LSE

my_pikaboot

november 2022

Version 1

Daniel FRÉDÉRIC <daniel.frederic@lse.epita.fr>

Pierre-Emmanuel PATRY <pierre-emmanuel.patry@lse.epita.fr>

Copyright

This document is for internal use only as EPITA.

Copyright © LRE

The copy of this document is subject to conditions :

- ▷ It is forbidden to share this document with anyone.
- ▷ Check that you have the latest revision of this document.

Contents

1 General rules

The following informations are very important and should be read carefully :

Failure to comply with any of the following instructions will result in penalties up to and including the multiplication of the final mark by 0.

These instructions are clear, unambiguous and have a precise objective. Moreover, they are non-negotiable.

Do not fear asking if you cannot understand one of those rules.

General rules 0 : You must read the subject.
General rules 1 : You must follow the rules.
General rules 2 : You must not handout late

General rules 3 : The work must be handed as described in Hand out format.
General rules 4 : The rendered work must not contain binary, temporary, or error files (`*~`, `*.o`, `*. a`, `*.so`, `*#*`, `*core`, `*.log` , `*.exe`, binaries, ...).
General rules 5 : However, if a binary file folder is present, it must be known from an external source (example: a Download section of an open source project). You **must** then explain in the file `README` its usefulness and its source.
General rules 6 : Throughout this document, case (uppercase and lowercase characters) is very important. You must strictly respect the upper and lower case imposed in the subject's messages and filenames.
General rules 7 : Throughout this document, **`login.x`** corresponds to your login.
General rules 8 : Any delay, even one second, results in the non-negotiable score of 0.
General rules 9 : Cheating (sharing code, copying code or text, ...) implies **at best** the non-negotiable grade of 0.
General rules 10 : If you use external code, mention it in your README. Don't forget that you are also graded on your approach.
General rules 11 : Any external code used and not sourced will appear as cheating. We consider it important to draw inspiration from existing sources in order to have an overview of the approach to follow or to improve the quality of the code. Mention it, hiding it will only penalize you.
General rules 12 : If there are any problems with the project, you should contact the people responsible for the subject as soon as possible at the indicated email addresses.

Advice : Do not wait too much before beginning, even if something looks easy, you will never be protected from a stubborn bug.

2 Hand out format

Project managers :	**Daniel FRÉDÉRIC** `<daniel.frederic@lse.epita.fr>` **Pierre-Emmanuel PATRY** `<pierre-emmanuel.patry@lse.epita.fr>`
Tags :	**`[RECRUT][BOOT]`**
Team size :	1
Submission procedure :	git repository
Directory name :	`login.x-my_pikaboot`
Repository url :	`login@git.cri.epita.fr:p/lse/my_pikaboot-login`
Date of submission :	22/12/2022 23h59
Project duration :	4 weeks
Architecture/OS :	Linux
Language(s) :	C/aarch64 assembly
Compiler :	**`aarch64-linux-gnu-gcc/...-as`**
Compile flags :	**`-W -Wall -Werror -std=c99`**

The following files are required :

`Makefile`	The main makefile.
`README`	Contain a project description as well as ways to build and run it, and a list of features.

A **`Makefile`** should be present in the root directory, it shall contain those rules :

`all`	*[Default rule]* launch `build`.
`build`	Compile the project.
`clean`	Remove all temporary files as well as compilation artifacts.
`run`	Launch the `build` rule and then a qemu instance with the build output.

Your code will be tested and inspected by a human, you should therefore respect the given specifications in order to get points. Document your design decisions and your failed attempts. Those things will impact your grade positively.

The expected tree structure for the project is as follows :

```
login.x-my_pikaboot/
login.x-my_pikaboot/README
login.x-my_pikaboot/Makefile
login.x-my_pikaboot/...
```

You will never lose points because of some source file (source code/headers/config) or Makefile as long as their presence can be justified. Empty files can lead to penalties.

The *make -j $(nproc)* **MUST ABSOLUTELY** work out of the box. The *login.x-my_pikaboot/thirdparty/* folder will have the correct symlinks.

3 Introduction

About this document

This document is the system part of the LSE recruitment. It contains a main part split in several steps as well as multiple bonuses. You are expected to do as much as possible but you are not expected to achieve all of them. You **must** finish the main part before beginning any bonus.

All work will be reviewed and taken into account for the final decision. Show us your best!

The code **must** be written in C as well as ARM assembly.

Bare metal development

AARCH64 is an architecture designed by ARM, specifically its 64 bits version. It has many differences with the one of your computer, amd64/ia32e. The best documentation to understand AARCH64 is the ARM website.
(https://developer.arm.com/documentation/102374/latest/).

QEMU is a set of emulators and hypervisors for different architectures and userlands. It support two accelerators and two mode.

QEMU version	TCG	KVM
qemu-system-$ARCH	usable on any $ARCH	usable only in the same $ARCH with kernel support
qemu-$ARCH	usable on any $ARCH, can emulate various userlands	usable only in the same $ARCH. Kernel-agnostic.

For this project we will use qemu-system-aarch64. It is a full system emulator of an aarch64 board, using the TCG accelerator. It translates the AARCH64 instructions into your host's ones.

On the amd64 architecture, the devices are discovered dynamically through various ways, for instance bus enumeration. The kernel must identify the device in order to probe the driver.

The AARCH64 boards do not have generally (there are a few exceptions) ways to discover dynamically devices. So you have to pass two parameters to QEMU: the machine and the cpu. Their combination creates a fully functional board, with cpus features and device tree completely defined. The used machine for this project is the virt one: it is pretty standard and simplifies development; you won't have to take into account complicated device and board-specific subtilities for the core features. The combination of both triggers the writing in the guest memory of a device tree, a tree representation of devices for the linux kernel, in the form of a binary blob.

Bare metal development is not your average ING1 project, there are several constraints and limitations, amongst others, you can't use a dynamic memory allocator for example. If you want a libc, you'll have to port it. You will have access to some given files. You are expected to understand those file as they provide some tools that could make your life easier or provide you informations about the system you're about to set up.

For this subject you will have to set up the linux boot protocol: it's a quite lightweight boot protocol for the linux kernel.

A *build root* is given: it will set up a fully functional linux-based arm64 boot environment from the source fetch. Make sure you are able to build it. It will be the one used to grade your work.

4 Build & Run

The first step will require to familiarize yourself with the given files. It is also **highly recommended** to take a look at qemu log options and monitor mode before proceeding further.
You will be given a whole **buildroot**, featuring Linux and Busybox. It is **strictly forbidden** to boot the linux kernel from the *-kernel* parameter directly. Trying to circumvent the subject will be considered as cheating.

Building the buildroot may take some time. We advise you to grab some documentation and explore the subject meanwhile. If you have troubles building the given files contact the managers.

When everything is built, go to the *ref/* subfolder. Try the command:

```
qemu-system-aarch64 -nographic -machine virt -cpu cortex-a72 -kernel
thirdparty/Image -initrd thirdparty/initramfs.cpio.gz -serial mon:stdio
-m 2G -smp 4
```

Information

If you can't boot the linux, i.e. kernel panic, contact a maintainer with indications on your environment. The official support is NixOS, with an additional support for Ubuntu 22.04.

If you everything goes well, congrats ! You can start developing the bootloader. You must be able to run the exact same linux environment.

5 Hello world

For this step you need to set up a minimal environment in order to use C and achieve an "Hello world" on your serial output.

5.1 Setting up the stack

To achieve so you'll need to set up the stack. The stack address has already been defined in the linker script link.ld as a section symbol. Once set up you can jump to a kmain function defined in C.

You should add your source files to the variable in the Makefile. The offset of the linux on the flash **MUST NOT** be changed unless approved by an organizer.

5.2 UART output

The UART on ARM is an MMIO device, it means accessing some special memory addresses does in fact interact with the device instead of the memory. Write a puts function similar to puts(3) on linux which prints a string to your serial. You will find more informations about this UART in the virt platform documentation.

It is also recommended to use *info qtree* in the qemu monitor.

5.3 Hello world

Reuse your previous work to display an "Hello world" message from C on your serial.

6 Interacting with the virt plat

6.1 UART input

Now that you got output, you need to handle inputs from the UART0 on the virt plat. To do so you need to implement a function such as *getc* as well as a CLI interface. You can for instance make it display "pikaboot> " at the beginning of each line. Handling deletions is not mandatory but can add some flavor to your bootloader.

7 Boot from pflash01 on virt plat

7.1 Booting

You should be able to boot the *thirdparty/Image* file. It is written on the flash, you can look at *Makefile* for the offset. You **must** copy it in the RAM to be compliant to the linux boot protocol. At the end of this part you should get the same linux+busybox setup than in the **Build and Run** part. Modifying the linux offset on the flash is **FORBIDDEN** unless explicitely authorized.

> Information
>
> Use the given files. Add your sources and the given ones to *srcs-y*, try to build them and let the Makefile generate the *pflash.bin* !
> Do not hesitate to create a better source code architecture and adding submakefiles with the source files !

7.2 Commands

From now on you should be able to boot a linux. From your command line you should implement:

boot - Command that boots the kernel
help - Displays help messages for each command

Now you **MUST** create a command for each step. We can be able to test it individually. A help message should be associated. The syntax should be **at least** precised in your *README*

8 Memdump

Implement a memory dump routine. This routine will be invoked with the **md** command on your serial. This command takes three arguments:

- The start address
- The address range
- The size of each load, in bytes

```
pikaboot> md 0x10 60 4
0000000000000010: e3a010ff e5821004 f57ff04f e320f003  | ........O.......
0000000000000020: e5901000 e1110001 0afffffb e12fff11  | ............../.
0000000000000030: 00000000 00000000 00000000 00000000  | ................
0000000000000040: 00000000 00000000 00000000 00000000  | ................
```

Memdump example

The first column represent the start address of each line. Each column then describes the memory content displayed in hexadecimal. Lastly, the right part of the screen should display the ascii representation of your memory, it will be handy when it comes to identify a string.

> Information
>
> AARCH64 has strict alignment. You **MUST** correct the address by decreasing it. If you can't then abort.

9 Memtest

Write a memtest routine to check whether all memory cells are writable/readable with 1/2/4/8 bytes granularity. Try also operations like bit rotation, writing bits one by one, etc... You must indicate a progression status. Lastly, generate a report of your findings.

10 Emergency boot

For this step you should be able to boot through an UART. As the first UART is used by your cli interface you'll need to use the second one. You can take a look at the Kermit project in order to understand how you should proceed to transfer the files. The most common client for this protocol is *ckermit* and we therefore advise you to use it.

11 Bonus

These are ideas for bonuses. You are not expected to implement all of them, choose what you want to do!

11.1 CRC32 check

It may be useful to assert an image integrity before booting. Implement a simple CRC32 code algorithm, and an appropriate command to CRC32 a part of the flash.

11.2 Image signature

You heard about some people trying to boot anything on your hardware, even special handcrafted images. Implement a signature system to counter them.

Information

Going with the BIOS parameter is authorized if you want to sign the whole flash content, but it is not mandatory for this part.

You **MUST** include a private key in your project in order for us to test this bonus. Do **NOT** use this key anywhere else. Generate a new one to be used only for this project.

11.3 Orangepi, Vexpress

Add more platforms to the bootloader. Handle the proper devices to port it completely.

11.4 pflash01 encryption

You heard about some other challengers that want to reverse your flash content to steal your submission! Encrypt it.

11.5 Exception

U-Boot is able to boot at any exception level and drop down privileges, or to dispatch linux at EL2, thus enabling kvm. Implement the same behavior for your project.

11.6 Debugger

Implement a gdb stub in order to debug your bootloader. This behavior should be disabled in release mode, add a compile flag.

11.7 Gotta go fast

Make your code fully functionnal with the different optimization levels. You want to test all the features by adding optimizations successively. Doing so will help you isolate bugs.

11.8 Tftpboot

It's network time! Overcome this fear of most kernel developers and assert your dominance over your doubts by implementing a basic network stack in order to support tftpboot.

11.9 Qemu black wizard

Did you know about semihosting mode ? Find out more about this topic and impress us.

11.10 Free porn

If you have an idea, and it is related to the subject, go for it! Remember to detail it in your *README* and prepare yourself to talk about it during your defense.