allows safe userspace drivers within guest
Virtio PMD: static vIOMMU map

- Cost: up to 4-5% for small packets. Tuning TBD
- Vhost-user can do the same
Future use-cases for vIOMMU

- Vhost-pci: VM2 can access all of VM1 memory

- Vhost-vIOMMU can limit VM2 access to VM1 memory
Wild ideas

- Apic programming: about 20% of exits
  - Virtio-apic might help coalesce with host polling?
- Idle – kvm already doing some polling
  - Virtio-idle and combine with vhost polling?
- Kgt – write-protect kernel memory in EPT
  - Extend virtio-ballon page hints?
Implementation projects

- Indirect descriptors – extra indirection
  - when not to use?

- Vhost polling
  - Scalability with overcommit – better integration with the scheduler?

- Error recovery
  - Host errors: restart backend transparently
  - Guest errors: guest to reset device
Contributing

- Implementation
  - virtualization@lists.linux-foundation.org
  - qemu-devel@nongnu.org
  - ... if in doubt – copy more

- Spec (must copy on interface changes)
  - virtio-dev@lists.oasis-open.org

- Driving spec changes
  - Report: virtio-comment@lists.oasis-open.org
  - https://issues.oasis-open.org/browse/VIRTIO
Summary

• Virtio 1.1 is shaping up to be a big release
  – Performance
  – Security
  – Features
• Join the fun
  – Spec is open: 9 active contributors / 7 companies
  – Implementations are open > 60 active contributors in the last year