Corporate Technology

Developing Linux inside QEMU/KVM Virtual Machines

Jan Kiszka, Siemens AG, CT T DE IT 1
Corporate Competence Center Embedded Linux
jan.kiszka@siemens.com

SIEMENS

Agenda

- Motivation
- Introduction & basic concepts
- QEMU/KVM as a kernel debugger
- Upcoming features & improvements
- Summary
- [Demonstration]

SIEMENS

How Do You Do Kernel Development?

Test & debug on the development host

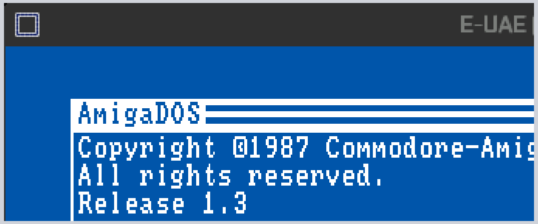
- \+ Handy and fast (modules)
- – Invasive (kernel reboots) and risky

Use separate test systems

- \+ Architectural independence, fault containment
- – Setup & maintenance efforts, hardware costs

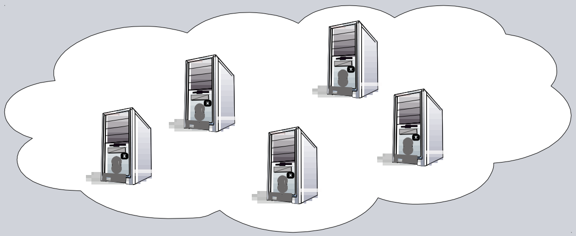
Emulate target system

- \+ Hardware independence, transparency, reproducibility, costs
- – Speed, potential modeling effort



Exploit hardware virtualization

- \+ Emulation + speed
- – Architectural support needed

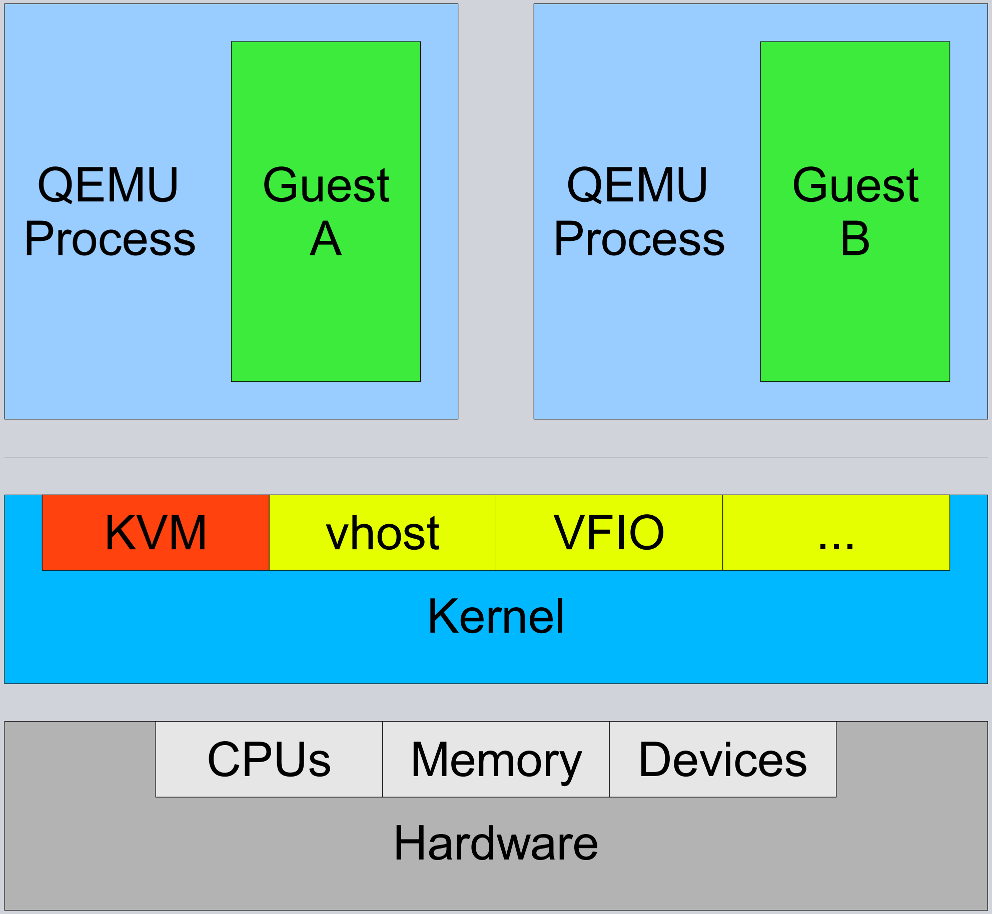


QEMU/KVM in a Nutshell

QEMU
- Multi-arch machine emulator
- Tons of device models
- gdb server & monitor
- KVM acceleration

KVM
- Gatekeeper for HW- and kernel-assisted virtualization
- Fast device models
- PCI pass-through

qemu-kvm fork
- Optimal x86-QEMU/KVM
- Required for pass-through
- To be obsoleted by QEMU



QEMU/KVM as Test Platform – Getting Started

SIEMENS

Enable KVM (x86)
- modprobe kvm-intel/amd

qemu-kvm package
- Pick at least 0.15.x or 1.0.x

Start from command line
- Hairy but powerful interface
- Can be as simple as
  `qemu-system-$arch /path/to/image`

Nubae, licensed under CC BY-3.0

Use run-qemu.sh wrapper
- lkml.org/lkml/2011/11/5/83
- Beginners guidance, kernel pick-up from build directory

Use libvirt
- Multi-VM management, privilege separation, language bindings
- Command line pass-through for enhance QEMU features

Virtual Consoles

Benefits

- No wiring, no limits
- Can be faster than real ports

Multiple frontend options

- Serial port emulation
- virtio
- VGA text console

...and backends

- Local tty
- TCP/Telnet
- Pipe
- File
- …

Guest Image Management

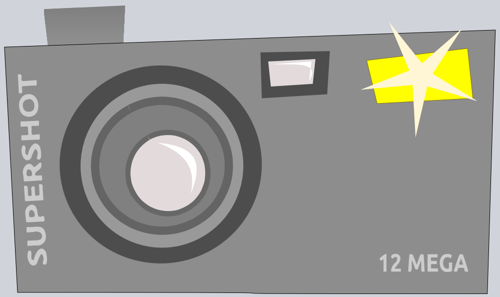
Disk images
- Check qemu-img for image management
- Use raw format for speed – and loop-back mounting
- Use qcow2 or qed for thin provisioning

Disk pass-through (for the brave ones)
- `qemu-system-$arch -snapshot /dev/sda`
- Will boot your host (but does not modify it)
- Requires root privileges, forgetting -snapshot is lethal

NFS root
- Classic way in embedded
- Use virtio-net for optimal performance

9pfs
- File system pass-through
- Use for rootfs and/or as shared folder

Taking and Using Snapshots



Use cases

- Accelerate test startup
- Roll back to consistent state

Disk image snapshots

- **`qemu-system-$arch disk.img -snapshot`**
- Create live (**`snapshot_blkdev`**) or offline (qemu-img)
- Merge-back live (**`commit`**) or offline

Machine snapshots

- loadvm/savevm with qcow2 images
- Migrate to disk (**`migrate exec:'cat > snapshot.img`**')
- Upcoming live backup

And with fs pass-through?

- Host-side snapshots (lvm, btrfs, unionizing fs)
- Need to coordinate fs and machine snapshot

Device Pass-Through

Various buses & devices supported

- PCI (x86-only so far)
- USB (1.1 & 2.0, experimental 3.0)
- Smartcards
- Bluetooth HCI
- SCSI (might be buggy)
- TPM (upcoming)

Beware of host controller emulation flaws!

Scenarios

- Satisfy HW dependencies w/o emulation
- Enable driver development against real HW
- Shorten turn-around times using snapshots + device hotplug or suspend/resume

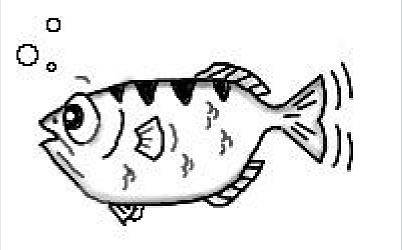
QEMU as Kernel Debugger – Basics

SIEMENS

Imagine QEMU as JTAG hardware debugger – and more!

Two central interfaces

- Built-in gdb server
- Monitor console
- Both support various transports



Jamie Guinan, licensed under CC BY-SA-3.0

gdb server quick-start

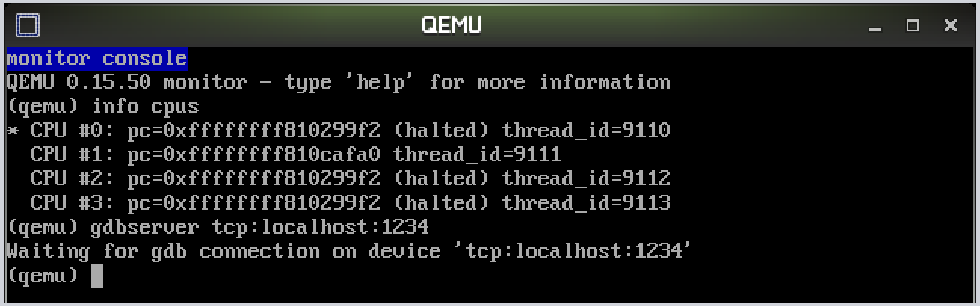
- `host# qemu-system-$arch -s`
- Build kernel with `CONFIG_DEBUG_INFO`
- `host# gdb vmlinux`
- `(gdb) target remote :1234`

Optional: load module symbols

- `guest# cat /proc/modules`
  Look up module base address
- `(gdb) add-symbol-file /path/to/module.ko <base address>`

SIEMENS

QEMU Monitor



Inspect the virtual machine

- `info qtree`, `mtree`, `pci`, `usb`, `network`, `cpus`, `registers`, …
- `x`, `xp` (memory access)
- `i`, `o` (I/O port access)

Control the VM

- Stop/continue, trigger reset or power button
- Hot plug devices
- Inject NMI, MCE, PCIe error
- Late gdb server activation, ...

Access channels

- Dedicated console (e.g. virtual console – “CTRL-ALT-2”)
- Via gdb session (`(gdb) monitor info registers`)

Soft, Hard or Step by Step?
KVM Breakpoint Architecture

Software breakpoints
- Unlimited resource
- Inject trap instruction into guest code
- Intercept traps
  - Report host originated traps to gdb
  - Reinject guest originated traps

Hardware breakpoints
- Limited by hardware resources
- If in conflict with guest usage, host wins

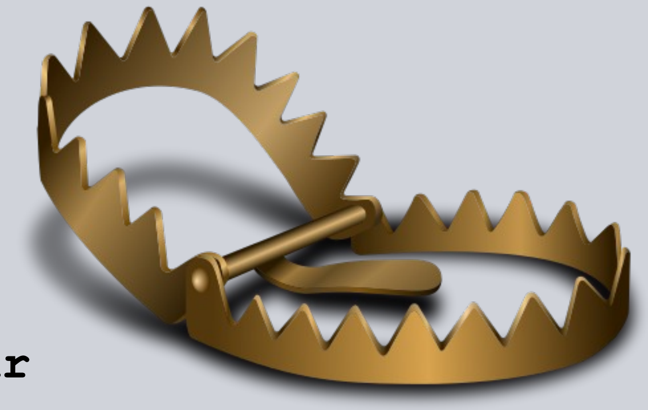
Single stepping
- Similar to hardware breakpoints
- x86: TF can “leak” to guest stack

Note: No limitations and guest visibility in CPU emulation mode

Using Watchpoints

Helpful to hunt memory corruptions

- Provided corruptions hits known area
- Provided low rate of valid changes



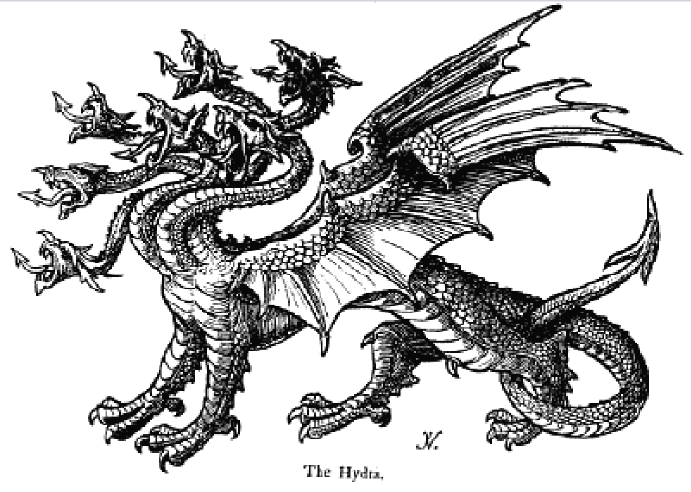
Beware of hard vs. soft

- `(gdb) watch my_global_var`
  `Hardware watchpoint 1: my_global_var`
  => Uses limited HW resources
  => Fails if sizeof(my_global_var) > watchpoint capacity
- `(gdb) watch *my_local_ptr`
  `Watchpoint 1: *my_local_ptr`
  => *Will single step, will be removed when leaving scope*
- `(gdb) watch -l[ocation] *my_local_ptr`
  `Hardware watchpoint 1: -location *my_local_ptr`

Working with SMP

VCPU number limits (x86)

- Soft: 160
- Hard: 254
- Virtual CPUs > physical CPUs: lock-holder preemptions, slowdowns!



Model for gdb: VCPU = thread

- Switch VCPU via `thread` command
- Switches memory view as well!
- Do not try to debug user land this way...
- Note: monitor uses different "current VCPU" (see `cpu` command)

Triggering SMP races

- Play with number of VCPUs
- Enforce serializations via taskset
- Slow down execution by disabling KVM

Host- and Guest-side Tracing

Collect / retrieve guest traces via host
- gdb script (WIP)
- Paravirtual channel (WIP)
- Helpful if guest is unable to dump

Merged host/guest tracing
- Primary use: KVM debugging / optimizing
- ftrace instrumentation of KVM
- Trace infrastructure in QEMU
- Merge via stderr-trace > .../tracing/trace_marker

Can be useful for guest debugging as well
- Augment guest traces
  with (virtual) hardware events

SIEMENS

Python Helpers for Kernel Debugging

gdb 7 gained Python binding – let's use it!

- `(gdb) lx-symbols [module paths]`
  ```
  loading vmlinux
  scanning for modules in /data/linux/build-dbg
  loading @0xffffffffa0067000: /data/.../scsi/sr_mod.ko
  loading @0xffffffffa0055000: /data/.../mouse/psmouse.ko
  ```
- `(gdb) lx-dmesg`
  ```
  [    0.000000] Initializing cgroup subsys cpuset
  [    0.000000] Initializing cgroup subsys cpu
  [    0.000000] Linux version 3.1.0-dbg+ (jan@mchn199C.mch
  [    0.000000] Command line: root=/dev/sda2 resume=/dev/s
  ```
- `(gdb) p $lx_per_cpu("current_task", 3)`
  ```
  $1 = (struct task_struct **) 0xffff88003fc0b5c0
  ```
- `lx-tasks`, `$lx_current()`, `$lx_thread_info(task)`, …

Python Helpers for Kernel Debugging (2)

Not bound to QEMU/KVM setup

- kgdb
- Hardware debuggers with gdb support
- …

...but fast as hell this way – provided you...

- Reduce symbol look-ups
  - Cache `gdb.lookup_type()` results
  - `ptr.cast()` is faster than `gdb.parse_and_eval()`
- Bundle guest memory accesses

Helper plans

- ftrace buffer access
- ps-like process listing
- Results should be maintained in-tree (e.g. linux/scripts/gdb)
- Watch out for patches! (now really soon ☺)

Working Around gdb's x86 Limitations

Incomplete gdb register set

=> Use `monitor info registers`

gdb assumes x86 target arch = target mode

- Different remote protocols for 16/32 bit and 64 bit
- QEMU must switch arch on mode change
- gdb dislikes run-time changes

=> Avoid guest mode changes while gdb is attached!

But how to set early breakpoints then?

- Boot guest into desired mode
- Attach gdb
- Set <u>hardware</u> breakpoints in early code
- Reboot guest

Post mortem – crash Utility Support

Crash allows offline kernel analysis
- Reads kdump, netdump, diskdump, …
- Linux-specific inspection commands
- Command pass-through to embedded gdb core

Can read QEMU migration format
- Generated by migrate-to-file
- Triggered by libvirt dump
- Doesn't work with PCI pass-through (it's a hack...)

Better approaches
- Write out kdump from QEMU (WIP)
- Add kdump format support to gdb
- Use gdb helper scripts

Features to Come

KVM guest debugging on non-x86

- Freescale's Book E Power cores

Device state visualization

- Capture and dump individual emulated devices
- Guest driver stuck? IRQ line blocked?
- Alternative to `gdb qemu-system-$arch ...`
- On hold due to device addressability issues
- See last slide for git repository

gdb tracepoint support

- Tracepoint = collect data @breakpoint
- kprobe + ftrace or KGTP – without guest support
- Ongoing student project
- Future plan: make tracepoints light-weight
  - KVM in-kernel support, no user space exits
  - Only stop affected VCPU

Eva Kröcher, licensed under CC BY-SA-2.5

Needed gdb Enhancements

Decoupling of x86 architecture and operation mode

- Stable wire format will allow cross-mode debugging
- Overcome ugly QEMU workaround

Extended system register support

- x86: gdt, ldt, idt, tr, crX, MSRs, ...
- Some gaps also reported for PowerPC

x86 segmentation support

- Enable full BIOS / boot loader debugging
- Allow $(legacy_OS) debugging

Real multicore awareness

- Ongoing concept work regarding application debugging
- Extension for system-level debugging needed
  - Per-CPU virtual memory view

SIEMENS

Summary

- Reduced test turn-around times
- Test environments “to go”
- Source-level kernel & module debugging
- Safe driver or subsystem development
- Full machine state access
- Prototype device models
- Pass-through real devices
- ...

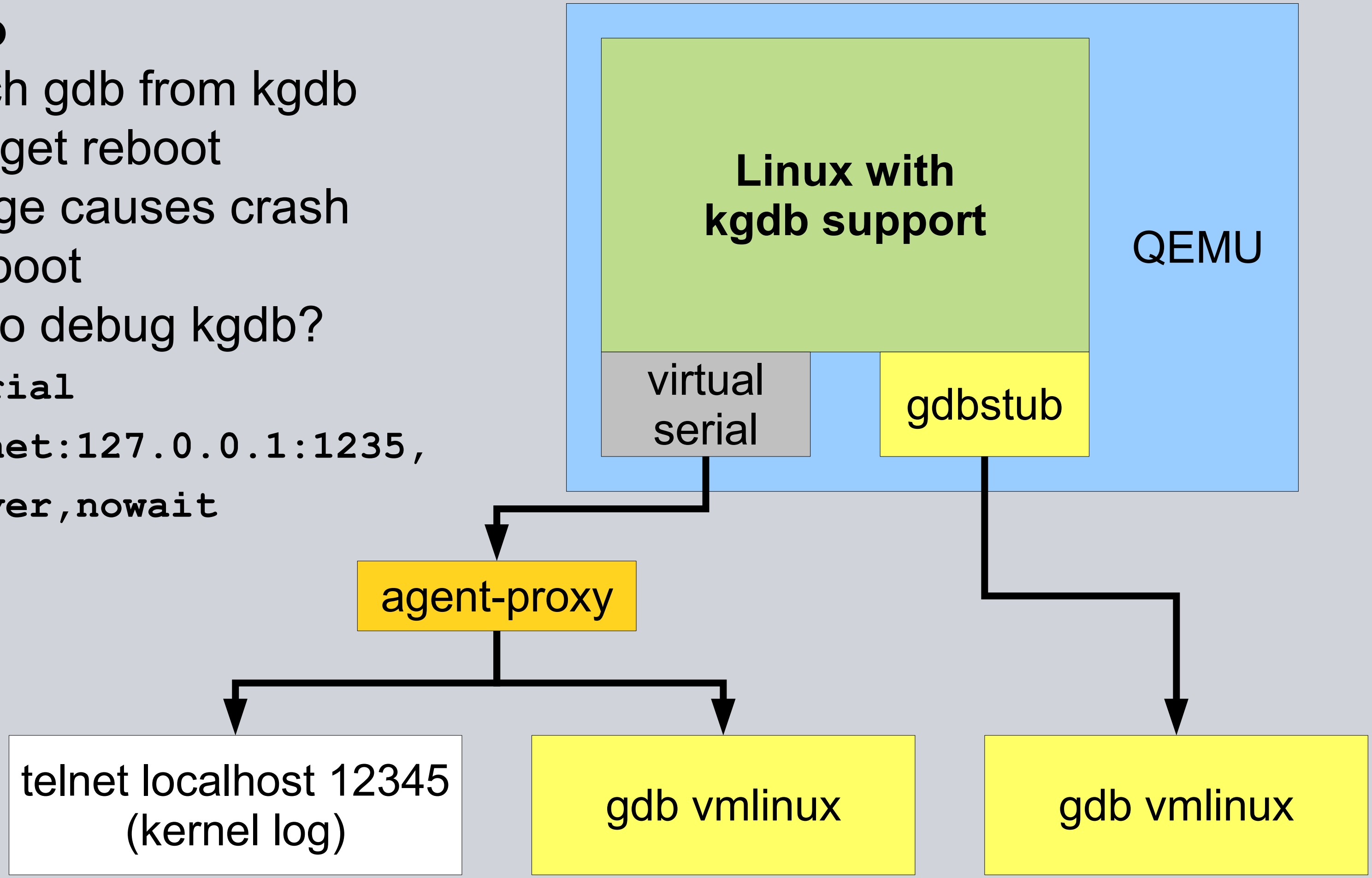
Thank you for listening!

Any questions?

Demonstration

Scenario

- Detach gdb from kgdb on target reboot
- Change causes crash on reboot
- How to debug kgdb?
  - `-serial telnet:127.0.0.1:1235,server,nowait`
  - `-s`



Resources

- www.linux-kvm.org
- wiki.qemu.org
- lkml.org/lkml/2011/11/5/83 (run-qemu.sh wrapper)
- sourceware.org/gdb/current/onlinedocs/gdb/Python-API.html
  (Python API for writing gdb helper scripts)
- git.kiszka.org/?p=qemu.git;a=shortlog;h=refs/heads/queues/device-show
  (device state visualization patches)