Name: Hammad Ansari
Roll No: 22111025

Linux Kernel Programming (CS614)
Assignment 3 Report

Design

- **Driver Module:**
  1. Character device is created with name demo_device, which is used to communicate with the kernel module from user space.
  2. PCI device cryptocard is registered.
  3. In the probe function of the device driver, VA handle is created to read/write into the memory mapped region of PCI device. This VA handle is global variable.
  4. In the probe function, IRQ handler is registered for the IRQ number of PCI.
- **User Space Library:**
  1. Handle is created for the character device to communicate with the driver module.
  2. All the functions namely **encrypt()**, **decrypt()**, **set_key()**, **set_config()** uses **write()** system call on device handle to write the data in the PCI device and **read()** system call to overwrite the encrypted/decrypted value on original value.

Implementation

- The following structure is passed to the driver module using write() system call.

```
struct data{
        unsigned char operation;
        uint8_t isMapped;
        char *str;
}
```

- In the above structure, the value of operation is 0,1,2 or 3.
  0 – Encryption
  1 – Decryption
  2 – Set key values a & b
  3 – Set configurations (DMA/Interrupt)
  The value isMapped is 0 or 1 based on whether the device memory is mapped to user space or not.
  str contains the string that needs to be encrypted/decrypted in case of encryption/decryption operation and it contains value of key a & b on first two indices in case of set key operation and contains configuration of DMA and Interrupt on first two indices which shows that DMA and Interrupt is enabled/disabled.
- In driver module, the character device's write() operation is implemented as follows:

1. Based on the operation value in the structure that is passed from the user space, the corresponding operation is performed.
2. If the operation is set config then the global variables present in driver module is changed correspondingly and during encryption/decryption the global variables are used to decide the configuration.

- Types of Operations on device:

  1. MMIO without Interrupt:
     - The user data is written to the unused area of device memory byte by byte.
     - MMIO status register and MMIO length register is wriiten with the desired value and at last MMIO Data address register is written to trigger encryption/decryption.
  2. MMIO with Interrupt:
     - Everything is same as MMIO without interrupt except that process sleeps until the interrupt is generated using wait_event_interruptible() function.
  3. DMA without Interrupt:
     - dma_alloc_coherent() function is called while probing the PCI device. It allocates the kernel memory for dma operation and returns the virtual address and dma address. DMA address is used by device to access memory and virtual address is used by driver module to access memory.
     - The user data is copied to the allocated memory using using above virtual address and DMA address is written to the DMA address register of the device and DMA message length register is written with the desired value.
     - DMA command register is written with the desired value and by setting the 0^{th} bit of the DMA command register triggers the operation.
  4. DMA with Interrupt:
     - Everything is same as DMA without interrupt except that process sleeps until the interrupt is generated using wait_event_interruptible() function.
  5. Mapping the device memory to user space:
     - In mmap file operation of character device, remap_pfn_range() function is used to map the device memory to the user space.
     - When the user space calls the mmap system call, it maps the device memory to the user space.

- In character device read() operation,

  1. MMIO/DMA without Interrupt:
     - Process will be stuck in while loop until the 0th bit of MMIO status register (in case of MMIO) or 0th bit of DMA command register (in case of DMA) will be equal to zero.
     - Then, the encrypted/decrypted value is read from device memory or physical memory (in case of DMA) and written to the user buffer.
  2. MMIO/DMA with Interrupt:
     - When the device is done with encryption/decryption, it will generate interrupt which will eventually call IRQ handler corresponding to the device IRQ number and IRQ handler will wake up the process sleeping in the queue.

  - After that, read() call is performed from the user space and similar thing will be as in the case of MMIO/DMA without interrupt.

- Handling of shared Interrupts:
  1. While registering the IRQ handler, last argument passed is the virtual address corresponding to the starting address of the memory-mapped device memory which is global variable in driver module code. It will be last argument in the IRQ handler definition.
  2. In the definition of IRQ handler, the equality of the same last argument is checked with the global variable to handle the shared interrupts.

- Handling of big size data:
  At user library level, the data is divided into chunks of 4KB and then it is provided to device to perform encryption/decryption.

Test Cases:

- In the test case, a string with length of 4MB is passed to for encryption and decryption is done on encrypted string. It is done using all the four operations i.e. MMIO with/without Interrupt and DMA with/without Interrupt.

Benchmark Results:

Operation	%user	%system	%iowait	%steal	Total CPU Utilization (%)
MMIO without Interrupt	0.05	50.80	3.01	0.04	53.9
MMIO with Interrupt	0.51	51.71	2.38	0.04	54.64
DMA without Interrupt	0.03	50.13	0.80	0.03	50.99
DMA with Interrupt	0.62	4.07	2.41	0.08	7.18
mmap without Interrupt	36.92	16.00	5.77	0.04	58.73
mmap with Interrupt	28.42	21.84	7.03	0.07	57.36