CSE 506 Operating Systems - Homework Assignment
Running Boot Disk on Virtual USB through QEMU

Ankit Dewan and Shashank Rao
Stony Brook University

I. PROBLEM STATEMENT

QEMU (short for Quick Emulator) is a free and open based virtual machine monitor that is used to perform hardware virtualization. In CSE 506, we use QEMU in order to emulate a bare-metal machine, which is capable of running the operating system kernel developed by us, so that we can test and debug our code. On compiling our code, we are provided with two image files - boot disk($USER.img) and data disk($USER-data.img). Currently, for running the developed OS, we load the boot and data disks both on to the virtual AHCI disks emulated by QEMU. Our problem statement was to get the QEMU to run our developed code, by loading the boot disk on virtual USB and data disk on the first virtual AHCI disk.

II. IMPLEMENTATION OF SOLUTION

The first step that we took was to understand the original script that we used to run on QEMU which loaded both boot and data disks on the virtual AHCI disks. The following is the script used:

```
qemu-system-x86_64 -curses /
-drive id=boot,format=raw, file =$USER.img,if=none /
-drive id=data,format=raw, file =$USER-data.img,if=none /
-device ahci,id=ahci /
-device ide-drive,drive=boot,bus=ahci.0 /
-device ide-drive,drive=data,bus=ahci.1 /
-gdb tcp::9999
```

This command script will run the *qemu-system-x86_64* executable and pass all the rest of the script as parameters. The *curses* parameter is the display interface to be used. The next two lines create two QEMU block devices (drives). The first drive is created with its ID as 'boot' (*id=boot*), using RAW format (*format=raw*); this drive is linked with the boot disk (*file=$USER.img*), and, this drive is not linked to any interface (*if=none*). Similarly, the second drive is created with its ID as 'data', and this drive is linked to the data disk (*file=$USER-data.img*). The next line creates the AHCI device with its ID as 'ahci'. Creation of this device also creates a bus, and individual drives can be connected to this device bus by using {device_id}.{drive_no}. This is done in the next two lines - where two ide-drive devices (or hard disks) are loaded with the boot and data disk images and connected to the AHCI device bus as device 0 and device 1 respectively. The final gdb command enables gdb for debugging in the QEMU.

This command when run, first calls the main function in *vl.c* file. On analyzing the source code present in the *vl.c* file, we observed that at first the cpu and modules are initialized (*qemu_init_cpu_list()*, *qemu_init_cpu_loop()* and *module_call_init()*) followed by call to the *qemu_add_opts()* function and *qemu_add_drive_opts()* function which will add the required drives and options for a particular machine, which in our case is x86_64. The different parameters passed in the script is then parsed next to determine the devices and drives the QEMU has to emulate. In particular, the definitions for *QEMU_OPTION_drive* and *QEMU_OPTION_device* defined in *qemu-options.hx* file is read and calls are made to *qemu_find_opts()* and *qemu_opts_parse_noisily()*, so that the QEMU can understand that it has to emulate the those particular devices. Further analysis of the main function in *vl.c* file, led us to observe that the code was present to emulate a virtual USB device. The '*-usb*' switch is used to create the UHCI controller and a USB 1.1 bus.

A host controller interface (HCI) is a register-level interface that enables a host controller for USB or IEEE 1394 hardware to communicate with a host controller driver in software [1]. UHCI or Universal Host Controller Interface was created by Intel as an implementation of the USB 1.0 host controller interface [2]. The UHCI specification defines a set of I/O mapped registers that allow communication between the controller and the operating system [2]. The Enhanced Host Controller Interface (EHCI) is a high-speed controller standard applicable to USB 2.0. UHCI-based systems [1]. Originally a PC providing high-speed ports had two controllers, one handling low- and full-speed devices and the second handling high-speed devices [1]. The UHCI driver provided low- and full-speed functions using an Intel or via chipset's USB host controllers on the motherboard [1]. The EHCI driver provided high-speed functions for USB ports on the motherboard or on the discrete USB controller [1]. More recent hardware routes all ports through an internal "rate-matching" hub (RMH), and the EHCI controller indirectly provides full and low speed USB functions [1].

The '*-usb*' switch is used to create the UHCI controller and a USB 1.1 bus. The '*-device*' switch is used along with this to add an EHCI controller to the virtual machine. The *QEMU_OPTION_usb* defined in *qemu-options.hx* file is used to enable the usb driver when '*-usb*' switch is read in the command script from the main in *vl.c* file. When used with the '*-device usb-ehci*' switch, we can emulate the USB 2.0 functionality. We then tried the following command script which also worked:

```
qemu-system-x86_64 -curses /
```

```
−drive id=boot,format=raw, file =$USER.img,if=none /
−drive id=data,format=raw, file =$USER−data.img,if=none /
−usb /
−device usb−ehci,id=ehci1 −device
      usb−storage,bus=ehci1.0,drive=boot /
−device usb−ehci,id=ehci2 −device
      usb−storage,bus=ehci2.0,drive=data /
−gdb tcp::9999
```

usb-storage emulates a virtual USB storage device; thus, here in this script, the boot and data disks are loaded onto the virtual USB storage devices and attached to the EHCI controller. However, in this command script, we are loading both the boot and data disks on the virtual USB, which is not as per our requirement (we required boot disk on virtual USB and data disk on first AHCI disk). In order to boot the QEMU as per our requirements, we tried the following command script:

```
qemu−system−x86_64 −curses /
−drive id=boot,format=raw, file =$USER.img,if=none /
−drive id=data,format=raw, file =$USER−data.img,if=none /
−usb −device usb−ehci,id=ehci −device
      usb−storage,bus=ehci.0,drive=boot /
−device ahci,id=ahci −device ide−drive,bus=ahci.1, drive=data /
−gdb tcp::9999
```

However, this command script failed in execution giving us 'No bootable device.' error. On further analysis of the source code and QEMU boot sequence, we realized that we had to specify the boot sequence in which the QEMU starts up. By default the QEMU SeaBIOS only checks for three devices - the Hard disk (AHCI disk), the floppy drive (cd-rom) and network drives, in that order. As a result, the boot disk, which is present in USB device is not included during bootloading. We had to include the *bootindex={boot-order-no}* as a parameter in the command. We ran the following script:

```
qemu−system−x86_64 −curses /
−drive id=boot,format=raw, file =$USER.img,if=none /
−drive id=data,format=raw, file =$USER−data.img,if=none /
−usb −device usb−ehci,id=ehci −device
      usb−storage,bus=ehci.0,drive=boot,bootindex=1 /
−device ahci,id=ahci −device
      ide−drive,bus=ahci.1, drive=data,bootindex=2 /
−gdb tcp::9999
```

which worked perfectly as well as met all our requirements. On including the bootindex=1 parameter for USB device and bootindex=2 parameter for AHCI disk, USB device becomes the first device checked during bootloading followed by AHCI disk. As a result, it is able to load the USB boot disk first followed by the AHCI data disk.

III. CONCLUSIONS

We were successful in running the boot disk on virtual USB and data disk on the first AHCI controller hard disk as per the requirement using the command script:

```
qemu−system−x86_64 −curses /
−drive id=boot,format=raw, file =$USER.img,if=none /
−drive id=data,format=raw, file =$USER−data.img,if=none /
−usb −device usb−ehci,id=ehci −device
      usb−storage,bus=ehci.0,drive=boot,bootindex=1 /
−device ahci,id=ahci −device
      ide−drive,bus=ahci.1, drive=data,bootindex=2 /
−gdb tcp::9999
```

This assignment helped us in understanding the working of QEMU, since we had to go through its source code in order to understand what parameters to use in the command script. We got to learn how QEMU started up as well as how different parameters impact its working. It also helped us in improving and building on our knowledge base which we had gained during Warmup Project 3 with regard to the subject of AHCI disks.

REFERENCES

[1] https://en.wikipedia.org/wiki/Host_controller_interface_(USB,_Firewire)
[2] http://wiki.osdev.org/Universal_Host_Controller_Interface
[3] https://www.qemu.org/documentation/
[4] https://qemu.weilnetz.de/doc/qemu-doc.html
[5] https://wiki.qemu.org/Main_Page
[6] https://wiki.qemu.org/Category:User_documentation
[7] https://wiki.qemu.org/Documentation/Migration_with_shared_storage
[8] https://github.com/qemu/
[9] https://github.com/qemu/qemu/blob/master/docs/qdev-device-use.txt
[10] https://github.com/qemu/qemu/blob/master/docs/usb-storage.txt
[11] https://github.com/qemu/qemu/blob/master/docs/usb2.txt
[12] https://github.com/qemu/qemu/blob/master/docs/bootindex.txt
[13] https://github.com/qemu/qemu/blob/master/docs/qemu-block-drivers.texi
[14] https://wiki.gentoo.org/wiki/QEMU/Options
[15] http://nairobi-embedded.org/debugging_qemu_with_gdb_ddd.html
[16] https://unix.stackexchange.com/questions/250938/qemu-usb-passthrough-windows-guest
[17] https://www.intel.com/content/www/us/en/io/universal-serial-bus/ehci-specification.html
[18] https://lists.gnu.org/archive/html/qemu-devel/2011-01/msg03216.html
[19] https://wiki.qemu.org/Documentation/Platforms/S390X
[20] https://rwmj.wordpress.com/2011/10/12/tip-debugging-the-early-boot-process-with-qemu-and-gdb/
[21] https://www.seabios.org/Runtime_config
[22] https://mail.coreboot.org/pipermail/seabios/2010-October/001041.html
[23] https://askubuntu.com/questions/190929/how-do-i-disable-unwanted-ipxe-boot-attempt-in-libvirt-qemu-kvm
[24] https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=740464
[25] https://www.redhat.com/archives/libvir-list/2013-November/msg01273.html
[26] https://en.wikipedia.org/wiki/SeaBIOS
[27] https://ubuntuforums.org/showthread.php?t=2322683
[28] http://willhaley.com/blog/simple-portable-linux-qemu-vm-usb/
[29] http://www.helenos.org/wiki/UsersGuide/RunningInQEMU
[30] https://bugzilla.redhat.com/show_bug.cgi?id=789621
[31] https://rubenerd.com/sata-on-qemu/
[32] https://discourse.redox-os.org/t/cannot-boot-iso-in-qemu/186
[33] https://wiki.qemu.org/Features/Q35
[34] https://www.redhat.com/archives/libvir-list/2011-September/msg01097.html
[35] https://bbs.archlinux.org/viewtopic.php?id=209828
[36] https://bugs.launchpad.net/ubuntu/+source/qemu-kvm/+bug/576086
[37] https://wiki.ubuntu.com/QemuDiskHotplug
[38] https://www.linux-kvm.org/page/USB
[39] https://lists.gnu.org/archive/html/qemu-devel/2017-05/msg06394.html
[40] https://unix.stackexchange.com/questions/37779/how-do-i-boot-from-a-liveusb-using-qemu-kvm
[41] http://wiki.osdev.org/Enhanced_Host_Controller_Interface