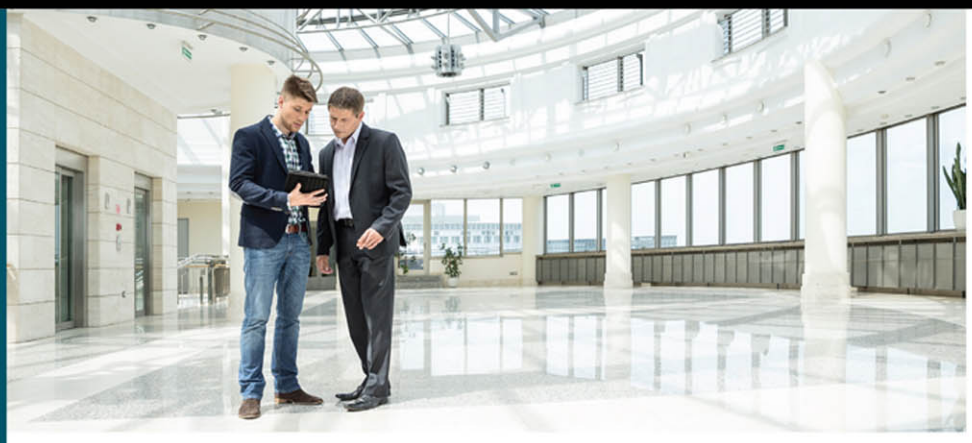

Virtual Routing in the Cloud

Arvind Durai
Stephen Lynn
Amit Srivastava

ciscopress.com

When any of these three events occur, `main_loop_wait()` invokes a callback that responds to the event. The callback should be quick to prevent the system from being unresponsive.

For executing guest code, QEMU deploys the following mechanisms:

- **Tiny Code Generator (TCG)**—This mechanism emulates guests by using dynamic binary translation.
- **KVM**—As discussed earlier, the KVM module virtualizes the hardware resources and presents the `/dev/kvm` interface in the Linux file system.

Both TCG and KVM allow the execution control to be given to the guest and allow the guest to execute its code.

QEMU has one thread per vCPU plus a dedicated event loop thread. This is called the `IOTHREAD` model. In the older non-`IOTHREAD` model, one QEMU thread executed the guest code and the event loop. In the new `IOTHREAD` model, each vCPU thread executes the guest code, while the `IOTHREAD` runs the event loop.

Figure 3-16 shows the architecture of a system using KVM and QEMU.

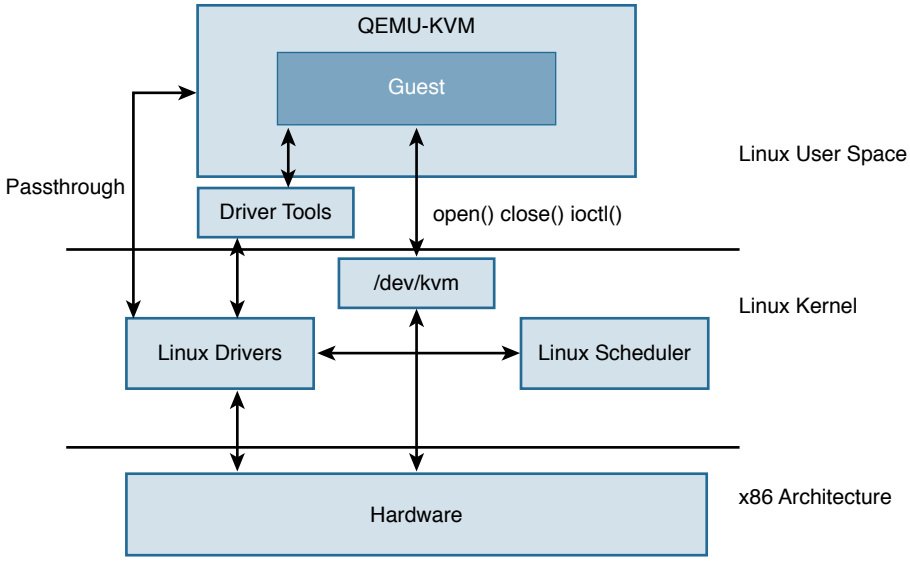

Figure 3-16 *QEMU KVM Architecture*

Management Daemon (Libvirt)

Management is an important aspect for virtualization. Libvirt is an open source management tool that can be used to manage a virtualized environment. Libvirt, written in C (with bindings to other languages, such as Python), provides an API for managing multiple hypervisors.

Writing and maintaining applications is expensive. Libvirt allows sharing of applications between hypervisors and provides security and remote access, too.

Libvirt is designed to mainly manage virtual machines. It offers the following types of management:

- **VM management**—Libvirt can manage the various life cycle operations of a virtual machine, such as starting, stopping, saving, pausing, moving, and adding/removing CPUs and memory.
- **Access management**—Because Libvirt functionality is available for all machines that run the `libvirtd` daemon, you can use SSH (or any remote login mechanism) for remote access.
- **Network management**—Any host that runs the `libvirtd` daemon can be used to manage physical or virtual interfaces and physical or virtual networks. When the `libvirtd` system daemon is started, a NAT bridge is created. This is called `default` and allows external connectivity. For other network connectivity, you can use the following:
  - A virtual `bridge` that shares data with a physical interface
  - A virtual `network` that enables you to share data with other virtual machines
  - A `macvtap` interface that connects directly to the physical interface on the server on which you host the VM
- **Storage management**—Any host that runs the `libvirtd` daemon can be used to manage various storage types and file formats.

Libvirt management is done mainly using `virsh` and `virt-manager`.

Figure 3-17 shows the Libvirt architecture.

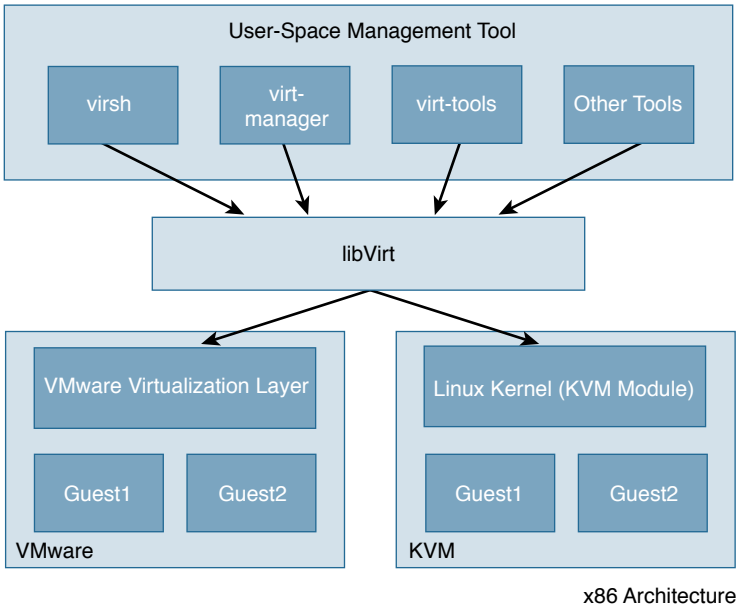

Figure 3-17 *Libvirt Architecture*

User Tools (`virsh`, `virt-manager`)

`virsh` is a piece of code that is used for managing VMs. This command-line tool is very useful for scripting and scaling VM installations. You also get an interactive terminal with `virsh` that can be entered if no commands are passed. Unprivileged users can use `virsh` in read-only mode.

Table 3-1 shows the command-line tools you can use with `virsh` to manage guest VMs.

Table 3-1 `virsh` *Quick Reference*

Command	Description
`help`	Prints basic help information.
`list`	Lists all guests.
`dumpxml`	Outputs the XML configuration file for the guest.
`create`	Creates a guest from an XML configuration file and starts the new guest.
`start`	Starts an inactive guest.
`destroy`	Forces a guest to stop.
`define`	Outputs an XML configuration file for a guest.
`domid`	Displays the guest's ID.
`domuuid`	Displays the guest's UUID.

Command	Description
`dominfo`	Displays guest information.
`domname`	Displays the guest's name.
`domstate`	Displays the state of a guest.
`quit`	Quits the interactive terminal.
`reboot`	Reboots a guest.
`restore`	Restores a previously saved guest stored in a file.
`resume`	Resumes a paused guest.
`save`	Saves the present state of a guest to a file.
`shutdown`	Gracefully shuts down a guest.
`suspend`	Pauses a guest.
`undefine`	Deletes all files associated with a guest.
`migrate`	Migrates a guest to another host.

You can also use `virsh` to manage the resources that are given to the guest or hypervisor with the commands shown in Table 3-2.

Table 3-2 *Commands Used to Manage Resources with* `virsh`

Command	Description
`Setmem`	Sets the allocated memory for a guest.
`Setmaxmem`	Sets the maximum memory limit for the hypervisor.
`Setvcpus`	Changes number of virtual CPUs assigned to a guest.
`Vcpuinfo`	Displays virtual CPU information about a guest.
`Vcpupin`	Controls the virtual CPU affinity of a guest.
`Domblkstat`	Displays block device statistics for a running guest.
`Domifstat`	Displays network interface statistics for a running guest.
`attach-device`	Attaches a device to a guest, using a device definition in an XML file.
`attach-disk`	Attaches a new disk device to a guest.
`attach-interface`	Attaches a new network interface to a guest.
`detach-device`	Detaches a device from a guest and takes the same kind of XML descriptions as the command `attach-device`.
`detach-disk`	Detaches a disk device from a guest.
`detach-interface`	Detaches a network interface from a guest.

With these commands you can manage the guest effectively and automate a lot of guest bring-up and configuration sequences.

`virt-manager` provides a graphical way to manage VMs and hypervisors. It does the same things as `virsh` but via a user-friendly graphical interface.

Hyper-V

Hyper-V, formerly known as Windows Server Virtualization, can create virtual machines on the x86 platform. In October 2008, Hyper-V Server 2008 was released, and since then it's been used as a hypervisor platform for other members of the Windows Server family.

Hyper-V is a type 1 microkernel hypervisor that resides directly on the hardware. The microkernel architecture optimizes performance and reduces adoptability issues with the underlying hardware.

The architecture uses a parent VM that hosts the drivers. The guest operating system interfaces with the parent partition to access hardware resources' memory, CPU, storage, and so on. The guest operating system works within the privileged boundary. Figure 3-18 gives a high-level overview of the Hyper-V architecture.

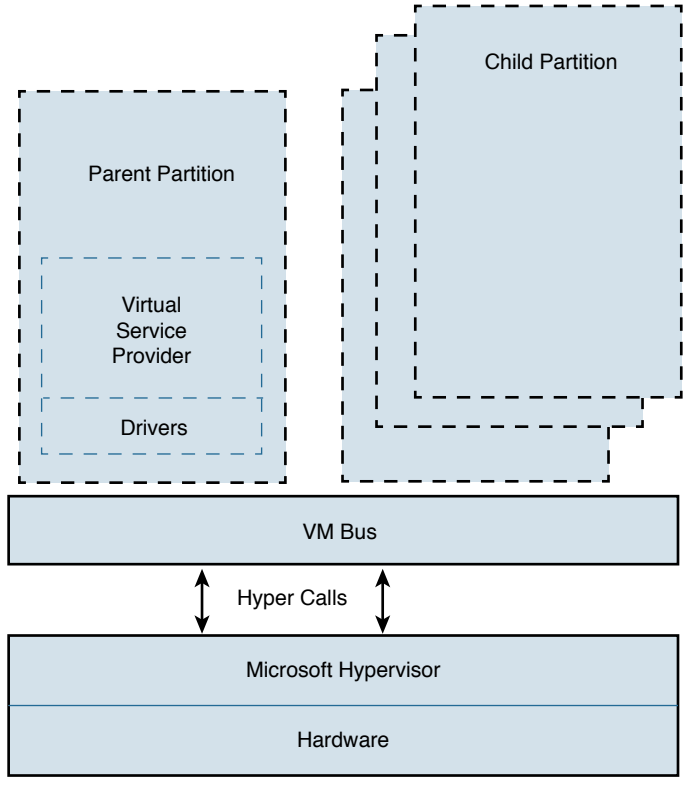

Figure 3-18 *Hyper-V Architecture High-Level View*

On the top of the hypervisor is one parent partition and one or more child partitions. This partitioning creates virtual isolation within the hypervisor for physical memory address space and virtual processors.

The child partition can host a guest operating system or system functions for Microsoft Windows Server. For example, when virtual Hyper-V stack management gets installed in the parent partition, the subsidiary Windows Server functionality is loaded in the child partition. Each virtual machine has its own security boundary. Microsoft refers to this as an operating system environment (OSE) that defines the component of a virtual machine. The VM has its own identity, computer account, IP address, and network resources. Child partitions have only the virtual view of the hardware resources. The child partition sends a request to the virtual devices, which gets redirected to the hypervisor in the parent partition that handles this request.

The parent partition has access to hardware devices and controls the resources used by the child partition. The child partition accesses the hardware resource using the drivers in the parent partition space. The parent partition also acts as the broker among the multiple child partitions and the hardware for accessing the resources. All data and instructions between the parent and child partition go through the virtual machine bus. The architecture allows the user to leverage plugin devices, which in turn allow direct communication between parent and child partitions.

Xen

Xen is another type 1 hypervisor that supports para-virtualization. Xen originated as a college research project at the University of Cambridge. Ian Pratt, who was a lecturer in Cambridge, led this project and later cofounded XenSource, Inc. First available in 2004, Xen was originally supported by XenSource, Inc. Eventually, Xen was moved under the Linux Foundation as a collaborative project.

Starting with Xen 3.0, all guest VMs can run their own kernels and take advantage of para-virtualization, which removes the need to emulate virtual hardware resources, makes the guest aware of the hypervisor, and enables access to the hardware resources for I/O efficiency. Instead of the guest spending time and extra cycles performing tasks to get resources from the virtual environment, these guests can use the hooks of para-virtualization to allow guest and host to communicate and acknowledge these tasks.

Figure 3-19 gives a high-level architectural overview of the XEN hypervisor.