Grenoble University

Marwen AZOUZI - Adrien FAURE

January 25, 2016

1 Qemu

1.1 What is Qemu for?

Qemu is a opensource virtual machine capable of simulating diffrent physical architecture such as 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 such hardware, the kernel wont be able to load.

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 to use the gdbdebugger. (with tcp via port 1234)

2. `-S`

   do not strat CPU on start-up. Allow to wait user input. USefull for debugging.

3. `-serial stdio`

   Tell 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 run the BIOS which will enumerate the hardware and initialize and execute the bootloader. The bootloader then load the kernel into memory and finally run the kernel.

2.2 What is the role of each involved parts?

1. Bios : Is a firmware used to perform hardware initialization and load a bootloader.
2. Master Boot Record (MBR) : A boot sector that contains the bootloader code.
3. Bootloader : Loads the kernel into the memory and run It.
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 commande dd. It reads streams (object file) and writes it into the image (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 with two modules. the loader.c and the boot.S.

4.2 What is the purpose of those different files?

The file `boot.S` initialize the physic environment and then call the C fonction diskboot from the file loader.c. The file `loader.c` read and load the kernel from the image. Once the kernel is loaded, diskboot jump to its first instruction.

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

ELF from is name Executable and Linkable Format is a file format which

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

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

The ELF file contains an header which can describe three type of files :

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 entry on the symbole table. Also we can see that the entry point of this file is at the adress 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 jump to the elf entry point.

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

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

4.9 Explain the dialog with the disk controller.

Dialoging with the disk implie using asm directive (such as `__asm__volatile` procedures). Those function are used to configure or initialize the hardware.

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

5 Master Boot Record Debugging

5.1 Look at the dbg target in the Makefile.

5.2 Look at the .gdbinit file.

5.3 Use source layout in gdb.

5.4 Use emacs as a front end.

List and explain the various gdb commands you use.

6 Our mini Kernel

6.1 What is the code in crt0.S doing?

The crt0.S code initialize memory and then run the function `kmain` from `main.c`.

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

The in/out are main mainly used to communicate with devices, keyboard, screen etc.

6.3 What are the inline attributes for?

1. The `__inline` is used to tell the compiler than the code can be run faster. In most case the compiler will substitute the whole code into its caller.
2. The keyword `__attribute__` can be used to specify special attributes of 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)` in main.c) is a very processor-intensive task which uses 100

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 recieve information (characters in our case) through the stdio.

6.6 Explain what is COM1 versus the console?

7 Debugging with Eclipse

How did you setup your Eclipse to debug your mini kernel?

8 Kernel Extensions

IT IS MANDATORY TO USE THE DEBUGGER TO DEBUG YOUR CODE.

8.1 Echo on the screen

This extension is to have the input from the UART be echoed on the console screen (the greenish output). Do not forget that you have only 25 lines and you will need to implement scrolling.

8.2 History and line editing

This extension is to have a history of typed lines. A line is added to the history when the return key is pressed. The arrow up and down allow you to scroll up and down in the history. The arrow left and write allow you to move left and right in the current line. The backspace and delete allow you delete characters.

8.3 Echo on COM2

This extension is to have the ability to have a printf-like capability on COM2. The code is in the kprintf.c file.
Hints:

8.4 Look at the target run2 in the Makefile to know how

to setup COM2.

8.5 Add the kprintf.c file to your kernel

8.6 Launch with “make run2” and use a telnet connection

for COM2.

9 Laboratory Log