Key-value pseudo device

OVERVIEW

In order to keep the kernel small but also provide the flexibility and generality for machines with different tasks, most modern operating systems support "loadable kernel modules". In this way, the system can boot with a simpler, smaller kernel and then load these modules into kernel space when necessary.

In this project, you will be implementing a loadable kernel module under Ubuntu Linux. This kernel module will create a pseudo device that maintains a key-value store in your kernel and allow different processes to share data through accessing this device. By doing so, you will learn how to write a kernel module, how to write programs working in the kernel space, how the kernel module interacts with user-space applications and the role of device drivers, library and applications in a system.

You may use the VCL service maintained by NCSU through https://vcl.ncsu.edu/ . You may reserve one virtual machine and connect to this machine remotely by selecting reservations. We will use the "Ubuntu 14.04 Base" to test your kernel module. However, this virtual machine will reset once your reservation timeout. You should use https://github.ncsu.edu to control/maintain/backup your work.

You may work either alone or in a group of 2. Groups do the same project as individuals. All members receive the same grade. Note that working in groups may or may not make the project easier, depending on how the group interactions work out. If collaboration issues arise, contact your instructor as soon as possible: flexibility in dealing with such issues decreases as the deadline approaches.

In this project, you will be given the prototype of the kernel module with a keyvalue.c file that only contains empty functions. We also provide a user-space library that allows an application to interact with this kernel module through ioctl interface and a sample benchmark application that you may extend to test if your kernel module functions correctly.

How to start

To begin, you may use git to fetch the code from https://github.ncsu.edu/htseng3/CSC501_KV. We have a Makefile under each subdirectory of the provided code.

In the kernel_module directory, you may try to "make" with root permission (e.g. "sudo make"). If it success, you will see a "keyvalue.ko" under this directory. You can then type "make all" using root permission to install the kernel module and the header into system directories. After you are done with this, you can now "insmod keyvalue.ko" using root. If this success, you can navigate to /dev directory and you should see a "keyvalue" device under this directory! However, at this point, this device doesn't really do anything. It's now your responsibility to endow this device with some features! You may need to unload the device by using "rmmod keyvalue" before you want to apply any change to the kernel module.

In the library directory, we provide an API for the application to interact with the device. You don't need to modify anything in the directory. However, you should read the library code to figure out how the library interact with the device to make things happen. To allow your application using this library, you need to "make all install" to have the library build and install in your system.

Finally, you will see a benchmark directory. In this directory, we provide two files for you to extend and test your kernel module. The benchmark.c file uses keyvalue_set function to update values stored by our kernel module while the validate.c file uses keyvalue_get function to retrieve data and compare if the values is the same as the ones from a log file. You may "make all" in this directory (you don't need root to do this) to see how these files works. You may need to include /usr/local/lib in your library path to avoid runtime errors.

Your task

In this project, you are responsible for completing the functions in the given keyvalue kernel module to support the required feature.

These functions are:

1. `static long keyvalue_set(struct keyvalue_set __user *ukv)`

This function receives a command from the argument with type `struct keyvalue_set`. This `keyvalue_set` struct contains an unsigned 64-bit key, unsigned 64-bit size and a void pointer to the input data. The input data can be any thing no longer than 4KB. The function will then store the given data in the keyvalue store along with the given key. If the kernel module runs out of space, this function should return -1. Otherwise, return and increment the transaction_id

2. `static long keyvalue_get(struct keyvalue_get __user *ukv)`

This function receives a command from the argument with type `struct keyvalue_get`. This `keyvalue_get` struct contains an unsigned 64-bit key, pointer to the size and a void pointer to the data array. The function should find the data that matches the given key. This function will put data in the given data pointer and tell the library about the number of bytes in the memory location that size pointing to. If the kernel module cannot find the data with the given key, this function should return -1. Otherwise, return and increment the transaction_id

3. `static long keyvalue_delete(struct keyvalue_delete __user *ukv)`

This function receives a command from the argument with type `struct keyvalue_delete`. This `keyvalue_delete` struct contains an unsigned 64-bit key. This function will delete the data along with the given key. The key and data should no longer in the keyvalue store module. If the kernel module cannot find the data with the given key, this function should return -1. Otherwise, return and increment the transaction_id.

When you implement these functions, you need to make sure your kernel module can support multiple different programs/processes/threads to access concurrently and execute commands atomically.

You may need to reference the Linux kernel programming guide and Linux Device Drivers, 2nd Edition since user-space libraries will not be available for kernel code.

Turnins

In this project, you only need to turn in the keyvalue.c in the kernel_module directory. Exactly 1 member of each group should submit the source code. All group members' names and Unity IDs should be easily found in a comment at the beginning of the code.