



POLITECNICO DI TORINO

DIPARTIMENTO DI ELETTRONICA E TELECOMUNICAZIONI

ACADEMIC YEAR 2023/2024

Operating systems

Laboratory experiences

Authors:

Bonino Andrea - 314709
Fasolis Gabriele - 314903
Mattiauda Mattia - 315096
Mondino Ettore - 317755
Tartaglia Federico - 319857

Professor:

Stefano Di Carlo

October 25, 2023



Contents

I	LAB 2 - Cryptocore Test and Library Creation	2
1	Attributes test	2
2	Library description	4
3	Library test	5

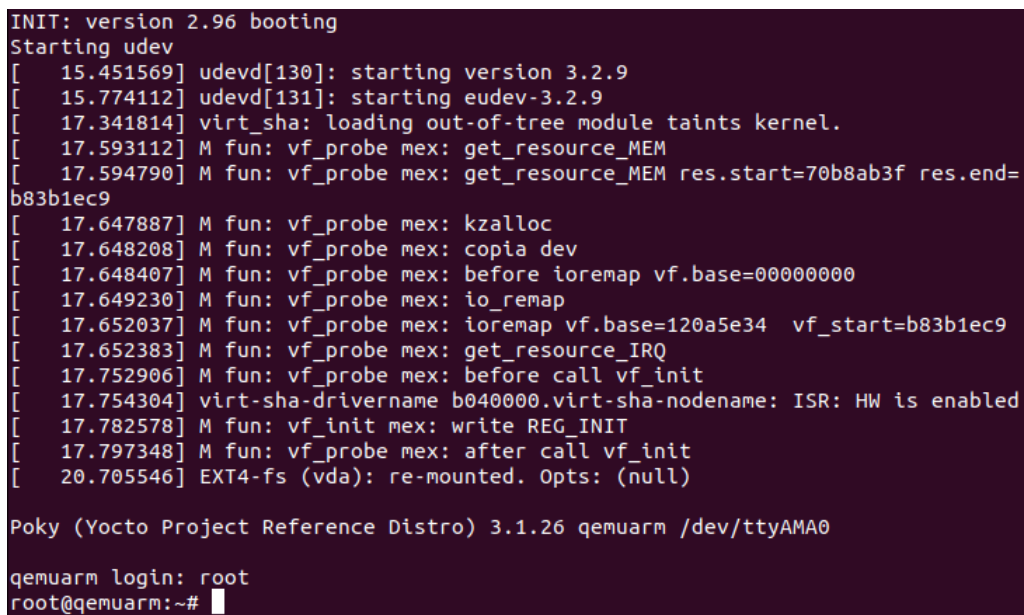
LAB 2 - Cryptocore Test and Library Creation

In this lab experience you will test the cryptocore simulated in LAB 1; then you will be required to code and cross compile a library which aim is to create an higher level interaction with the cryptocore device, allowing you to encrypt an input text string without using the device driver directly (avoiding messy and time consuming code repetition).

1 Attributes test

Up to now we have modified the board adding the device and we have written the driver. Now we can test if everything is working.

1. Start the machine (launch the machine manually or with the script).
Before entering the system you should see the initialization messages as in figure 1.



```
INIT: version 2.96 booting
Starting udev
[ 15.451569] udevd[130]: starting version 3.2.9
[ 15.774112] udevd[131]: starting eudev-3.2.9
[ 17.341814] virt_sha: loading out-of-tree module taints kernel.
[ 17.593112] M fun: vf_probe mex: get_resource_MEM
[ 17.594790] M fun: vf_probe mex: get_resource_MEM res.start=70b8ab3f res.end=
b83b1ec9
[ 17.647887] M fun: vf_probe mex: kzalloc
[ 17.648208] M fun: vf_probe mex: copia dev
[ 17.648407] M fun: vf_probe mex: before ioremap vf.base=00000000
[ 17.649230] M fun: vf_probe mex: io_remap
[ 17.652037] M fun: vf_probe mex: ioremap vf.base=120a5e34 vf_start=b83b1ec9
[ 17.652383] M fun: vf_probe mex: get_resource_IRQ
[ 17.752906] M fun: vf_probe mex: before call vf_init
[ 17.754304] virt-sha-drivername b040000.virt-sha-nodename: ISR: HW is enabled
[ 17.782578] M fun: vf_init mex: write REG_INIT
[ 17.797348] M fun: vf_probe mex: after call vf_init
[ 20.705546] EXT4-fs (vda): re-mounted. Opts: (null)

Poky (Yocto Project Reference Distro) 3.1.26 qemuarm /dev/ttyAMA0

qemuarm login: root
root@qemuarm:~#
```

Figure 1: Initialization messages

2. Enter the system with the user root.
Now you can explore inside the machine looking into different paths:
 - /sys/bus/platform/drivers
Here you should find your device driver with the name assigned in the previous lab (figure 2).
 - sys/devices/platform/
Here you should find your device with the name assigned in the previous lab (figure 3).

```

root@qemuarm:/sys/devices/platform# cd /sys/bus/platform/drivers/
root@qemuarm:/sys/bus/platform/drivers# ls
alarmtimer          of_fixed_factor_clk  virt-sha-drivename
armv7-pmu           pci-host-generic     virtio-mmio
gpio-clk            sbsa-uart
of_fixed_clk        simple-framebuffer
root@qemuarm:/sys/bus/platform/drivers#

```

Figure 2: Device driver inside the simulated ARM machine

```

qemuarm login: root
root@qemuarm:~# cd ../../
root@qemuarm:/# cd sys/devices/platform/
root@qemuarm:/sys/devices/platform# ls
0.flash              a002200.virtio_mmio
3f000000.pcie        a002400.virtio_mmio
90000000.pl011       a002600.virtio_mmio
90100000.pl031       a002800.virtio_mmio
90200000.fw-cfg      a002a00.virtio_mmio
90300000.pl061       a002c00.virtio_mmio
a000000.virtio_mmio  a002e00.virtio_mmio
a000200.virtio_mmio  a003000.virtio_mmio
a000400.virtio_mmio  a003200.virtio_mmio
a000600.virtio_mmio  a003400.virtio_mmio
a000800.virtio_mmio  a003600.virtio_mmio
a000a00.virtio_mmio  a003800.virtio_mmio
a000c00.virtio_mmio  a003a00.virtio_mmio
a000e00.virtio_mmio  a003c00.virtio_mmio
a001000.virtio_mmio  a003e00.virtio_mmio
a001200.virtio_mmio  b040000.virt-sha-nodename
a001400.virtio_mmio  gpio-keys
a001600.virtio_mmio  platform@c000000
a001800.virtio_mmio  power
a001a00.virtio_mmio  psci
a001c00.virtio_mmio  timer
a001e00.virtio_mmio  uevent
a002000.virtio_mmio
root@qemuarm:/sys/devices/platform#

```

Figure 3: Device inside the simulated ARM machine

- `sys/devices/platform/b040000.virt-sha-device`
Here you should find the device attributes of your device (figure 4).

```

root@qemuarm:/sys/bus/platform/drivers# cd /sys/devices/platform/b040000.virt-sh
a-nodename/
root@qemuarm:/sys/devices/platform/b040000.virt-sha-nodename# ls
busy          driver_override  modalias        power
cmd           id               of_node         subsystem
driver        input           output          uevent
root@qemuarm:/sys/devices/platform/b040000.virt-sha-nodename#

```

Figure 4: Device attributes

3. Now you can use the device attributes to test your device; the attributes can be managed exactly like files, so you can write and read them with these commands:

- `echo <VALUE> > <ATTRIBUTE>` to store the string `<VALUE>` inside a writable attribute.
- `cat <ATTRIBUTE>` to show the content of an attribute that can be read.

Anyway, since attributes can be considered as files, you can also open them in a script and write or read text inside them.

First we are going to test the device directly with the basic commands `echo <VALUE> > <ATTRIBUTE>` and `cat <ATTRIBUTE>`, but then we will encapsulate this procedure inside the functions of a library. The sequence of commands and outputs is shown in figure 5 .

Execute these commands in sequence:

(a) `cat id`

This command shows the `id` of the device.

(b) `cat busy`

This command shows the content of the attribute `busy`. This attribute is initialized to 0 and its value is 1 while the device is not available (i.e. is busy). The `busy` attribute is automatically set to 1 once the input is written and it is set to 0 again once the output is read (the cryptocore device is ready to receive another input). You can execute this command many times to check the value of `busy` throughout the different steps.

(c) `echo ciao > input`

This command stores the string `ciao` inside the `input` attribute.

(d) `echo 1 > cmd`

This command stores the value 1 inside the `cmd` attribute, which default value is 0. Once the value 1 is stored the input string conversion process starts.

(e) `cat output`

This command shows the content of the attribute `output`. If the `cmd` attribute has been set to 1 the `output` attribute stores the result of the SHA-256 hashing function on the input string.

```
root@qemuarm:/sys/devices/platform/b040000.virt-sha-nodename# cat id
Chip ID: 0xf001
root@qemuarm:/sys/devices/platform/b040000.virt-sha-nodename# cat busy
0
root@qemuarm:/sys/devices/platform/b040000.virt-sha-nodename# echo ciao > input
root@qemuarm:/sys/devices/platform/b040000.virt-sha-nodename# cat busy
1
root@qemuarm:/sys/devices/platform/b040000.virt-sha-nodename# echo 1 > cmd
root@qemuarm:/sys/devices/platform/b040000.virt-sha-nodename# cat output
b133a0c0e9bee3be20163d2ad31d6248db292aa6dcb1ee087a2aa50e0fc75ae2
root@qemuarm:/sys/devices/platform/b040000.virt-sha-nodename# cat busy
0
root@qemuarm:/sys/devices/platform/b040000.virt-sha-nodename#
```

Figure 5: Attributes test

2 Library description

If everything went well up until now it means that you have a working device interfaced with the user through device attributes.

Accessing the device directly through those attributes is not user-friendly, so specific libraries are usually provided with any device. Now you are required to develop a library which implements specific functions to simplify the interaction with the device.

Tip: among the other functions implement `sha256_algorithm(input, output, number of characters)` that will be used by all the user programs that want to hash a string with SHA-256. This function should take care of all the command flow that you used to manually execute in the terminal (e.g. `echo ciao > input`, `echo 1 > cmd`, `cat output`).

Here is an example of the functions that could be contained in the library:

```
int sha256_inizialize();
int sha256_request(const char* input, int n);
int sha256_get_output(char* output);
int sha256_check_busy();
int sha256_algorithm(char* input,int n, char* output);
```

- `sha256_initialize`
This function can be used for any kind of initialization.
- `sha256_request`
This function inserts the input string inside the `input` attribute and starts the conversion process by inserting the value 1 inside the `cmd` attribute.
- `sha256_get_output`
This function retrieves the output of the encryption process by reading the content of the `output` attribute.
- `sha256_check_busy`
This function retrieves the value stored in the `busy` attribute in order to understand if the device is free or if it is currently hashing another input string.
- `sha256_algorithm`
This function recalls in sequence
 1. `sha256_check_busy` (polling)
 2. `sha256_request`
 3. `sha256_get_output`

3 Library test

In this section we will implement a trivial application, just to understand how to cross compile and use the library. To do so, we will write a C program `Test_sha256.c` which will exploit the library described in section 2. In order to cross compile our program and the library we have to create a new Poky recipe following these steps:

1. At the path:

```
/poky/meta-example/recipes-example/
```

create a new directory with the command:

```
mkdir myhello
```

2. Inside the folder `myhello` create:

- the subfolder `files`
- the file `myhello.bb`

3. Inside the folder `files` insert the program source `Test_sha256.c` and the library `Sha256_library.c`, `Sha256_library.h`

4. Edit the file `myhello.bb` in order to cross compile all the files needed into a single executable.
This is how the `myhello.bb` file should look like after you have edited it:

```
1 DESCRIPTION = "Simple helloworld application"
2 LICENSE = "MIT"
3 LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302
  ↪"
4 SRC_URI = "file://Sha256_library.c file://Sha256_library.h file://Test_sha256.c"
5 S = "${WORKDIR}"
```

```

6 do_compile() {
7     set CFLAGS -g
8     ${CC} ${CFLAGS} Sha256_library.c Sha256_library.h Test_sha256.c ${LDFLAGS} -o
      ↪ TestSha256
9     unset CFLAGS
10 }
11 do_install() {
12     install -d ${D}${bindir}
13     install -m 0755 TestSha256 ${D}${bindir}
14 }

```

(since this is only a user program you do not need to insert a Makefile as you did when you cross compiled the device driver)

5. Add the following line in the the ending part of the file `local.conf`:

```
IMAGE_INSTALL_append = " myhello"
```

(do not insert the line `KERNEL_MODULE_AUTOLOAD` since it is only needed when dealing with a driver)

6. Run the ARM machine simulation and you should find the executable file `TestSha256` at the path `/usr/bin/` (inside the ARM machine).

