Qemu for System Software Development

(Slides) (Chatlog)

- Programmatic control of QEMU
- Read/Write Registers & Memory
- Writing new GDB Commands in Python

Prebuilt Toolchains

- GCC ARM Embedded: bare metal Cortex-M, A and R
- Linaro: I use the Linux glibc variant of their toolchain

The “Old Way” of Using QEMU

- 1-add
- 2-hello-semihosting
    - “`__semhost()`” is implemented using GCC Extended Asm
        - __asm__(“...” : *outputs* : *inputs* : *clobbers*)
    - “`-semihosting-config target=gdb`” uses GDB Remote Protocol: Host I/O Packets to send output to the attached debugger.

Programmatically Control QEMU

QEMU API

- qemu/qapi-schema.json
- QOM exegesis and apocalypse by Paolo Bonzini
- QEMU API client: qemu/scripts/qmp/qmp

GDB Python API

- See Scott's answers to GDB related questions on StackOverflow for more examples

Problem: We want the QEMU console

Workaround: run it inside `tmux`:

```
$ tmux -S ~/t.sock new-session -d -s armv7a0 sleep 1
$ tmux -s ~/t.sock set-option -t armv7a0:0 remain-on-exit 1
$ tmux -S ~/t.sock attach-session
```

GDBServer Control

HMP
(QMP) **gdbserver unix:./g.sock,server,nowait**
(QMP) **gdbserver none**

Semihosting: can only be activated through command line options

State

in vl.c

```
typedef struct SemihostingConfig {
    bool enabled;
    SemihostingTarget target;
    const char **argv;
    int argc;
    const char *cmdline; /* concatenated argv */
} SemihostingConfig;

static SemihostingConfig semihosting;
```

`setmihosting` has getters but not setters. getters used in arm-semi.c etc.

Command line Interface

-semihosting-config arg=hello
-semihosting-config arg=world

semihosting_arg_fallback() uses -kernel and -append

System Control Registers

CP15 Access under GDB

UART on Virt-2.7

```
/machine/unattached/device[5]/pl011[0]
  addr: 0x09000000 (uint64)
  type: qemu:memory-region (string)
  container: /machine/unattached/system[0] (link<qemu:memory-region>)
  priority: 0 (uint32)
  size: 4096 (uint64)
```

fw_cfg

QEMU Firmware Configuration (fw_cfg) Device

```
(QEMU) info roms
        addr=0000000040000000 size=0x010000 mem=ram name="dtb"
        /rom@etc/acpi/tables size=0x200000 name="etc/acpi/tables"
        /rom@etc/table-loader size=0x000880 name="etc/table-loader"
        /rom@etc/acpi/rsdp size=0x000024 name="etc/acpi/rsdp"
```

How Other People Use QEMU

Viller

```
qemu-system-arm \
-M vexpress-a15 \
-kernel buildroot/output/images/zImage \
-dtb buildroot/output/images/vexpress-v2p-ca15_a7.dtb \
-drive file=buildroot/output/images/rootfs.ext2,if=sd \
-smp 2 \
-s \
-serial stdio \
-append "root=/dev/mmcblk0 console=ttyAMA0,115200n8" \
-net nic,vlan=1 \
-net user,vlan=1,hostfwd=udp:127.0.0.1:6669-:69
```

Wen

```
qemu-system-arm \
-M versatilepb \
-kernel /tmp/kernel/linux-stable/arch/arm/boot/zImage \
      -drive file=output/images/rootfs.ext2,if=scsi,format=raw \
      -append "root=/dev/sda console=ttyAMA0,115200" \
      -serial stdio \
      -net nic,model=rtl8139 \
      -net user

Virt-2.7
      qemu-system-arm \
      -M virt-2.7 \
      -S \
      -m 1024 \
      -cpu cortex-a15 \
      -nographic \
      -device virtio-9p-device,fsdev=host_fs,mount_tag=/dev/root \
      -fsdev local,id=host_fs,security_model=none,path=$(pwd)/../sysroot \
      -kernel ./arch/arm/boot/zImage \
      -append 'root=/dev/root rootfstype=9p rootflags=trans=virtio rw \
      -netdev user,id=unet -device virtio-net-device,netdev=unet
```

Why favor “virt” and not “vexpress”?

- Shares I/O path with KVM.
    - Battle tested.
- More virtio channels
- Less irrelevant hardware equals more low memory
- It’s often the first machine supported for a new CPU architecture (e.g. aarch64)

ARM Peripherals

- SP804 Dual-Timer Module ("two programmable 32/16-bit down counters that can generate interrupts on reaching zero.")
- PL031 RTC
- PL041 audio codec
- PL061: GPIO
- PL111 LCD controller
- PL181 MMCI, Multimedia Card Interface

Versatile Express Board Specs

- Vexpress A15x2