OpenAMP:
Virt I/O MMIO w/ MSI

Joshua Pincus

October 27, 2020

WHEN IT MATTERS,
IT RUNS ON WIND RIVER.

Agenda

- Goals
- Background
- Current Research Focus
- Status Update
- Results
- Demo

Goals

- Objective: Migrate to an emulation-free environment with Virt I/O (MMIO transport)
- Step 1: Statically define configuration details
  - Memory for queues and shared data structures is allocated in advance via shared memory
  - Features are communicated without negotiation
  - Interrupts and related resources are defined in advance
- Step 2: Improve interrupt latency via MSI
- Step 3: Send events without emulation using hardware resources
- Step 4: Demonstrate Virt I/O devices running on new framework

Background

- Virt I/O MMIO performance is below par
    - One legacy interrupt for all queues
    - Interrupt acknowledgement involves several emulated operations
        - Check status
        - ACK interrupt
    - Notification of events also requires several emulated operations
        - Select a queue
        - Trigger an event via NOTIFY
        - All queue events are multiplexed through one NOTIFY
- Virt I/O on X86 can use PCI or MMIO.
- Virt I/O on ARM (mostly) relies on MMIO.

Current Research Focus

- Intel developed an MMIO + MSI prototype
  - Public patches for Linux virt I/O MMIO driver
  - Public patches to the 1.1 Virt I/O specification
  - QEMU and LKVM changes are left as an exercise to the wary developer
- Two sets of changes are included in the proposed spec update:
  - MSI support
    - Share vectors among queues: Map vectors to queues dynamically
    - Not share vectors among queues: Vectors and queues share a 1:1 relationship
    - MSI configuration and commands available to caller (mask, unmask, etc.)
  - MSI notification support
    - Writes to specific regions of memory trigger events specific to a queue
    - Implemented as an array indexed by vector

Status Update

- Linux changes and proposed spec were reversed to figure out what the emulation framework (aka LKVM) required.
- LKVM's Virt I/O MMIO implementation was augmented by borrowing heavily from the Virt I/O PCI MSIX implementation.
- MSI sharing/non-sharing were both implemented.
- MSI notification was also implemented.
  - Still relies on emulation to function but shows cost of using 'notify' register
  - May be adapted to be hardware-specific, aka a write to the LAPIC/IOAPIC or GIC/ITS instead of writing to an emulated register
- Linux YOCTO environment w/ UBUNTU used to warrant LKVM

Results: Performance

Test	Virt I/O PCI	Virt I/O MMIO w/o MSI	Virt I/O MMIO w/ MSI
TCP_RR (host -> guest)	20182	11009	20352
TCP_RR (guest -> host)	20463	10955	20058

- TCP_RR measures round trip latency (more trans/s = lower latency)
- Host is a Walnut Canyon system with Ubuntu
- Guest is Yocto Linux running via LKVM

Results: Emulation Traps

	TRAP (R)	TRAP (W)	CHECK IRQ (R)	ACK IRQ (W)	NOTIFY (W)	IRQ (host signal)	MSI (host signal)
Virt I/O MMIO w/o MSI	652633	652638	652615	652615	329666	660911	0
Virt I/O MMIO w/ MSI	20	66	0	0	591161	0	1.182M

- IRQ: 1M more traps => 600K fewer interrupts handled
- MSI: 2x the number of interrupts => 80%+ improved transfer rate

Demo

- Use LKVM to present a virtio-mmio instance of the virt I/O network device to an x86_64 YOCTO guest
- LKVM w/o MSI
  - Show legacy interrupt settings
  - Demonstrate netperf transmit rate via TCP_RR request/receive test
- LKVM with MSI
  - Show that MSI interrupts are being allocated
  - Demonstrate netperf transmit rate via TCP_RR request/receive test

Next Steps

- Modify LKVM tool to accommodate both Linux and VxWorks guests
- Test other Linux device drivers against LKVM changes
- Test MMIO + MSI changes on ARM target (Xilinx, Rasp Pi)
- Refine the Intel patches
  - Refine spec changes
  - Submit spec and code changes to Intel contacts for discussion + approval
- Discuss performance improvements with Virt I/O community
- Update LKVM to handle Virt I/O 1.1+ (Only legacy spec is supported)

WIND™