27th April 2015

Virtualization, kernel by-pass, fastpath acceleration

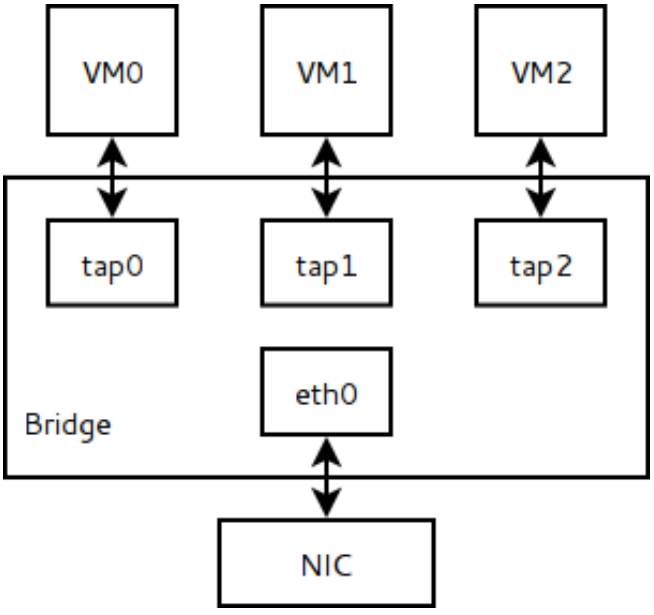
Hypervisors/Paravirtualization

KVM (kernel modules kvm.ko, kvm_intel/amd.ko). ioctl's to create and run VM's. QEMU emulates peripherals and IO. libvirt as the virtual machine manager.

KVM eco system: KVM kernel modules, KVM-QEMU (now in mainline QEMU), virtio, vhost-net.
http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf [http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf]

Packet Handling

KVM Networking (virtio-net -> virtio-pci -> qemu+kvm -> tap -> bridge)



[http://3.bp.blogspot.com/-NFj87J_jD-M/VUOx3BOJynI/AAAAAAAAGbc/Qo6FvMdTqBY/s1600/bridge.png]

Physical NIC RX/TX

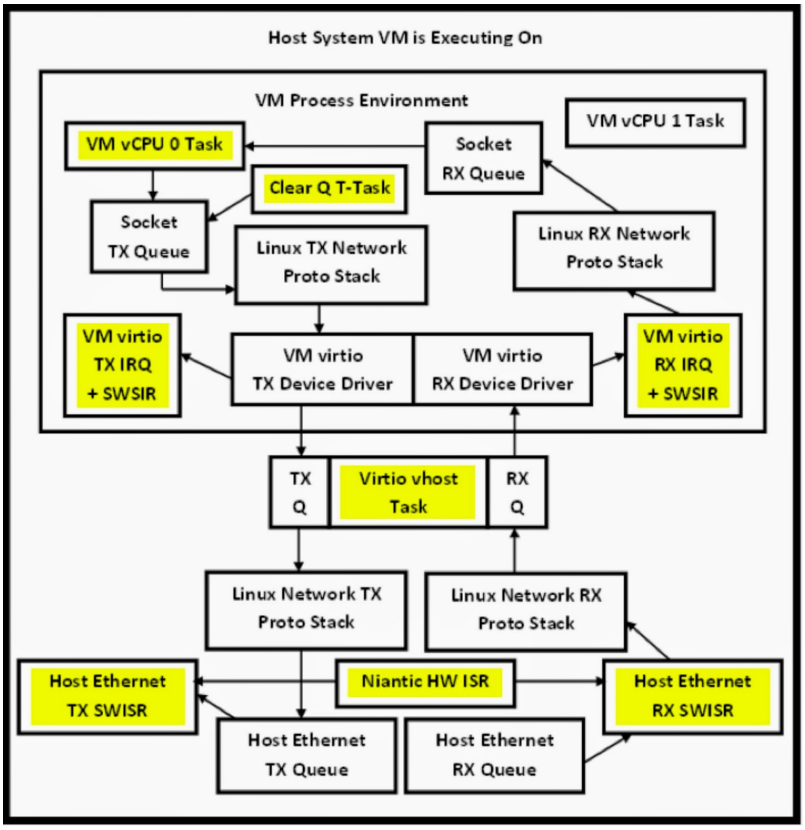
The NIC network RX queue is loaded with skb's. The NIC can DMA packet's into this memory without CPU assistance. Once, a set of RX buffers is populated the CPU is interrupted to process the RX/TX buffers. The hardware interrupt will be processed by the CPU affinitized for the RX queue. The HWISR will trigger the NAPI SWISR. NAPI SWISR will pull a non-empty RX buffer and inject it into the IP stack. The RX buffer is then replenished with a new skb. It also clears and TX buffer's and makes them free to be re-used.

Virtio Network Interface

Once the host RX and protocol stack is done it will trigger the virtio device driver SWISR in the guest (TODO: How TAP interface?). This pulls the packet into guest memory and queue's it in the virtio device RX queue.

Virtio Vhost-net Interface

Once the host RX is done the host protocol stack pushes the skb into the vhost net RX queue. It will trigger the vhost-net kernel task to pull the host skb into the guest memory skb. (TODO: How and why is it more efficient than above?). The guest skb is then placed in the guest virtio net device driver receive queue (which is the interface seen by the guest). The vhost net driver then wakes up the guest virtio net driver's HWISR. (TODO: How is this done?). At this point the flow is similar to the packet being received on a physical NIC on the host, only this process happens again in the guest.



[http://4.bp.blogspot.com/-eo23V8cOKdA/VUOalTztDOI/AAAAAAAAGbM/BM9HlVOnQLM/s1600/vm-flow.jpg]

http://www.linux-kvm.org/page/Networking_Performance [http://www.linux-kvm.org/page/Networking_Performance]

NETMAP

http://queue.acm.org/detail.cfm?id=2103536
[http://queue.acm.org/detail.cfm?id=2103536]

Virtio

http://www.ibm.com/developerworks/library/l-virtio/
[http://www.ibm.com/developerworks/library/l-virtio/]
http://www.linux-kvm.org/page/Virtio [http://www.linux-kvm.org/page/Virtio]
http://dpdk.org/doc/virtio-net-pmd [http://dpdk.org/doc/virtio-net-pmd]
http://dpdk.org/doc/guides/nics/virtio.html
[http://dpdk.org/doc/guides/nics/virtio.html]
https://www.youtube.com/watch?v=FY-iQnrrOgk
[https://www.youtube.com/watch?v=FY-iQnrrOgk]

VMXNET3

http://www.vmware.com/pdf/vsp_4_vmxnet3_perf.pdf
[http://www.vmware.com/pdf/vsp_4_vmxnet3_perf.pdf]

IO Virtualization

VMDQ, SR-IOV and IO-MMU (Intel VT-d, AMD Vi)

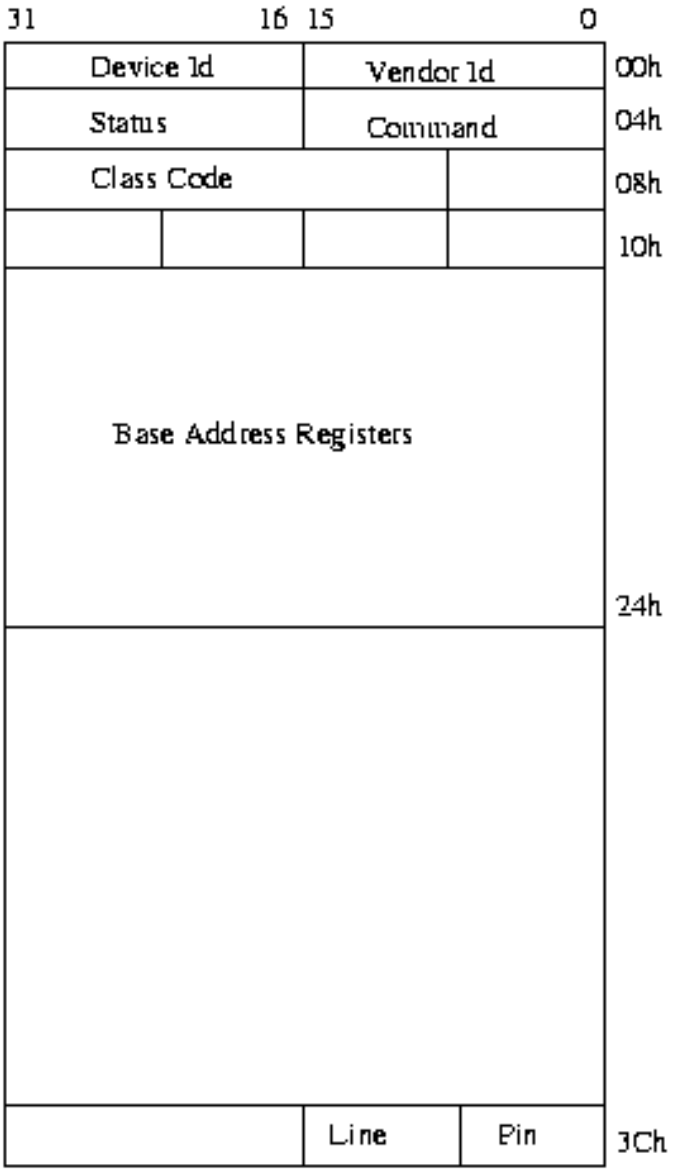
VMDq and SR-IOV are Intel's IO virtualization techniques available on some Intel NIC's. VMDq is Intel support for a L2-classifier in the NIC to figure out which destination. With SR-IOV, virtual functions are actually owned by the VM's. Physical function is owned by the hypervisor. With SR-IOV the packet gets DMA's directly into the VM's memory. This exposes the physical NIC as multiple logical NIC's that each VM can use independently.
https://www.youtube.com/watch?v=hRHsk8Nycdg
[https://www.youtube.com/watch?v=hRHsk8Nycdg]

IO-MMU is another IO virtualization technique which uses mappings from host device physical IO memory locations into guest addresses. PCI pass through. (TODO: Understand more).
http://www.intel.com/content/www/us/en/embedded/technology/virtualization/vt-directed-io-spec.html
[http://www.intel.com/content/www/us/en/embedded/technology/virtualization/vt-directed-io-spec.html]

PCI (Peripheral Component Interconnect)

http://www.tldp.org/LDP/tlk/dd/pci.html
[http://www.tldp.org/LDP/tlk/dd/pci.html]

PCI has 3 address spaces. PCI I/O, PCI Memory and PCI configuration space. I/O, Memory regions are used by PCI drivers and configuration space is used by kernel PCI initialization.



[http://3.bp.blogspot.com/--iLNvk3JN2Q/VUPVSPYcFlI/AAAAAAAAGbs/dIU3g2DOiUI/s1600/pci-config-header.gif]

Every PCI device has it's PCI configuration header at a specific offset within the PCI configuration space. BAR (Base Address Registers) are used to determine the type and amount of I/O and memory space the device can use.

The PCI device maps internal registers into the I/O space when enabled via the PCI configuration header (no access to internal registers is possible by the device drivers if not enabled). The device drivers use the PCI I/O and memory space to communicate with the PCI device.

Linux PCI initialization

Three pieces PCI device driver, PCI BIOS, PCI Fixup. Device driver learns and builds topology (pci_root -> pci_bus -> pci_dev).

UIO

UIO provides a standard way of developing user space drivers. The generic UIO driver in the kernel provides for two main things (1) Register for device interrupts and propagate them to the user space driver. (2) Way to map device memory to user space. This functionality is then exposed to user space by creating a device in the kernel "/dev/uioXX" and entries in the sysfs.
Taking a PCI UIO device as an example (drivers/uio/uio_pci_generic.c). The PCI UIO 'probe' function creates a generic UIO device "/dev/uioXX" and registers an interrupt handler 'uio_interrupt'.
A 'read' on this device put's the user process in this device's 'wait' list for an interrupt to fire. When an interrupt is fired it is handled by the PCI device interrupt handler and 'uio_event_notify' is called to notify the blocked user process. A 'write' on the device changes IRQ state.

User Space Network Driver Example

A network driver usually has 3 pieces (1) Configuration space for registers (2) Packet descriptor space to communicate with the device (3) Packet data space for actual IO'd packet data.
In a kernel space driver these regions are mapped into kernel space. Packet's then make their way upto the user space when the socket call is serviced for e.g and 'copy_to_user' happens.

A user space network driver has the (3) pieces of network device memory mapped into the user space. So, (1)/(2)/(3) are all directly accessible and there is no copy involved as in the kernel space driver. Only interrupts are handled by a small component in the kernel on behalf of the user space driver.

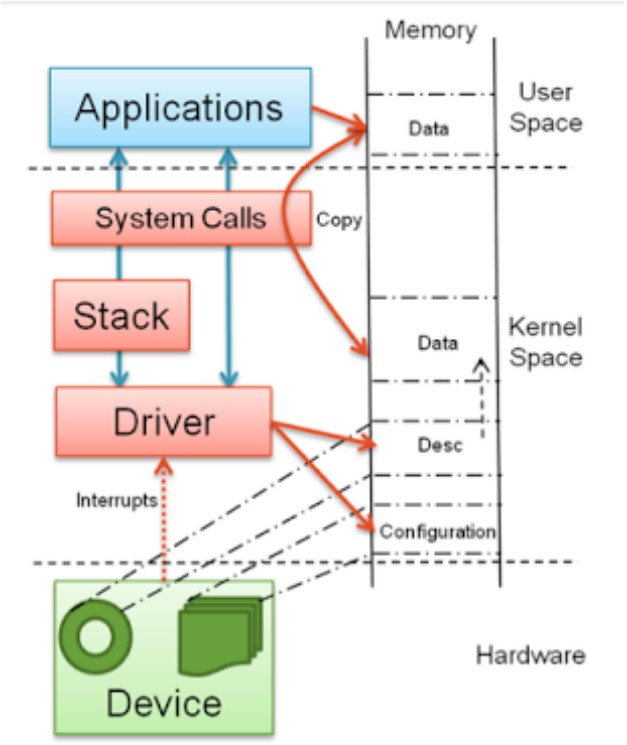


Figure 1: Kernel space network driver

[http://1.bp.blogspot.com/-GWG0vkoGtZQ/VUPr9Fy4g6I/AAAAAAAAGb8/hDoYVIGvtvQ/s1600/kernel_driver.png]

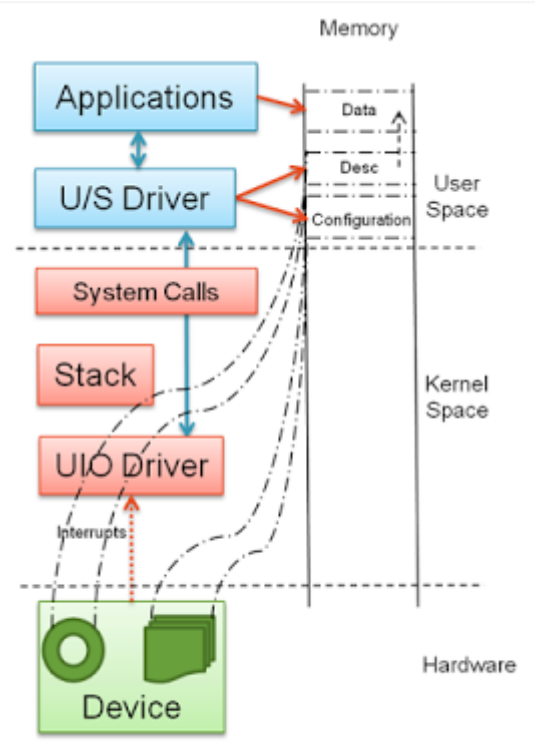


Figure 2: User space network driver

[http://3.bp.blogspot.com/-SVkip4DghR8/VUPr9PCJBFI/AAAAAAAAGcA/u1kdVTq0NSg/s1600/user_space_driver.png]

http://www.embedded.com/design/operating-

systems/4403058/Accelerating-network-packet-processing-in-Linux [http://www.embedded.com/design/operating-systems/4403058/Accelerating-network-packet-processing-in-Linux]

Links

- http://slides.com/braoru/kvm/fullscreen#/ [http://slides.com/braoru/kvm/fullscreen#/] (Nice introduction to KVM, QEMU, VIRTIO, VHOST-NET)
- http://www.ibm.com/developerworks/library/l-virtio/ [http://www.ibm.com/developerworks/library/l-virtio/] (Virtio)
- http://www.ibm.com/developerworks/linux/library/l-hypervisor/index.html [http://www.ibm.com/developerworks/linux/library/l-hypervisor/index.html]
- http://blog.vmsplice.net/2011/03/qemu-internals-overall-architecture-and.html [http://blog.vmsplice.net/2011/03/qemu-internals-overall-architecture-and.html] (Good blog on KVM/QEMU/Virtio)
- https://networkbuilders.intel.com/docs/network_builders_RA_NFV.pdf [https://networkbuilders.intel.com/docs/network_builders_RA_NFV.pdf] (Section 9 provides a good overview of the network interface components)
- http://www.embedded.com/design/operating-systems/4435211/KVM-ARM--The-Design-and-Implementation-of-the-Linux-ARM-Hypervisor [http://www.embedded.com/design/operating-systems/4435211/KVM-ARM--The-Design-and-Implementation-of-the-Linux-ARM-Hypervisor]

Posted 27th April 2015 by Vamsi

Labels: software, tech