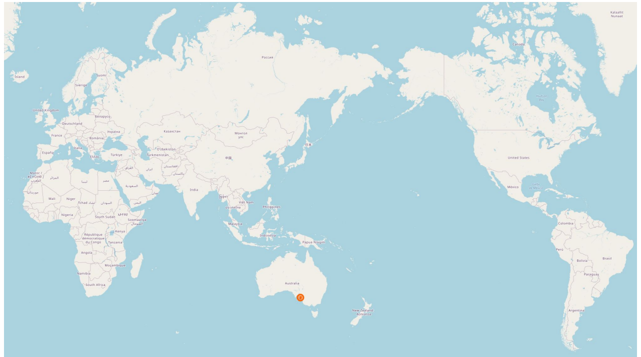
QEMU and OpenBMC

Joel Stanley
joel@jms.id.au / @shenki

Your Presenter

- Free and Open Source (FOSS) developer
- Live in Adelaide, Australia
- Long distance runner
- Cricket fan, Adelaide Oval member
- Worked on OpenBMC at IBM since 2015
- Maintain OpenBMC Linux and u-boot
- Upstream ASPEED kernel maintainer



OpenBMC

QEMU and OpenBMC

- What is QEMU
- Use case 1: Booting a custom kernel
- Use case 2: Running userspace tool
- Use case 3: Testing PGOOD
- Developing for QEMU
- Next steps

OpenBMC

What is QEMU

- “QEMU is a generic and open source machine emulator and virtualizer”
- Two main usecases: virtualisation (KVM) and **emulation**
- System Emulation
  - ARM CPU
  - System on Chip peripherals
  - External devices: I2C, GPIO, Ethernet, FSI, LPC, eMMC, SPI

- (Also user emulation: running binaries from a different architecture. We’re not talking about that today)

QEMU in OpenBMC

- Cedric, Andrew, Jeremy, Rashmica and Joel at IBM have created an ASPEED model
  - AST2400, AST2500, AST2600 emulation
  - Used for pre-silicon bringup of AST2600
  - Coverage of most periperhals used by OpenBMC
  - Upstream: apt install qemu-system-arm
- Can boot full OpenBMC system images
- Used in OpenBMC CI

Demo

AST2600 with direct kernel boot and provided initrd

```
qemu-system-arm -M tacoma-bmc -nic user -nographic
-kernel aspeed-g5-dev/arch/arm/boot/zImage \
-dtb aspeed-g5-dev/arch/arm/boot/dts/aspeed-bmc-opp-tacoma.dtb \
-initrd ~/dev/kernels/misc/broomstick-v3.cpio.xz
```

AST2500 booting from SPI NOR

```
qemu-system-arm -M romulus-bmc -nic user -nographic \
-drive file=flash-romulus,if=mtd,format=raw
```

AST2600 with SD card and direct kernel boot

```
qemu-system-arm -M tacoma-bmc -nic user -nographic \
-kernel aspeed-g5-dev/arch/arm/boot/zImage \
-dtb aspeed-g5-dev/arch/arm/boot/dts/aspeed-bmc-opp-tacoma.dtb \
-drive file=obmc-phosphor-image-witherspoon-tacoma.wic,if=sd,index=2,format=raw
```

OpenBMC

Use case 1: Booting a custom kernel

- Kernel development workflow
- Build a kernel, point qemu at it, and get results quickly

```
apt install gcc-arm-linux-gnueabi qemu-system-arm
wget https://is.gd/romulus_cpio -O obmc-phosphor-initramfs-romulus.cpio.xz
wget https://is.gd/flash_romulus -O flash-romulus
export ARCH=arm CROSS_COMPILE=gcc-arm-linux-gnueabi
git clone https://github.com/openbmc/linux openbmc-linux && cd openbmc-linux
```

- Docker

```
docker run -it shenki/openbmc-qemu
cd /build/openbmc-linux
make aspeed_g5_defconfig
make -j`nproc`
qemu-system-arm -M romulus-bmc -nic user -nographic \
        -kernel arch/arm/boot/zImage \
        -dtb arch/arm/boot/dts/aspeed-bmc-opp-romulus.dtb \
        -initrd ../obmc-phosphor-initramfs-romulus.cpio.xz
        -drive file=../flash-romulus,if=mtd,format=raw
```

- To exit, ctrl+a x

Use case 1: Booting a custom kernel (cont)

- I have a bug!
- gdb can debug the running kernel
  - -s: gdb server listens on localhost, port 1234
  - -S: do not start execution

```
docker run --name=myqemuimg -it shenki/openbmc-qemu
cd /build/openbmc-linux
qemu-system-arm -M romulus-bmc -nic user -nographic \
        -kernel arch/arm/boot/zImage \
        -dtb arch/arm/boot/dts/aspeed-bmc-opp-romulus.dtb \
        -initrd ../obmc-phosphor-initramfs-romulus.cpio.xz \
        -drive file=../flash-romulus,if=mtd,format=raw -S -s
```

- Start gdb

```
docker exec -it myqemuimg
cd /build/openbmc-linux
gdb-multiarch -s vmlinux ‘target remote localhost:1234’
0x80000000 in ?? ()
(gdb) c
ctrl+c
(gdb) bt
```

OpenBMC

Use case 2: Running userspace tool

- We can copy a binary into QEMU and run it
- Start QEMU with a port forward from 2222 to 22 (SSH)
  - This means localhost:2222 will connect to port 22 on our emulated BMC

```
docker run --name=myqemuimg -it shenki/openbmc-qemu
qemu-system-arm -M romulus-bmc -nic user,hostfwd=::2222-:22 \
        -nographic \
        -drive file=flash-romulus,if=mtd,format=raw
```

- Build a simple program and scp it to the emulated BMC

```
docker exec -it myqemuimg
echo "main() {printf(\"Hello, OpenBMC!\\n\");}" > hello.c && make hello
scp -p2222 hello root@localhost:
root@localhost's password:
hello                   100%    0       0.0KB/s   00:00
ssh -p2222 root@localhost
root@localhost's password:
root@romulus:~# hello
Hello, OpenBMC!
```

- With this in our toolkit, we can build applications using the SDK and run them in an emulated BMC environment

Use case 3: Testing PGOOD

- Scenario:
  - OpenPower systems have the concept of PGOOD, a signal that tells the BMC that the system is ready for the host CPU to be powered on
  - This is a GPIO input to the BMC
  - We can manipulate GPIOs using the QEMU monitor
- Once QEMU has booted the machine press `ctrl+a c` to switch to the monitor

```
docker run --name=myqemuimg -it shenki/openbmc-qemu
qemu-system-arm -M romulus-bmc -nic user,hostfwd=::2222-:22 \
        -nographic \
        -drive file=flash-romulus,if=mtd,format=raw
…
Phosphor OpenBMC (Phosphor OpenBMC Project Reference Distro) 0.1.0 romulus ttyS4

romulus login: root
Password:
# obmcutil power
pgood = 0
state = 0
pgood_timeout = 10
ctrl+a c
(qemu)
```

Use case 3: Testing PGOOD

- QOM is the QEMU Object Model
  - We can manipulate properties of objects using the QOM APIs
  - This allows us to manipulate the BMC state

```
(qemu) help qom-get
qom-get path property -- print QOM property
(qemu) help qom-set
qom-set [-j] path property value -- set QOM property.
                 -j: the value is specified in json format.

(qemu) qom-get gpio gpioD2
False
(qemu) qom-set gpio gpioD2 true
(qemu)

ctrl+a c

root@romulus:~# obmcutil power
pgood = 1
```

Developing QEMU

- QEMU is written in C
- The development model is close to the kernel
  - git, mailing list code review, pull requests by maintainers
- Modeling involves
  - Providing read() and write() callbacks for the address space a device occupies
  - Modelling logic such that the emulated code behaves as it does on hardware
  - Surprisingly minimal in many cases
- `hw/misc/tmp105.c` models temperature sensor i2c devices
- A simple model can involve dumping the default state of a device under linux, and putting it in an array
  - `i2cdump`
  - https://github.com/shenki/qemu/blob/dps310/hw/misc/dps310.c#L249
  - This is how I developed the upstream dps310 iio kernel driver

Next Steps

- Try using qemu to test your userspace change
- Write a qemu model for a i2c sensor your board has
- Add more qemu booting to CI

- IBM team continues to work on modelling
  - eMMC boot and secure boot is a work in progress
  - Simple FSI modelling
  - Proof of concept connecting “host” QEMU to BMC QEMU
- Google submitted model for Nuvoton NPCM730

Resources

- https://github.com/openbmc/qemu/wiki/
- https://github.com/openbmc/qemu/wiki/Usage

- https://github.com/shenki/openbmc-qemu-tutorial

- https://www.qemu.org/docs/master/system/arm/aspeed.html

- https://wiki.qemu.org/Documentation