Grenoble University

Marwen AZOUZI - Adrien FAURE

February 1, 2016

1 Qemu

1.1 What is Qemu for?

Qemu is an open-source virtual machine and hypervisor capable of simulating different processor architectures such as the x86, ARM, etc.

1.2 Why cannot you run a linux kernel in a regular linux process?

The kernel's main purpose is to provide an interface between the hardware and the software. Without the hardware (or at least virtualization), the kernel wont be able to run.

1.3 Comment the different options you used to start qemu?

1. `-s`

   Shorthand for -gdb tcp::1234, i.e. open a gdbserver on TCP port 1234. This option will allow us to use the gdb debugger (using tcp via the port 1234).

2. `-S`

   Specifies that we don't want to start the CPU until we provide input (useful for debugging).

3. `-serial stdio`

   Tells qemu we need to bind the current stdin/stdout to the kernel via the serial port.

4. `-hda`

   The disk image we need to boot from.

2 Boot Process

2.1 How is an x86 machine booting up?

The machine first runs the BIOS from the ROM which will enumerate the hardware, initialize it, and finally execute the bootloader. The bootloader then loads the kernel into memory and runs the kernel.

2.2 What is the role of each involved parts?

1. Bios : The Bios, which resides in the ROM, is used to perform hardware initialization and to load the bootloader.
2. Master Boot Record (MBR) : Is a boot sector, situated at the beginning of the disk, containing the bootloader code (and eventually partition information).
3. Bootloader : Loads the kernel into the memory and runs it (could involve multiple stages when the MBR size is relatively small).
4. Kernel : The operating system.

2.3 How is built the disk image that you use to boot with qemu?

The disk image is built thanks to the command dd. It reads streams and writes them directly into the disk (hda.img). The image contains two sectors :

1. The master boot record
2. The kernel

3 Using Eclipse to browse the sources

We used Atom ;)

4 Master Boot Record

4.1 From what sources (.c and .S files) is the MBR built?

The MBR is built from two modules : the loader.c and the boot.S.

4.2 What is the purpose of those different files?

The file `boot.S` initializes the hardware environment and then calls the C function diskboot from the file `loader.c`. The file `loader.c` reads and loads the kernel from the image. Once the kernel is loaded, diskboot jumps to its first instruction.

4.3 What is an ELF? (Hint: man elf, Google is your friend)

ELF (Executable and Linkable Format) is a commonly-used binary file format used to create executables from different objects and resources.

4.4 Why is the objcopy program used? (Hint: look in the Makefile)

Creates a pure executable from different objects stripping all descriptions and extra information.

4.5 What kind of information is available in an ELF file?

The ELF file contains a header which can describe three file types :

1. Program header table, describing zero or more segments
2. Section header table, describing zero or more sections
3. Data referred to by entries in the program header table or section header table

4.6 Give the ELF layout of the MBR files (hint: readelf and objdump)

The elf file contains 32 entries on the symbol table. Also we can see that the entry point of this file is at the address 0x7c00.

4.7 Look at the code in loader.c and understand it.

The loader.C read an ELF file and when its done, it jumps to the elf entry point.

4.8 What are the function waitdisk, readsect, and readseg doing?

Helper to read an elf file. The function readseg is used to load a segment described into the elf file. ReadSect loads the variables into the memory. Wait disk waits for the end of the operation.

4.9 Explain the dialog with the disk controller.

The dialogue with the disk is done using asm directives (such as `__asm__volatile` procedures). Those functions are used to configure or initialize the hardware.

4.10 What can you say about the concepts at the software-hardware frontier?

The communication between the software and hardware in our kernel is performed at a very basic level and needs knowledge of the way they both communicate. A more sophisticated kernel is needed in order to provide a higher abstraction above the hardware and makes it easier to be accessed by software.

5 Master Boot Record Debugging

5.1 List and explain the various gdb commands you use.

1. br nline
2. step (to go into a function for example)
3. next jump : to the next instruction
4. info locals : list all local variables

6 Our mini Kernel

6.1 What is the code in crt0.S doing?

The crt0.S code initializes memory and then runs the function `kmain` from `main.c`.

6.2 What are the function in/out for at this level?

The in/out functions are mainly used to communicate with the devices (i.e. the keyboard, the screen, etc.)

6.3 What are the inline attributes for?

1. The `__inline` is used to tell the compiler that the code can run faster if we substitute the function code into its caller (it avoids assembly goto jumps).
2. The keyword `__attribute__` can be used to specify special attributes for some functions in order to help the compiler optimize calls to them. For example, we can force a function to be inlined, even when optimization is not enabled, by using the `always_inline` attribute.

6.4 Explain why is your fan ramping up when you launch qemu with:

An infinite loop (`while(1)`, `while ((inb(port + 5) & 0x20) == 0)`, and `while ((inb(port + 5) & 1) == 0)` in main.c) is a blocking processorintensive task which uses 100% of one CPU core hence the fan ramping.

6.5 Explain what is the relationship between the qemu option (-serial stdio) and the COM1 concept in the program.

The COM1 is a communication port (COM) which refers to a virtual serial port in our case. It uses the 0x3F8 IO address to send and receive information (characters in our case) through the stdio.

6.6 Explain what is COM1 versus the console?