CS730 Assignment 1: PCI Device Driver

180301 - hrugved wath

March 1, 2022

1 Design

User programs can access the services of cryptocard device using the `cryter.c` library which interacts with the device driver `cryptocard_mod.c` using a combination of sysfs variables and character device.

- Driver - `cryptocard_mod.c`
  - Configuration can be set using following sysfs variables - INTERRUPT(0/1), DMA(0/1), KEY_A(0-255), KEY_B(0-255), DECRYPT(0/1), IS_MAPPED(0/1).
  - DMA=1 and IS_MAPPED=1 is not supported.
  - encrpytion and decryption requires writing input to character device and reading from the same for getting result back.
  - Limited concurrency support with effectively exposing a lock using a sysfs variable TID. Each task has to lock the driver, set desired configuration, do the processing and unlock the driver.
- Library - `cryter.c`
  - provides a minimal interface for user programs
  - Support encryption/decryption of arbitary sizes by operating on data in chunks.
  - Provides a multi-device handle support. The device configuration is per-handle, i.e changing any configuration including keys of one handle will not affect other device handles.

2 Implementation

- Driver - `cryptocard_mod.c`
  - All sysfs variables except TID, are in an attribure group controlled by `sysfs_store()` and `sysfs_show()`.
  - `sysfs_store_tid()` and `sysfs_show_tid()` controls sysfs varialble `TID`. `TID` at any time is either the tid of the task currently operating the device or -1 if free. `sysfs_store()` is protected by a mutex `dev_lock`. Each task tries to acquire `dev_lock`, then writes its tid to `TID`. Now the task is free to operates on the device. Once done, this task has to store -1 to TID which is accomapnied by releasing `dev_lock`.
  - `cdev_write()` is characted device write function. If memory is not mapped, it first copies data to `mmio_buf/dma_buf` and then calls `mmio()/dma()`. This functions returns only when result is available.
  - `cdev_read()` is called for reading result. The result is guranteed to be available when called.
  - `cdev_mmap()` is called for mmap the device memory.
  - `write_to_device()` and `read_to_device()` are used for copying kernel buffer to and from device memory respectively.

  - mmio() and dma() either polls the status register in case of non-interrupt mode, else in interrupt mode, it waits on a wait queue (wq_is_data_ready) in intereruptable mode.
  - irq_handler() is called in interrupt mode. It signals the wq_is_data_ready wait queue.

- Library - cryter.c
  - It maintains a linked list of opened cdevs in head of struct dev_handle. Calling create_handle() adds a new entry to this list. dev_handle apart from cdev, contains device configuration.
  - The linked list has utility functions for adding, deleting and most importantly fetching dev_handle of a cdev. All operations are protected by mutex mutex_list for multi-threaded support.
  - The set_* functions is used by user to set config related to cdev, i.e updating the corresponding dev_handle. The _set_device_config() is called for actually setting the device configuration. This functions internally uses _set_device_* functions to set the individual sysfs variables.
  - _lock_device() tries to get hold of driver by writing its tid to sysfs TID. This is blocking call as it requires locking the mutex inside driver. _unlock_device() writes -1 to TID.
  - encrpyt first tries to get hold of driver using _lock_device(), it then setup the device config of provided cdev using a _set_device_config(). Calls _device_operate() repeadelty with chunks of data. Finally calls _unlock_device() to release the device. decrpyt operates in similar fashion.
  - MMIO_CHUNK_SIZE and DMA_CHUNK_SIZE determins the chunk size for mmio and dma operation respectively. It is the max size of data device can operate on in given mode.
  - map_card() and unmap_card() shifts the pointer with DEVICE_MEMORY_OFFSET which basically points to start of unused memory region where device expects the data.

3 Testing

- Passed all the tests given in eval-test suite.
- systematic testing targeted for checking a particular feature. Like creating multiple device handles in multi-process and multi-threaded environments to check for correct list implementation. Using data of various sizes for both checking both operating in chunk feature and mmap calls.
- Multi-threaded testing with around 10 threads using the device concurrently.
- Ad-hoc testing, trying to find edge cases.
- perturbation to find possible scenarios of deadlocking.

4 Benchmarks

`pidstat 1 -e ./program` : collect stats every 1 second and averages them.

program	%usr	%system	%guest	%wait	%CPU
mmio	0.87	76.80	0.00	22.36	77.67
mmio_interrupt	0.25	7.57	0.00	14.85	7.82
dma	0.02	98.47	0.00	1.50	98.49
dma_interrupt	0.01	0.34	0.00	0.08	0.35
mmap	1.46	78.87	0.00	19.56	80.33
mmap_interrupt	0.05	1.12	0.00	2.09	1.18