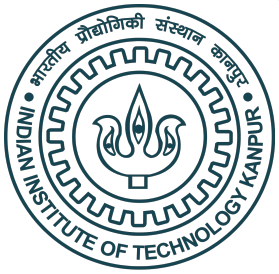


CS614 Linux Kernel Programming

Assignment 3

Debadatta Mishra

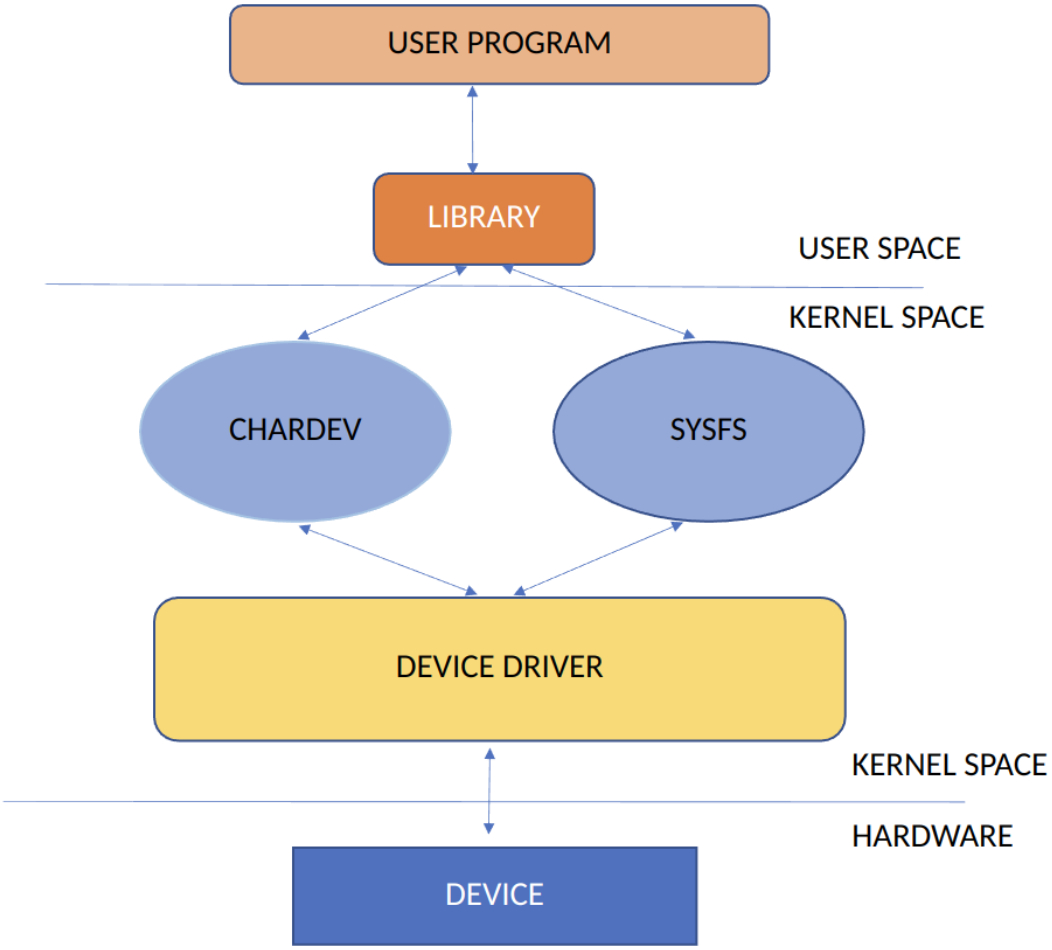
April 20, 2023

Jaya Gupta

Roll No: 200471

Contents

1 Design



The basic flow of the codebase is as follows:

- User program calls library function to access different functionalities of the device.
- Library function uses sysfs and chardev interfaces to communicate with the kernel device driver module.
  - `create_handle`, `close_handle`, `set_key`, `set_config` → These functionalities uses sysfs file to write or read the required info.
  - `encrypt`, `decrypt` → These functionalities uses chardev to communicate information to and from the device driver.
- Device Driver module initialises the driver and register the `probe` method. Different sysfs and chardev handler functions call the `readl, writel..` to communicate with the device.

2 Implementation Details

2.1 Device Driver Implementation

2.1.1 Few Properties of our Device and Implementation

- CryptoCard device handles only one request at a time. Hence it is not a concurrent device.
- Because of the above reason, if a process generates encrypt or decrypt request, it will return to the user library only after the request is completed by the device both in case of interrupt and no-interrupt.
- In case of multiple process, if a request(encrypt or decrypt) is active on the device then it waits for it to get completed and then submits its request to the device.

2.1.2 Implementation Specifics

- A linked list of all the process is maintained. Each entry is of the following type:

```
struct proc_keys_list_entry{
   struct list_head list;
   pid_t pid; // pid of the process
   int handle; // handle
   unsigned char keya;
   unsigned char keyb;
   char* result; // result of encryption and decryption
   int result_len;
   struct process_config config; // proc configs
   bool is_config_set;
   bool is_key_set;
   atomic_t is_intr_handled;
}
```

- `handle_status` → When this function is called, it creates a new entry for process and gives it a new handle.
- `handle_set` → This function deletes entry for the process from the list.
- `key_set, config_set` → These function calls set the keys and configs for the process. It finds the matching process in the list and set info in its struct.
- `chardev_write` → This function is called by decrypt and encrypt using the chardev. It is used to register encrypt and decrypt request from a process. Since only one request can be active on a device at a time, `encrypt_decrypt_mutex` is held. The process which gets the lock submits its request to the device and waits for it to get completed.
  - In case of non-interrupt mode, it keeps polling to the device.
  - In case of interrupt mode, it checks its variable `is_intr_handled` periodically which is set by the interrupt handler.

Once the request gets completed it writes the encrypted/decrypted data into a kernel buffer in its struct. Further read call on the sysfs file writes the data back to user buffer.

- `perform_mmio_encrypt_decrypt` and `perform_dma_encrypt_decrypt` → These functions are used to handle encryption and decryption in the two cases(MMIO or DMA).
- `map_card` → This function used `io_remap_pfn_range` function to map device memory to user space.

2.1.3 Basic Initialisaton Functions

- `init_module` → We initialise our sysfs directory and chardevs here. Also the PCI device is registered along with its `probe` and `remove` methods.
- `crypto_probe` → This is the probe handler for our device driver which gets called as soon as PCI finds our device. This function requests the $mmio$ region and maps it to kernel virtual address space. It also allocates a DMA buffer and writes the DMA address in the device DMA Address region.
- `crypto_request_irq` → This function is used to request irq for our device.
- `crypto_intr` → It is the interrupt handler for our CryptoCard device. Since we have used a shared interrupt line, it checks whether the interrupt generator is our device or not. If yes, it acknowledges the interrupt by writing ISR value to Interrupt Acknowledgement Register.

2.2 User Library Implementation

In order to extend the kernel functionality in the user space, user-library interface is provided. It makes use of sysfs and chardevs to communicate with the kernel module.

- `create_handle` → This function sends the handle to the process, which the process can further use to communicate with the device driver.
- `close_handle` → This function deletes the entry of the process from the linked list in the driver module corresponding to the handle sent.
- `encrypt` and `decrypt` → In case of mapped region,it just write its request for encryption/decryption to the chardev and returns. In case of unmapped region, it writes its request and then reads back the encrypted/decrypted data from the device.
  In case of larger message it is divided into chunks.

3 Test strategies

- In order to ensure that the system was free of deadlocks, several threads were created and it was verified that the encryption/decryption processes were accurate.
- To verify the functionality of the system when dealing with larger data buffers, various file sizes were employed during the encryption/decryption processes and it was confirmed that the results were correct.
- Furthermore, different system configurations were also tested, including those that utilized MMIO with and without interrupts, as well as DMA with and without interrupts, and MMAP.

4 Benchmark Results

4.1 CPU Usage Stats

4.1.1 MMIO with no-Interrupt

FILE SIZE	CPU	%user	%nice	%system	%iowait	%steal	%idle
4 MB	all	0.50	0.00	51.26	0.00	0.00	48.24
8MB	all	0.51	0.00	51.27	0.00	0.00	48.22
10MB	all	1.00	0.00	51.50	0.50	0.00	47.00

4.1.2 MMIO with Interrupt

FILE SIZE	CPU	%user	%nice	%system	%iowait	%steal	%idle
4 MB	all	0.50	0.00	52.00	0.50	0.00	47.00
8MB	all	1.01	0.00	51.76	0.00	0.00	47.24
10MB	all	0.50	0.00	51.76	0.00	0.00	47.74

4.1.3 DMA with no-Interrupt

FILE_SIZE	CPU	%user	%nice	%system	%iowait	%steal	%idle
4 MB	all	0.49	0.00	51.23	0.00	0.00	48.28
8MB	all	1.00	0.00	51.50	0.00	0.00	47.50
10MB	all	0.50	0.00	52.24	0.00	0.00	47.74

4.1.4 DMA with Interrupt

FILE_SIZE	CPU	%user	%nice	%system	%iowait	%steal	%idle
4 MB	all	0.50	0.00	50.50	0.00	0.00	49.00
8MB	all	1.00	0.00	51.74	0.00	0.00	47.26
10MB	all	0.50	0.00	51.5	0.00	0.00	47.74

4.1.5 MMAP with no-interrupt

FILE_SIZE	CPU	%user	%nice	%system	%iowait	%steal	%idle
4 MB	all	27.86	0.00	14.43	0.00	0.00	57.71
8MB	all	34.17	0.00	17.59	0.00	0.00	48.24
10MB	all	32.32	0.00	20.20	0.00	0.00	47.47

4.1.6 MMAP with Interrupt

FILE_SIZE	CPU	%user	%nice	%system	%iowait	%steal	%idle
4 MB	all	28.14	0.00	16.08	0.00	0.00	55.78
8MB	all	32.66	0.00	19.60	0.50	0.00	47.24
10MB	all	4.50	0.00	19.00	0.00	0.00	76.50

4.2 Total Time Taken in different IO modes for different file size

FILE_SIZE	MMIO	MMIO-INTR	DMA	DMA_INTR	MMAP	MMAP_INTR
1MB	0.962	1.025	51.214	51.211	0.234	0.251
4MB	3.734	3.873	204.826	204.829	0.8402	0.880
8MB	7.380	7.689	409.709	409.628	1.673	1.698
10MB	9.336	9.348	512.074	512.076	2.108	2.127

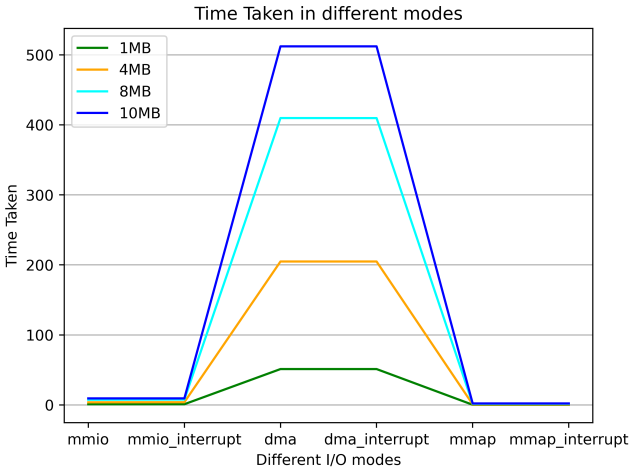


Figure 1: Fig1: Linear Time Scale

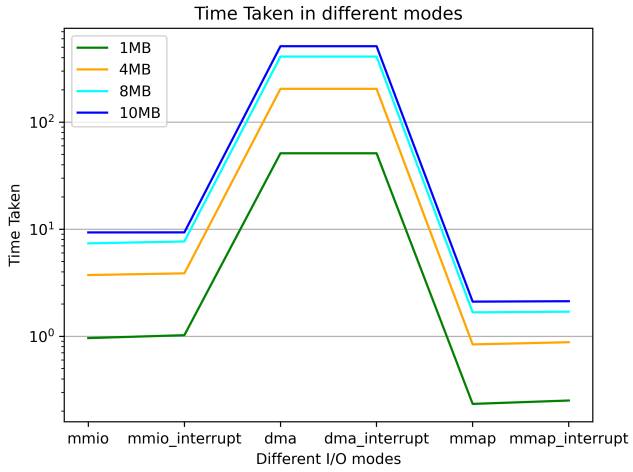


Figure 2: Fig2: Log Time Scale