Reports of my ~~death~~ bloat have been greatly exaggerated

KVM Forum 2019

Paolo Bonzini, Red Hat
Sr. Principal Software Engineer

Or: How I learned to stop worrying and love QEMU

KVM Forum 2019

Paolo Bonzini, Red Hat

Sr. Principal Software Engineer

Why this talk?

- “Why are we investing in QEMU?”
- “I heard that QEMU is not secure”
- “Why do you even need a floppy disk controller?”

Is QEMU big?

Yes!

How big?

```
$ git ls-tree -r --full-name HEAD | awk '/.c$/ {print $4}'
```

- Excluding submodules, C files only:
  - ~2800 files, 1.650.000 lines of C code
  - Of these, 800 files and 150.000 lines are tests
- Also excluded:
  - Header files (~10% of C code)
  - Build and test scripts written in other language
  - Test data

Is QEMU **too** big?

Maybe!

Let’s see…

- Too big for some usecases or in general?
- Why do you care about size?
- Do you know how to measure size?
- Have you measured it?
- Is QEMU’s complexity essential?

QEMU can…

- … emulate other processors
- … emulate *your* processor
- … run old operating systems
- … run foreign Linux binaries
- … use KVM/HAX/HVF/WHPX for CPU virtualization

Know your usecase

- ~~… emulate other processors~~
- … emulate *your* processor
- … run old operating systems
- ~~… run foreign Linux binaries~~
- … use KVM/HAX/HVF/WHPX for CPU virtualization

Answer: `./configure --target-list=...`

Know your usecase

- ~~… emulate other processors~~
- ~~… emulate *your* processor~~
- … run old operating systems
- ~~… run foreign Linux binaries~~
- … use KVM/HAX/HVF/WHPX for CPU virtualization

Answer: `./configure --disable-tcg`

Is **your QEMU executable** too big?

And how does that affect you?

Why do you care?

- Attack surface
- Disk/memory footprint
- Startup time
- Number of bugs
- Customer support
- Cost of auditing for security

Attack surface

- Guest device drivers
- Management interface (QMP)
- Migration data
- Image formats
- ELF parsing
- VNC server
- Not all code is created equal

Vulnerabilities

- Of the top 100 vulnerabilities reported for QEMU:
  - 65 were not guest exploitable
  - 3 were not in QEMU :)
  - 5 did not affect x86 KVM guests
  - 3 were not related to the C language
  - Only 6 affected devices normally used for IaaS
- The most recent of these 6 was reported in 2016

Attack surface

- Have you secured your network?
- What data do your customers provide to you?
  - Kernel images
  - Disk images
  - VM snapshots (migration data)
- Are your guests sandboxed (SELinux, seccomp, ...)?
- Is your kernel up-to-date?

However!

- Developers want to hear from you!
- Patches are welcome, but suggestions are too!
- What code would you like to configure out?

Footprint and startup time

- Measured similarly: RSS, binary size, shared library count
- Let’s look at QEMU RSS:
  - moxie, -M none -display none: 21 MiB
  - moxie, -M moxiesim -accel qtest -display none: 21 MiB
  - x86_64, -S -M none -accel qtest -display none: 27 MiB
  - x86_64, -S -M pc -accel qtest -display none: 33 MiB
  - x86_64, -S -M pc -accel kvm -display none: 33 MiB
  - x86_64, -S -M pc -accel kvm -display gtk: 62 MiB

Footprint and startup time

- Always measure it!
  - RSS
  - Shared libraries
  - Time to first non-firmware instruction
- Beware of wrong assumptions
  - Text is shared across multiple VMs
  - Not all text in a shared library will be in memory
  - Firmware runs as fast as hardware (and less security sensitive)

Most of the memory footprint is shared

- This QEMU binary (3.1 from Fedora 30) is 12.5 MiB big
- It loads 99 shared libraries, for another 43.5 MiB
- Code that is never used never reaches memory

```
ldd /usr/bin/qemu-system-x86_64
  | awk 'NF>=3 {print $3}' |sort -u | xargs size
  | awk '{sum += $4} END {print sum}'
```

QEMU is already modular

- QEMU backends can be loaded from .so modules
- These link to 77 more shared libraries (215 MiB more!)

```
ldd /usr/lib64/qemu/*.so
  | awk 'NF>=3 {print $3}' |sort -u | xargs size
  | awk '{sum += $4} END {print sum}'
```

Dependencies can be configured out

- Full (default) build: 176 shared libraries
- Minimal build: 16 libraries, total size 19 MiB, RSS 16 MiB

libc.so.6
librt.so.1
libstdc++.so.6
libm.so.6
libgcc_s.so.1
libpthread.so.0
libutil.so.1
libgthread-2.0.so.0
libglib-2.0.so.0
libpcre.so.1
libnettle.so.4
libpixman-1.so.0
libseccomp.so.2
libaio.so.1
libnuma.so.1
libz.so.1

Bugs, auditing and customer support

- Know your environment!
  - Do your customers need SDL/GTK+ backends?
  - Do your customers need audio backends?
  - Which devices will be configured in your virtual machines?
- Example:
  - Target-specific boards and core devices
  - Shared devices: virtio, PCI, SCSI, ACPI
  - Backends: raw, qcow2, VNC

Creating custom configurations

- configure arguments for backends (and some features)
- default-configs/ files for boards and devices
  - Can be customized to remove boards and/or devices
  - Introduced in 2009
  - Revamped in 2019 with automatic dependencies (kconfig style)

Sample default-configs/i386-softmmu.mak

```
# Uncomment the following lines to disable these optional
# devices:
#CONFIG_AMD_IOMMU=n
#CONFIG_APPLESMC=n
#CONFIG_FDC=n
...
# Boards:
#
CONFIG_ISAPC=y
CONFIG_I440FX=y
CONFIG_Q35=y
CONFIG_MICROVM=y
```

Reduced default-configs/i386-softmmu.mak

```
CONFIG_MICROVM=y
CONFIG_SERIAL_ISA=y
CONFIG_WDT_IB700=y
```

```
CONFIG_VIRTIO_BALLOON=y
CONFIG_VIRTIO_BLK=y
CONFIG_VIRTIO_NET=y
CONFIG_VIRTIO_RNG=y
CONFIG_VIRTIO_SCSI=y
CONFIG_VIRTIO_SERIAL=y
```

- Remember to configure --without-default-devices!

Essential vs. accidental complexity

- Essential complexity: a property of the **problem** you are trying to solve
- Accidental complexity: a property of the **program** that solves the problem
- What seems accidental complexity to you now, may become essential tomorrow
- Or may already be essential

Accidental complexity

```
#define QEMU_GENERIC(x, ...) \
    QEMU_GENERIC_(typeof(x), __VA_ARGS__, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0)

/* There will be extra arguments, but they are not used.  */
#define QEMU_GENERIC_(x, a0, a1, a2, a3, a4, a5, a6, a7, a8, a9, count, ...) \
    QEMU_GENERIC##count(x, a0, a1, a2, a3, a4, a5, a6, a7, a8, a9)

/* Two more helper macros, this time to extract items from a parenthesized
 * list.
 */
#define QEMU_FIRST_(a, b) a
#define QEMU_SECOND_(a, b) b

/* ... and a final one for the common part of the "recursion".  */
#define QEMU_GENERIC_IF(x, type_then, else_)                                   \
    __builtin_choose_expr(__builtin_types_compatible_p(x,                      \
                                                       QEMU_FIRST_ type_then), \
                          QEMU_SECOND_ type_then, else_)
```

Essential complexity

- Concurrent I/O
- Serial port TLS
- Hotplug
- Stable CPU models after hardware upgrade
- Stable hardware models after VMM upgrade
- Live migration
- Boot a distribution kernel

What’s next?

Multi-process split

- vhost-user as the sanctioned multi-process interface
- Out-of-process block layer
  - Performance improvements
  - Finer-grained seccomp filters

Easier configuration

- List what is enabled by default
  - PCI devices
  - virtio devices
  - On-board devices
- Text file configuration of host components

Documentation

- QEMU 4.0: initial port of documentation to Sphinx
- Work in progress to reorganize and rethink the manual
- Document best practices for running QEMU securely

Conclusions

- Know your usecase
- Know your customer
- Talk to the developers

Thank you

in linkedin.com/company/red-hat

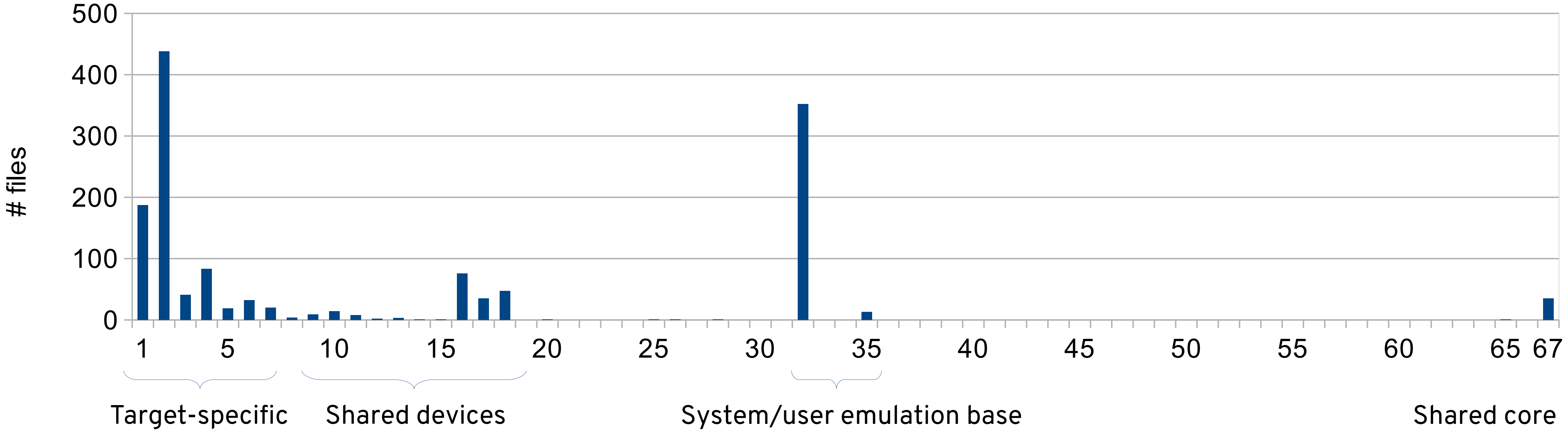
▶ youtube.com/user/RedHatVideos

f facebook.com/redhatinc

twitter.com/RedHat

How much code is shared across targets?

- Look at linker command lines
- Associate object files to executables, count occurrences


# files
Target-specific
Shared devices
System/user emulation base
Shared core