CS730 Assignment-1 Report

Ketan Chaturvedi (190428)

March 3, 2022

Design

CryptoCard allows user to encrypt/decrypt a buffer. To facilitate this, module creates a device file (/dev/cryptocard) so that users can access it with regular file operations. It also provides following sysfs variables so that users with sufficient privileges can directly tweak the configuration of the device.

1. **irq**: RW, 0 means IRQ not set and 1 means IRQ set
2. **key**: RW, gives the 2 byte key in the format $(a << 8)|b$
3. **memop_type**: RW, 0 means MMIO and 1 means DMA
4. **mmap**: RO, Tells if the device is mmaped or not (0 - not mapped, 1 - mapped)
5. **status**: RO, Tells if the device is busy or not (1 - Busy, 0 - Free)

The char device file allows user to perform following file operations:

1. **open**: Gets called when any process opens the file. It creates a structure which is useful in distinguishing encryption/decryption requests.
2. **release**: Gets called when the file descriptor is closed. Responsible for freeing the data structures allocated at the time of open.
3. **read**: This functions allows a process to read data from device memory. If a process has mmaped the file to its virtual space then only it can read from it. Simply opening the file and reading will result in error.
   WHY? - Let's say 2 processes A and B opened the file and process A is writing to it. Now process B can read process A's data by using read syscall. Clearly its a security flaw and should not be allowed.
4. **write**: This function allows a process to write data to device memory. If a process has mmaped the file to its virtual space then only it can write to it.
5. **mmap**: Gets called when any process maps the device file to its virtual space. If a process maps the file, then no other process can open it. The driver also registers a function which gets called at the time of munmap.
6. **ioctl**: Following ioctl commands are provided to the user to interact with the device:
   (a) **CC_IOCENC**: Requests the device to encrypt the accompanied data
   (b) **CC_IOCDEC**: Requests the device to decrypt the accompanied data
   (c) **CC_IOCSETDMA**: Requests the device to use DMA mode
   (d) **CC_IOCSETMMIO**: Requests the device to use MMIO mode
   (e) **CC_SETIRQ**: Requests the device to use interrupts

(f) **CC_UNSETIRQ**: Requests the device to not use interrupts

(g) **CC_SETKEYS**: Requests the device to use accompanied key

(h) **CC_MMAPLEN**: If a process has mapped the file to its address space, then it can use this command to get the size of mmaped region.

Implementation Details

Managing multiple open files

If multiple processes have opened the file and have set different configurations, then it is necessary to keep track of configuration for each file descriptor. This is done inside the driver. Driver maintains a hashtable indexed by the file pointer. Each node in the hashlist also contains pid of the process and the file pointer value. This hashtable uniquely identifies each open file descriptor:

1. If same process open the file twice and the 2 file pointers indexes to:
   (a) **same entry in hashtable** - File pointer value and pid can uniquely identify the node.
   (b) **different entry in hashtable** - The nodes are in different hashlists.
2. If 2 processes open the file and their file pointer indexes in :
   (a) **same entry in hashtable** - pid can uniquely identify the node.
   (b) **different entry in hashtable** - This creates no problem as they are in different lists.

Handling multiple encryption/decryption requests

When a user request uses the device and initiates encryption/decryption, it takes a mutex (*device_mutex*), which prevents further requests from modifying the ongoing state of the current request. Further user requests are kept in a wait queue with exclusive flag on to maintain the order in which requests will get processed. At the end (when request is completed), lock is released and other processes are allowed to continue.
When a requests gets complete it calls *wake_up_interruptible* to wake up the next entry from wait queue.

Atomic counters

Two atomic counters are used in implementation:

1. **mmap_lock**: This lock increases in mmap file operation and decreases in munmap only. This is used to prevent a process from opening the device file when it is already mmapped.
2. **device_opened**: To keep track of number of times the char device is opened.

Managing Interrupts

When a process set interrupt enabled, then after it has initiated the request, its state is changed to TASK_INTERRUPTIBLE. The interrupt handler is responsible for waking up this process when device completes the request.

Restrictions

The driver has following restrictions:

1. If a process mmap the device file, then no other process can open it.

Benchmark Results

Benchmark	%usr	%system	%wait	%CPU
mmap_interrupt	72.14	5.00	0.29	77.14
mmap	75.43	24.46	0.00	99.89
dma	0.00	99.85	0.12	99.85
dma_interrupt	0.00	0.29	0.08	0.29
mmio	0.00	99.77	0.18	99.77
mmio_interrupt	0.00	94.91	0.00	94.91