# Peek and poke

Stefan Grimminck & Skip Geldens T62

## Part 1

We set out to make an application that could read registers from user-space. We wanted to see what the results were, because we believe that this is not possible.

```
int main(){
    uint32_t* info = (uint32_t*)0x40024000;
    printf("the result is: %u \n", *info);
    return 0;
}

/*
 * Result from LPC: 3813867535
 * Result from Lubuntu: Segmentation fault
 */
```

The program above is what we used to read the data on a register. The register that we need for the RTC on the LPC is: `0x40024000` . This information was in the LPC3250 datasheet, on page 34.

This program gives a Segmentation fault on the Lubuntu image, but gives out a seemingly random number on the LPC. However, this number is fixed, even if we reboot the LPC. The reason that this is happening is that we do have some access to the registers that are outside of our address space. But if we change the address on this line: `uint32_t* info = (uint32_t*)0x40024000;` To `400240000` (so without 0x in front) we do get a segmentation fault. So the LPC3250 clearly has a MMU, because we are restricted when we try to read at this address. In the LPC datasheet there is also a short mention of the MMU, but that does not give a answer on why we can read from an address that isn't ours.

## Part 2

After we'd gathered the basic knowledge of kernel modules by reading LKMPG we created a kernel module to read and write to the /sys filesystem the correct way.

The basic structure of a kernel module is practically always the same, these are the things that should be in every module:

### Includes

```
linux/module.h
linux/kernel.h
```

### An initializing function

This runs when the module is inserted into the running system with insmod.

### A cleanup function

This function runs when the module is removed from the running system with rmmod.

There are some specific functions and macros you use when using the /sys filesystem, we will explain them here

### sysfs_store() (this can be any name, as long as you register it)

This function is called when a user writes something to the file you have defined and made in the /sys filesystem. So when a user or application wants to contact your kernel module, this is going to be done via the /sys filesystem in our case. There are also different ways of talking to the kernel, for example via the /proc filesystem or via /dev.

In this function the kernel module should handle the input of the user, and do something with it. In our case it reads or writes some user specified registers.

### sysfs_show() (this can be any name, as long as you register it)

This function will return something to the /sys file when a user wants to read from it. In our case it just prints a buffer that we filled in the sysfs_store function.

```
static DEVICE_ATTR(hw, S_IWUGO | S_IRUGO, sysfs_show, sysfs_store);
```

This line of code above is a macro for sysfs, this populates a struct with the following parameters:

- name
- mode
- the show function (sysfs_show in our case)
- the store function (sysfs_store in our case)

```
static struct attribute *attrs[] = {
    &dev_attr_hw.attr,
    NULL   /* need to NULL terminate the list of attributes */
};
static struct attribute_group attr_group = {
    .attrs = attrs,
};
```

This snippet shows a struct that contains multiple of the before mentioned device attributes. This is done so that one kernel module can have multiple files in the /sys filesystem.

```
    if (hello_obj == NULL)
    {
        printk (KERN_INFO
        "%s module failed to load: kobject_create_and_add failed\n",
        sysfs_file);
        return -ENOMEM;
    }

    result = sysfs_create_group(hello_obj, &attr_group);
    if (result != 0)
    {
        /* creating files failed, thus we must remove the created directory! */
        printk (KERN_INFO
        "%s module failed to load: sysfs_create_group failed with result %d\n",
        sysfs_file, result);
        kobject_put(hello_obj);
        return -ENOMEM;
    }
```

This code in the init function of the module eventually creates the files in the /sys filesystem. You can see two functions, the `kobject_create_and_add()` and `sysfs_create_group()`.

The `kobject_create_and_add()` function makes a kobject struct, and registers it with the sysfs. The name (first argument) is what gives us a directory in the sysfs where we can create different files in.

The `sysfs_create_group()` function takes the kernel object we just created, and fills it the attr_group struct. This looks to be the same as using `sysfs_create_file()`, but for multiple files at once!

After we have initialised the sysfs the module can do its work!

### Reading registers

First we implemented the functionality for reading the registers. This way we can confirm our code by reading the Up Counter from te RTC. When that's verified we can use this to check if our writing function

works correct.

Reading registers is done by the following code:

```
for (i = 0; i < value; i++) {
    /* print to the kernel log */
  printk(KERN_INFO "Value of Register : %u\n", * (uint32_t * ) regaddr);
  sprintf(temp_buffer, "%u", * (uint32_t * ) regaddr);
  buflen = strlen(sysfs_buffer);

  if ((sysfs_max_data_size - buflen) > (strlen(temp_buffer) + 1)) {
    strcat(sysfs_buffer, " ");
    strcat(sysfs_buffer, temp_buffer);
  }
  regaddr++;
    }
```

The following code is is responsible for retrieving the register value and printing it to the kernel log.

```
printk(KERN_INFO "Value of Register : %u\n",
 *(volatile uint32_t*)regaddr); /* print to the kernel log */
```

regaddr is a casted pointer to a unsigned 32-bit integer. After this operation, the result wil the actual value stored in the address pointed to by regaddr. Volatile is used to make sure the register value isn't based on the compiler's optimalisation or the use of an old copy of the variable.

The register address ( `regaddr` ) is first translated from a physical to a virtual address before reading its values. This translation is done earlier in our code with the following line.

```
regaddr = io_p2v(addr);
```

After the translation is made we loop through the for-loop for the amount of registers we want to read. First, the register that is specified by the user is read, than the upcomming onces if desired. This is done by `regaddr++` which moves the pointer up by one register so that we read the succeeding one in the next cycle of the loop.

We check that the amount of data we read is smaller than or equal to our buffer length, this is done to prevent writing in memory that isn't ours. If the `temp_buffer` is bigger than our `sysfs_buffer` we won't write to the `sysfs_buffer` buffer, otherwise we do.

After writing to the buffer `sysfs_show()` is called to read the buffer and display its contents to the user.

### Writing registers

Our kernel module should also be able to write values to specific registers. We do that with the following code:

```
*(uint32_t*)regaddr = value;
printk(KERN_INFO "Wrote: %u to address: %x", value, addr);
```

Just like the reading part of our code, the register address `regaddr` is first translated to the virtual address. After that we write `value` wich is specified by the user to that register. Next, we write this information to the kernel log

## Testing the kernel module

### Reading registers

To test our kernel module we'll be reading the value of Up Counter of the Real Time Clock (RTC) using its address. Using the LPC3250 manual we found the register for the Up Counter to be `0x40024000`. The value of this register will increase every second, wich we'll use to verify that we are reading the correct registers. When reading this register 3 times we verified this feature as seen below.

We also read two registers to check if this functionality worked. In contrast to the Up Counter the value of the second buffer decreases every second. By executing this command we can also check if the reading from or writing to our buffer works, which it does.

```
# echo "r 40024000 2" > /sys/kernel/es6/hw && cat /sys/kernel/es6/hw
Value of Register : 2178304742
Value of Register : 2136923088
sysfile_read (/sys/kernel/es6/hw) called
# echo "r 40024000 2" > /sys/kernel/es6/hw && cat /sys/kernel/es6/hw
Value of Register : 2178304746
Value of Register : 2136923084
sysfile_read (/sys/kernel/es6/hw) called
# echo "r 40024000 2" > /sys/kernel/es6/hw && cat /sys/kernel/es6/hw
Value of Register : 2178304753
Value of Register : 2136923077
sysfile_read (/sys/kernel/es6/hw) called
```

### Writing to registers

Ofcourse testing wouldn't be done, if we didn't test writing to registers. We used register `0x400a8014` to write our values to.

First we read the value of the register which is 110, than we wrote a value of 0xff3 (1023) to it and confirmed the writing action by reading the value of the register again.

```
# insmod /usr/bin/hoi/sys-reader-1.ko
/sys/kernel/es6/hw created
# echo "r 400a8014 1" > /sys/kernel/es6/hw
Value of Register : 110
# echo "w 400a8014 3ff" > /sys/kernel/es6/hw
Wrote: 1023 to address: 400a8014#
# echo "r 400a8014 1" > /sys/kernel/es6/hw
Value of Register : 1023
```

## References


Walls, C. (2014, July 27). Device registers in C. Retrieved March 6, 2018, from https://www.embedded.com/design/programming-languages-and-tools/4432746/Device-registers-in-C

The kernel development community. Real Time Clock (RTC) Drivers for Linux. Retrieved March 6, 2018, from https://www.infradead.org/~mchehab/kernel_docs/unsorted/rtc.html

The kernel development community. Driver Basics. Retrieved March 6, 2018, from https://www.kernel.org/doc/html/latest/driver-api/basics.html?highlight=kobject*create#c.kobject*create*and*add

Mochel, P. (2005). The sysfs Filesystem. Retrieved March 6, 2018, from https://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf

Mochel, P. (2011, August 16 sysfs - *The* filesystem for exporting kernel objects. Retrieved March 6, 2018, from https://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf

The Linux Kernel Programming Guide, visited on March 6,2018, from https://fhict.instructure.com/courses/5839/pages/lkmpg-6-sysfs?module*item*id=219188