Safe harbor statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.

The development, release, timing, and pricing of any features or functionality described for Oracle’s products may change and remains at the sole discretion of Oracle Corporation.
Multi-process QEMU

2019 KVM Forum

John Johnson
Elena Ufimtseva
Jag Raman
Oracle Virtualization
November 1, 2019
Agenda

1. Architecture overview
2. Current status and usage
Agenda

1. Architecture overview
2. Current usage and status
QEMU as cloud HV

- QEMU is HV of choice in many cloud environments
- Public clouds can host VMs from many customers
  Vast majority of these customers are running workloads in the cloud
  taking advantage of the scalability and flexibility of public clouds
  But some can be malevolent actors
  trying to look into other customers’ VMs
- Public clouds need to plan for the worst case
Cloud defense mechanisms

- **Use a different HV**
  But QEMU provides many features desirable in a cloud environment

- **Minimize QEMU**
  Reduce attack surface by configuring as few devices and services as possible
  Use virtio devices and virtio-user daemons to further reduce surface
  But this reduces ‘lift and shift’ capability since existing OS instances may rely on legacy devices

- **Run each QEMU instance in its own container**
  Place each VM’s data in the container
  Use SELinux or AppArmour policies to restrict access of processes within each container
  These policies apply to processes, and QEMU is a single process
Monolithic QEMU

- QEMU is a monolithic process that combines many different functionalities
  - VM control plane
    - Initial guest setup and monitoring
    - Live-migration, hot-plug, storage snapshots etc.
  - CPU and chipset emulation
    - Executes guest under KVM
    - Handles guest exits, interrupts, etc.
  - IO device Emulation
    - SW emulation of IO devices
- All these functionalities require QEMU to access many host services
- Any exploit can allow a malevolent guest to gain all of QEMU’s access rights
Current monolithic QEMU

QEMU

- VGA emulation
- LSI 895 emulation
- CPU/chipset emulation
- QEMU control plane

KVM

- USB emulation
- IDE emulation
- E1000 emulation

Guest
Running QEMU in multiple processes

- Many security policies are process based
  SELinux has rules to limit processes as what files or device objects it can access
  Seccomp can limit processes as what system calls they can execute

- Separating QEMU into multiple processes allows finer-grained privileges to be assigned to each one
  Disk controller emulation process only given access to disk images belonging to the guest
    iSCSI emulation can be limited to iSCSI ports and storage host IPs
  All device emulation processes can be limited from using fork() or exec() to create a host shell
Device emulation in separate process

- Good place to start for a number of reasons
  - Largest surface a malevolent guest could attack
    - provided by the large number of devices QEMU can emulate
  - Ease of implementation
    - device emulation internally implemented as objects within QEMU
    - object method boundaries can be used as process separation point
  - Scalability
    - number of processes can scale to number of devices in VM
QEMU with device emulation in separate processes

QEMU

IDE emulation

LSI 895 emulation

E1000 emulation

QEMU control plane

CPU/chipset emulation

Guest

KVM
QEMU object model

- **Class-based inheritance model**
  “object” is super-class

- **“machine” class models platform**
  Initializes emulated system configuration

- **“device” class models IO devices**
  Most device emulation code is in objects of this sub-class

- **“bus” class models IO buses**
  Enumerates child devices of an IO bus
QEMU initialization

- **QEMU command line options are parsed**
  - machine, -device, -blockdev options parsed

- **Device backends initialized**
  - Placed on lists so they can be found by their associated device objects

- **Machine object initialized**
  - Hand crafts platform built-in device objects
    - host bridge, IDE, APIC, serial, etc.

- **Device objects initialized**
  - Looking up any backends they need
QEMU objects for SCSI drive

- pc-i440fx-machine
- PCI
- lsi53c895a
- SCSI
- scsi-hd
- file backend
**Emulation process**

- **Runs unmodified device and bus objects from monolithic QEMU**
  - Built from same QEMU source tree
  - Startup handshake to ensure QEMU and remote process are from same build

- **Runs unmodified device backends from monolithic QEMU**
  - Also built from QEMU tree
  - Same command line arguments used in both QEMU and remote process

- **New remote machine class object**
  - Replaces machine object in monolithic QEMU
  - Performs similar functions
    - Creates initial machine from configuration messages from QEMU
    - Handles interrupt and IOMMU requests from device models
Emulation process cont.

- **Proxy service to talk to QEMU**
  - Instantiates machine object
  - Instantiates device objects from QEMU configuration messages
  - Creates device backends from command line arguments
  - Routes requests from QEMU to device objects
    - e.g., guest reads and writes to address space of device
  - Routes machine requests back to QEMU
    - e.g., IOMMU mapping requests
Emulation process

remote machine → PCI

lsi53c895a

SCSI

scsi-hd

file backend
QEMU changes

- **New remote process manager**
  Manages communication with emulation process
  Created with new “-remote” command line argument
  - `remote rid=<rid>,socket=<socket path>`
    communicates with existing process over given socket
  - `remote rid=<rid>,command="<emulation process args>"`
    creates socket and executes given command

- **No Device backends - only needed in emulation process**
QEMU changes

- **Proxy objects replace the device emulation objects**
  
  Forwards guests events, such as MMIOs, to emulation process
  
  Specified by new rid=<rid> option for -device
  
  - `device lsi53c895a,id=scsi0` specifies traditional emulation within QEMU
  - `device lsi53c895a,id=scsi0,rid=disk-proc` specifies remote emulation
    
    “disk-proc” is ID of remote process manager to forward requests to
  
  Exist in QEMU at same point the object and bus hierarchy as device object it replaces
  
  e.g., LSI SCSI proxy is a sub-class of “pci-device” and is a child of a PCI bus object

- **Not all devices emulation objects need proxies**

  Only those with guest interactions
  
  e.g., SCSI controller does, but SCSI devices do not
Proxy object hierarchy for LSI SCSI controller

- “pci-device-proxy” class forwards guest config space reads and writes to emulation process, mostly for BARs and interrupts
- “lsi53c895a-proxy” class forwards guest MMIOs to the device memory space to emulation process.
QEMU objects left behind

pc-i440fx-machine → PCI

lsi53c895a-proxy
Putting it all together

QEMU

pc-i440fx-machine → PCI → Lsi53c895a-proxy

remote machine → PCI → Lsi53c895a → SCSI → scsi-hd

file backend

emulation process
Agenda

1. Architecture overview
2. Current usage and status
I want to try it!

- Clone it from http://github.com/oracle/qemu
- Configure it with --enable-mpqemu
- More command-line options?
  Not really! “-remote”, “rid=”
- One device per process?
  Nope!
- One remote process?
  Nope, can have more!
Try with lsi53c895a

qemu-system-x86_64 -name "aww_qemu" -machine q35,accel=kvm \
-smp sockets=1,cores=1,threads=1 -m 2048 \
-object memory-backend-file,id=mem,mem-path=/dev/shm/,size=2G,share=on \
-numa node,memdev=mem -drive format=raw,file=/root/ol7.qcow2 \
-device lsi53c895a,id=lsi0,rid=8 \
-device scsi-hd,id=drive2,drive=drive_image2,bus=lsi0.0,scsi-id=0,rid=8 \
-remote rid=8,command="-drive id=drive_image2,,file=/root/remote-process-disk.img" \
-boot d -monitor stdio -vnc :0
Functionality

- QMP monitor and HMP commands
- Device hot-plug
- Live Migration
- SELinux policies
- Libvirt support
Teach your device to run in a separate process

- Write proxy object for QEMU
  Leverage current PCI proxy for PCI devices

- Add your device’s object to remote process build

- Add QMP/HMP commands that manage your device to remote process build
Future work

- Work with KVM/QEMU community, address feedback more efficiently
- Add more device types
- Improve performance
  - Shorten path for MMIOs
- Security hardening
- Libvirt support upstreaming
Performance

iSCSI Block Volume - Built-in vs. Separated LSI - 4K IOPS

- Built-in LSI
- Separated LSI

IOPS

Read / Write Mix

0/100 5/95 35/65 50/50 65/35 95/5 100/0
MMIO Acceleration

- Impact on CPU usage
- Delay in processing MMIO
  - Return to QEMU
  - Syscall overhead

Accelerated MMIO processing
Thank you

John, Elena and Jag