Programming Assignment 1
(PCI Device Driver)
CS730: Topics in Operating Systems
2019-20 - Semester II
Computer Science and Engineering Department
Indian Institute of Technology Kanpur
Due Date: 2nd March 2020, 11:55pm

February 5, 2020

This programming assignment is designed to make you acquainted with the working principle of an end-to-end system. In this assignment, you need to write a device driver for a PCI device called `CryptoCard` and provides a user-space library through which an end-user can utilize the functionalities of `CryptoCard`. An end-user can use the device in all possible scenarios, i.e., single process and multi-process.

CryptoCard: Overview and Specifications

Here, we provide an overview of the CryptoCard along with its specifications.

Overview

CryptoCard is a PCI device that encrypts/decrypts the data by using a specified key pair `a & b` and returns the encryption/decryption result to the end-user. CryptoCard uses a single PCI region, i.e., `BAR0`, and have one MB of input/output memory through which a user can communicate with the device. The device can perform encryption/decryption using MMIO or DMA. Both MMIO and DMA can be configured to raise an interrupt after the operation is finished by the device. In the case of non-interrupt mode, user needs to periodically check the status bit of the operation to get the result back. A list of services provided by the CryptoCard is enlisted bellow.

1. Encryption/Decryption with interrupt using MMIO
2. Encryption/Decryption without interrupt using MMIO
3. Encryption/Decryption with interrupt using DMA
4. Encryption/Decryption without interrupt using DMA
5. Set Key pair `a & b`
6. Status of the operation
7. Set length of the data
8. configure address for the data message (MMIO/DMA)

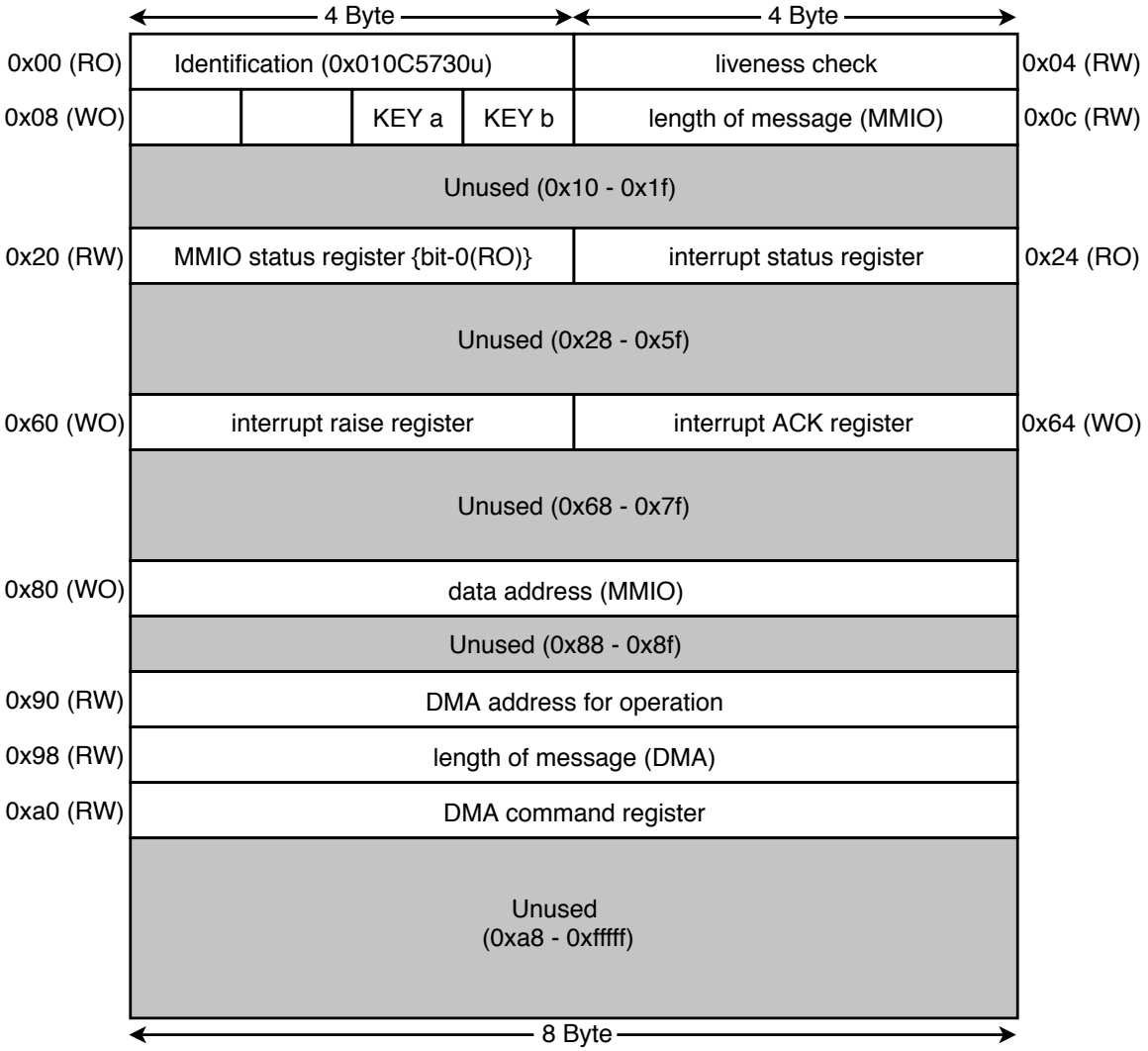

Figure 1: CryptCard memory region.

Apart from the PCI region, device has an internal buffer of 32 KB to store the intermediate results during an operation. This internal buffer is being used for both MMIO and DMA. Once an operation is triggered, device will not accept other request to change the address of data. Hence, user has to handle subsequent requests at driver or library level.
This device is implemented in a custom Qemu, and same has to be used for the assignment. You can download/clone the custom Qemu source code from the link `https://git.cse.iitk.ac.in/skmtr/CS730-Qemu-PA`. To utilize the CryptoCard, you need to build the custom Qemu image and use the script *start_vm.sh* to run an operating system image (can be downloaded from the ftp server `ftp://172.27.36.64/tos.qcow2`). You are free to modify the *start_vm.sh* as per your needs. To access the OS image running through the *start_vm.sh*, you can use the following command.

ssh -p 5555 <username>@localhost

Note: We will notify you if there are any changes in the Qemu source code, you need to rebuild the Qemu image with the updated code. However, any changes in the Qemu source code will not change your driver/library implementation.

Specification

Figure 1 shows the one MB of input/output memory layout that is used to communicate with the CryptoCard. Device has registers of two sizes —*(i)* 4 byte, and *(i)* 8 byte. Device input/output memory below address 0x80 can only accommodate 4 byte while addresses above 0x80 can accommodate 8 byte of data at a time. As shown in Figure 1, operation on device registers or memory regions can be performed on three ways, i.e., (i) read only (RO), (ii) write only (WO), and both read & write (RW). Some of the memory regions of the CryptoCard is unused, that can be utilized by the device driver if needed.

The description of the rest of the memory region (registers) are as follows:

- **identification**: This holds the identification of the device and can only be read. On read, it always returns a static value `0x010C5730u`.
- **liveness check**: To check whether the device is live or not. You can write some value (4 byte) on it. When you read the value, this will return the complement of earlier provided value, which indicates that the `CryptoCard` is live.
- **KEY**: This is a write only register and holds the key for encryption/decryption operation. Both MMIO and DMA utilizes this for the corresponding operation. It holds both components of the key, i.e. `"a"` at byte1 and `"b"` at byte 0.
- **length of message (MMIO)**: It represents the data size on which an encryption/decryption operation has to be performed through MMIO.
- **MMIO status register**: It gives status of the operation for MMIO and can also be used to configure operation type and interrupt. A user can read from and write to this register except the bit 0, which is read only and used to denote the status of operation. Apart from status bit, user can write to other bits. The bit corresponding to operations are shown in table 1:

Bit position	Meaning
0	Read returns 0 if device is free else returns 1
1	Set value to 0 for encryption and to 1 is decryption
7	Set value to 1 to raise interrupt after finishing operation and write 0x001 to interrupt status register on completion

Table 1: Details of MMIO status register Bit Positions

For example:
If the user want to perform decryption using MMIO and raise interrupt, he/she has to write `0x80|0x02` to MMIO status register.

- **interrupt status register (ISR)**: It contains values corresponding to MMIO/DMA. `0x001` indicates MMIO and `0x100` indicates DMA.
- **interrupt raise register**: Can be used to raise interrupt manually for testing.
- **interrupt ACK register**: Writing to this register clears an interrupt. You should write the value present in ISR to clear an interrupt. This is needed to stop generating interrupts.
- **data address**: The stored value contains the address of the data to perform Encryption/Decryption and the result of Encryption/Decryption is put back to the same buffer. This happens only after bit 0 in the MMIO status register (refer table 1) is cleared. Note that writing to this address triggers the Encryption/Decryption process through MMIO.

- **DMA address for operation**: Address to perform Encryption/Decryption and store the result.
- **length of message (DMA)**: It represents the data size on which an encryption/decryption operation has to be performed through DMA.
- **DMA command register**: It controls the DMA operation and uses a bitwise OR operation to combined multiple command. The bit corresponding to operations are shown in table 2:

Bit position	Meaning
0	By setting it to 1 triggers the operation through DMA
1	Set value to 0 for encryption and to 1 is decryption
2	Set value to 1 to raise interrupt after finishing operation and write 0x100 to interrupt status register on completion

Table 2: Details of DMA command register Bit Positions

Task to be done

Till now you were getting familiarized with the functionalities and specification of `CryptoCard`. Form here, your real task starts. You have to build below mentioned two core components,

- Device driver for `CryptoCard`
- User-space library.

Device Driver

Your device driver should implement all the functionalities mentioned under overview. All these functionalities should be configurable from the user-space using `sysfs` variables . Apart from these specified functionalities, you should also provide a functionality to map device memory into user-space and read-/write data directly.
You are free to follow any good coding style for the device driver implementation as no skeleton code is provided by us. We will be using your user-space library for evaluation. We will evaluate expected functionalities using multiple test programs with different scenarios like a single process, multi-process, etc.

Library

Here you have to implement a shared library using the provided skeleton code. You can download/clone the library skeleton code from the link `https://git.cse.iitk.ac.in/skmtr/CS730-PA-library`. Go through the `README` file to understand the directory structure and build process. It contains one test case or example program that uses your library for performing encryption/decryption. In later stage, we will add more test cases for the evaluation. You need to complete the definition of the library functions according to your device driver. You can not modify the signature of a library function.

Testing Scenarios

We will evaluate two scenarios—(i) Single process and (ii) multi-process/multi-threaded.
Test cases for both the scenarios are discussed bellow.

Single Process Test-cases

Single process test-cases are as follows:

1. Encryption/Decryption through MMIO without interrupt.
2. Encryption/Decryption through MMIO with interrupt.
3. Encryption/Decryption through DMA without interrupt.
4. Encryption/Decryption through DMA with interrupt.
5. Encryption/Decryption through MMIO where the device memory region is mapped to the user-space. User will write data directly to the mapped memory and triggers the operation.

Multi-process/multi-threaded test cases

This category of test-cases will also perform encryption/decryption same as mentioned in single process test-cases except the memory-mapped encryption/decryption.

Deliverable:

1. Source code of `library` and `device driver` along with the makefile for the driver code. You should put drivers code inside the directory `drivers` of the library skeleton code along with its makefile. Your makefile should generate drivers kernel module with the name `cryptocard_mod.ko`. During the evaluation, automated script will search only for the driver module with the name as mentioned above.
2. Feel free to discuss about the assignment among your friends, instructor and TAs. However, at the end of the day, we want you to do the implementation by yourself.
3. Write a `README` file containing what you have discussed/with whom/any other sources that you have referred to do the assignment. If you have used any scripts/code from your friends, do mention that as well.
4. Submit the tarball of your assignment named as your roll number. You can use makefile provided in the library code to prepare tarball. The syntax for preparing tar file using makefile is listed in the `README` file.
   We will accept your submission via Canvas only, and any other submission mode is strictly prohibited such as submitting via email or piazza.

All the best. Looking forward to the submissions !!.....PS: Start Early.