Linux Kernel Programming
Assignment 3

Samarth Joshi (22111051)

April 20, 2023

1 Design and Implementation

At the beginning the pci device is probed as the module is initiated ,within the probe function interrupt is set,with an appropriated handler code.

The user-space functions use a char dev named "cs614-device" to communicate with the pci card. As the encrypt/decrypt function is called via user-space,these functions wrap the important config data in a structure.

```
struct data
{       int type;
        ADDR_PTR s;
        uint64_t length;
        KEY_COMP a;
        KEY_COMP b;
        uint8_t ismmap;
        config_t typ;
        uint8_t set;
};
```

Now this structure is read from the kernel driver via *copy_from_user* function.All the processing and related consequences happen in the device_write function. The information is extracted from the structure and the type of operation is decided (encryption,decryption,config ...etc). The data is loaded into the H/W in the expected manner.

To handle MMIO case ,the keys are loaded and together in a 32-bit integer like this $(a << 8|b)$ because the operation is performed on the whole register and possibly can't be done byte by byte.After this the data is copied at the offset of 0xa8 of the BAR0 area.Appropriate bit is loaded for encryption or decryption.After this the operation is triggered by pasting the location of the string in the driver to MMIO data address register.

To handle the interrupt case,the process instead of wasting cycles to check the completion of encryption,goes to sleep and whenever the MMIO status register bit becomes 0(encryption/decryption complete)the device raises interrupt and the following logic is implemented in our interrupt handler function. The interrupt_handler,code checks whether the interrupt was raised by our device ,as the line can be shared,if yes it wakes up the process after acknowledging the interrupt register,so that interrupt is not repeatedly called.

To handle the DMA case a DMA bus address in generated paired with a CPU physical address,the string to be processed is placed at the physical address and the dma_handle is placed at the appropriate place in the hardware,afterwards the operation is triggered by placing the operation mask in the DMA command register.

Interrupt for DMA needs no different handler and,the code for MMIO interrupt handler works well with it.

To handle the mmap case,the whole 1MB physical region of the PCI device is mapped onto the user-space,this is done by taking the physical address of the h/w device and using remap_pfn_range to allocate pages to corresponding vma in the userspace.Then the user can directly access the 0xa8 region to place the string there so the copying of string can be avoided in kernel space. Although other values,like key etc. can be placed via user-space as the whole region is mapped but the copying of string is the main bottle neck so only it is done.Results of mmaped area can be verified via kernel space.

To handle the big strings ,the user library divides the strings into chunks of size 128*4096 and then writes to the character driver.

To handle multiprocessing spinlocks are put in the kernel device_write region, where all the processing takes place. Also the per process values like keys are updated to their apt values if changed by other process before encrypting or decrypting.

2 Benchmarks

	%system	%iowait	%user	%steal
mmio	34.5	1.3	0.14	0.02
mmio_interrupt	24.66	1.13	0.47	0.006
dma	37.96	1.78	0.096	0.006
dma_interrupt	6.46	1.92	0.865	0.008
mmap	10.76	1.28	2.04	0.01
mmap_interrupt	15.73	2.86	3.12	0.02

3 References

Char device code inspired from Assignment 1.
Blake's Blog for understanding DMA
Oleg Kutkov's blog for understanding general pci driver coding
Basic PCI setup youtube playlist
How to use DMA API

:) Kernel Source tree