Is QEMU too complex
and what can we do about it?

KVM Forum 2021

Paolo Bonzini, Red Hat
Distinguished Engineer

Red Hat

Betteridge's law of headlines

"Any headline that stion mark can be an word no."

Red Hat

Is QEMU complex?

Yes

Is QEMU **too** complex?

Yes (but everything is)

Why is complexity a problem?

- More bugs (even if not security-sensitive)
- Code is harder to review
- Contributing is more difficult

Essential complexity

- What makes QEMU complex
- A property of the **problem** you are trying to solve

Essential complexity

- Guest devices (emulation, live migration)
- Management interface (monitor)
- Storage management
- Network servers (VNC, NBD)
- Portability
- Configurability

Essential complexity

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

Also essential complexity

- “Human” monitor
- “Easy” options for command line use
- Disassembler
- Multiple accelerators
- Object model + marshaling/unmarshaling
- GUI

Complexity of tools

- Both internal and external!
- Make common tasks easier vs. making debugging harder
- Examples:
  - Build system (configure+Meson vs. configure+rules.mak)
  - Configuration management (kconfig)
  - Code generation (QAPI)

Accidental complexity

- Makes QEMU too complex
- A property of the **program** that solves the problem

Fighting accidental complexity?

- Understand complexity
  - Know essential complexity
  - Know **the sources** of accidental complexity
- Keep complexity in mind when making changes
  - Use essential complexity to your advantage
  - Watch out for accidental complexity, don’t let it take over
- This is the reviewer’s job!

Signs of accidental complexity

- Incomplete transitions
  - A new way to do the same thing
  - Features supported only by a few targets/devices
- Duplicated logic
  - Missing abstractions
  - Excessive ad hoc code

A new way to do the same thing

- Error reporting (Error* vs. error_report)
- Board modeling (e.g. QOM child objects)
- Live migration (VMStateDescription vs. vmstate_register)
- QEMUTimer vs. QEMUTimer*
- Configure vs. meson

Automated conversions with Coccinelle

timer_del(mytimer); timer_free(mytimer);	timer_free(mytimer);

if (!foo(..., &err)) {
  error_propagate(errp, err);
  return;
	if (!foo(..., errp)) {
	  return;

sysbus_init_child_obj(parent, propname, \
  child, sizeof(T), type)
	sysbus_init_child_obj(parent, propname, \
	  child, sizeof(*child), type)

object_new/isa_create/pci_create qdev_init_no_fail	qdev_new/isa_new/pci_new qdev_realize/qdev_realize_and_unref/...

Incomplete support for features

- I/O error reporting (rerror/werror)
- I/O accounting (query-blockstats)
- Asynchronous I/O (block or character devices)
- No real solution, you just have to put in the work

Fear of incomplete transitions?

- Incomplete transitions are not always bad
  - ... if the new feature requires a transition period anyway
  - ... if the old API affects the command line
- Work in phases
  - Identify the smallest amount of work that is an improvement
  - Plan for what comes later
  - Incomplete transitions should not deter from improving QEMU!

Summary

- Do not be afraid of transitions—but make a plan
- Ensure good test coverage
- Learn Coccinelle

Signs of accidental complexity

- Incomplete transitions
  - A new way to do the same thing
  - Features supported only by a few targets/devices
- Duplicated logic
  - Missing abstractions
  - Excessive ad hoc code

Duplicated logic

- Missing abstractions
  - Example: dirty page handling in display emulation

```
$ git grep -w memory.*dirty hw/display

hw/display/bochs-display.c:        snap = memory_region_snapshot_and_clear_dirty(&s->vram,
hw/display/bochs-display.c:            dirty = memory_region_snapshot_get_dirty(&s->vram, snap,

hw/display/cg3.c:        snap = memory_region_snapshot_and_clear_dirty(&s->vram_mem, 0x0,
hw/display/cg3.c:            update = memory_region_snapshot_get_dirty(&s->vram_mem, snap, page,

hw/display/exynos4210_fimd.c:            snap = memory_region_snapshot_and_clear_dirty(w→mem_section.mr,
hw/display/exynos4210_fimd.c:                is_dirty = memory_region_snapshot_get_dirty(w->mem_section.mr,

hw/display/framebuffer.c:    snap = memory_region_snapshot_and_clear_dirty(mem, addr, src_width * rows,
hw/display/framebuffer.c:        dirty = memory_region_snapshot_get_dirty(mem, snap, addr, src_width);

hw/display/g364fb.c:    return memory_region_snapshot_get_dirty(&s->mem_vram, snap, page, G364_PAGE_SIZE);
hw/display/g364fb.c:    snap = memory_region_snapshot_and_clear_dirty(&s->mem_vram, 0, s->vram_size,

hw/display/macfb.c:    return memory_region_snapshot_get_dirty(&s->mem_vram, snap, addr, len);
hw/display/macfb.c:    snap = memory_region_snapshot_and_clear_dirty(&s->mem_vram, 0x0,

hw/display/sm501.c:    snap = memory_region_snapshot_and_clear_dirty(&s->local_mem_region,
hw/display/sm501.c:        update |= memory_region_snapshot_get_dirty(&s->local_mem_region, snap,

hw/display/tcx.c:    ret = memory_region_snapshot_get_dirty(&s->vram_mem, snap, addr, len);
hw/display/tcx.c:        ret |= memory_region_snapshot_get_dirty(&s->vram_mem, snap,
hw/display/tcx.c:        ret |= memory_region_snapshot_get_dirty(&s->vram_mem, snap,
hw/display/tcx.c:    snap = memory_region_snapshot_and_clear_dirty(&ts->vram_mem, 0x0,
hw/display/tcx.c:    snap = memory_region_snapshot_and_clear_dirty(&ts->vram_mem, 0x0,

hw/display/vga.c:        snap = memory_region_snapshot_and_clear_dirty(&s->vram, region_start,
hw/display/vga.c:            update = memory_region_snapshot_get_dirty(&s->vram, snap,
hw/display/vga.c:            update |= memory_region_snapshot_get_dirty(&s->vram, snap,
hw/display/vga.c:            update = memory_region_snapshot_get_dirty(&s->vram, snap,
```

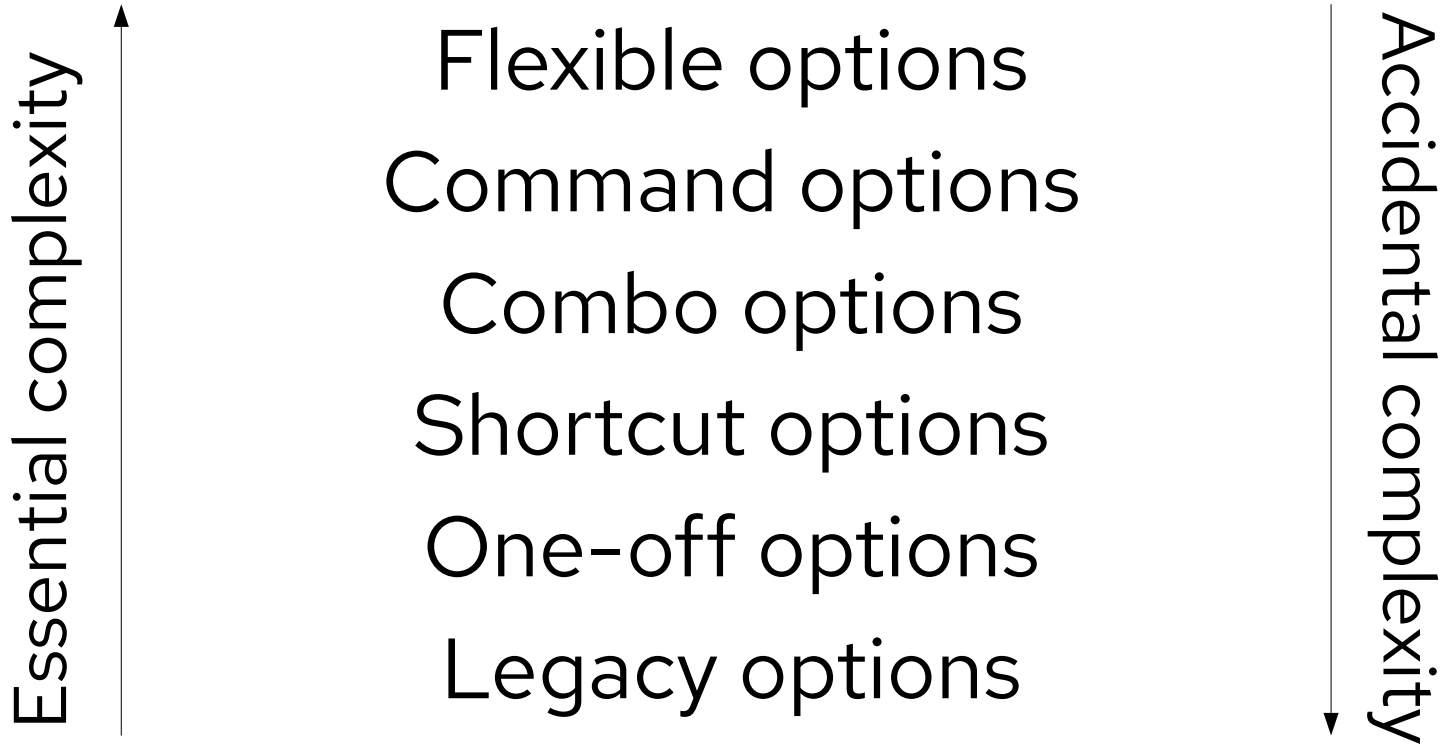
Duplicated logic

- Missing abstractions
  - Example: dirty page handling in display emulation
  - New abstractions may become incomplete transitions
- Tradeoffs: ad hoc code vs. data structures
  - Manual parsing with sscanf vs. QemuOpts/keyval
  - Scattered tables and functions vs. modinfo
- Excessive duplication requires a transition plan

Case study: command line

- 117 command line options, ~3000 lines of code
  - Essential complexity: some
  - Accidental complexity: too much
- How to not make things worse?
- How to simplify things?

Case study: command line



Flexible options

- Back-end: -accel, -blockdev, -chardev, -display, -netdev, -object
- Front-end: -cpu, -device, -machine, -mon
- Delegated via function pointers, QOM classes, etc.
  - Often no need to touch command line parsing code!
  - Main mechanism for new features
- Too many parsers: QemuOpts, keyval, JSON, bespoke

Flexible options

- Back-end: -accel, -blockdev, -chardev, -display, -netdev, -object
- Front-end: -cpu, -device, -machine, -mon
- Delegated via function pointers, QOM classes, etc.
  - Often no need to touch command line parsing code!
  - Main mechanism for new features
- Too many parsers: QemuOpts, keyval, JSON, bespoke

Command options

- -S, -add-fd, -action, -loadvm, -plugin, -trace
- Small burden if 1:1 mapping to QMP
- High bar for adding new ones

Combo options

- Block device: -drive
- Character device: -serial, -monitor, -qtest, ...
- Other: -nic, -net, -virtfs
- Useful, but very high burden
- Worst modularity

Shortcut options

- -accel: -enable-kvm
- -action: -no-reboot, -no-shutdown
- -display: -sdl, -full-screen, -curses, -nographic
- -drive: -cdrom, -hdX, -pflash, -sd, ...
- -machine: -kernel, -nographic, -no-acpi, -smp, -usb,...
- Good modularity
- Smallest burden, but do not add more

One-off options

- Machine: -acpitable, -boot, -m, -option-rom, -rtc,...
- Developer: -d/-D, -L
- Environment: -msg, -name, -compat
- Backend: -spice
- Mix: -overcommit
- **Transform** if possible **into property shortcut options**
- Avoid creating new ones, prefer QMP+command options

Legacy options

- Configuration: -alt-grab, -ctrl-grab, -echr, -portrait, -rotate...
- Wannabe management: -daemonize, -pidfile, -runas, -chroot
- Failed experiments: -readconfig, -writeconfig, -no-user-config
- Deprecate/remove, or transition to shortcut options

Do not design in a void!

- Do you really need a new flag? Command-line parsing is already integrated with:
  - QOM object and properties
  - QAPI structs
  - QMP commands
- Advantage: improved modularity, all code in one place
- Exploit existing interactions between subsystems

~~There should be one obvious way to do it~~

There should be one documented way to do it

Summary

- Do not be afraid of transitions—but make a plan
- Ensure good test coverage
- Learn Coccinelle
- Evaluate tradeoffs, but don’t make things worse!
- Know essential complexity, exploit it to your advantage
- Document best practices

Thank you

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat

Red Hat