Operating Systems Course project

Year: 2016/2017

Intro

Linux kernel modules – also called Loadable Kernel Modules (LKM) – are among the most useful tools that Linux provides to allow advanced users to write kernel programs without actually messing up the kernel. They are pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system.

Unlike development on core subsystems of the kernel—which is much of the material discussed thus far—module development is more like writing a new application, at least in the sense that modules have entry points and exit points and live in their own files.

LKMs are actually the way you write device drivers today. One advantage is that you don't have to rebuild your kernel as often. This saves you time and spares you the possibility of introducing an error in rebuilding and reinstalling the base kernel. Once you have a working base kernel, it is good to leave it untouched as long as possible.

Another advantage is that LKMs help you diagnose system problems. A bug in a device driver which is bound into the kernel can stop your system from booting at all. And it can be really hard to tell which part of the base kernel caused the trouble. If the same device driver is an LKM, though, the base kernel is up and running before the device driver even gets loaded. If your system dies after the base kernel is up and running, it's an easy matter to track the problem down to the trouble-making device driver and just not load that device driver until you fix the problem.

LKMs can save you memory, because you have to have them loaded only when you're actually using them. All parts of the base kernel stay loaded all the time. And in real storage, not just virtual storage.

LKMs are much faster to maintain and debug. What would require a full reboot to do with a filesystem driver built into the kernel, you can do with a few quick commands with LKMs. You can try out different parameters or even change the code repeatedly in rapid succession, without waiting for a boot.

LKMs are not slower, by the way, than base kernel modules. Calling either one is simply a branch to the memory location where it resides.

Sometimes you have to build something into the base kernel instead of making it an LKM. Anything that is necessary to get the system up far enough to load LKMs must obviously be built into the base kernel. For example, the driver for the disk drive that contains the root filesystem must be built into the base kernel.

For more information on how to build Linux modules, you can read here, or watch the video to come later on scalable-learning:

1. `https://tnichols.org/2016/07/16/Basic-Loadable-Linux-Kernel-Module-Example/`

2. http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/
3. https://wiki.archlinux.org/index.php/kernel_modules
4. https://linux.die.net/lkmpg/index.html

1 Project description

We want you to build an LKM that implements a key-value store that can be used by multiple programs to share data. We will build this in steps. Please try to stick to the soft deadlines. We would like to really make sure you are on track!

1.1 Module1: Your first step (Soft Deadline: 29th of November)

We want you to build a kernel module that provides a simple key-value store like functionality. Key-value store technologies now power most of our web. If you are unaware of what Key-Value stores are, you can check the Wikipedia entry as it provides an excellent introduction. In short "A key-value store, or key-value database, is a data storage paradigm designed for storing, retrieving, and managing associative arrays, a data structure more commonly known today as a dictionary or hash. Dictionaries contain a collection of objects, or records, which in turn have many different fields within them, each containing data. These records are stored and retrieved using a key that uniquely identifies the record, and is used to quickly find the data within the database."

In your first step, we want you to implement a Key-Value store like functionality without thinking about synchronization for now if multiple threads try to read and write. You will need to demonstrate that your Key-Value store works by building more than two programs/processes/threads that read and write to your key-value store.

Implementing such a system as a Kernel module is not easy to get full functionality. We are thus happy to hear from you what assumptions you are going to make to fill the holes in our description.

1.2 Module2: Add Synchronization (Soft Deadline: 5th of December)

Now that you have learnt about different synchronization primitives in Operating Systems and in the Linux kernel. Adjust your previous implementation to allow multiple processes to read and write with consistency.

Bonus: Do not make the whole data structure block!

1.3 Module3: Store the data somewhere safe (Soft Deadline: 17th of December)

Now that you have learnt about the filesystem and memory, store the data somewhere safe. Reading and writing directly from files in an LKM is in general not advised. We want you to figure out the best way to make such communication. If the only solution you can find is to do direct read/write to the file system. We will accept that solution.

2 Deliverables and grading

1. During the soft Deadlines, not all team members need to be there. You just need to send us a short report and possibly we can talk in the lab/our office about things.

2. For the real deadline, we will accept submissions and demonstrations between the 9th of January until the 18th of January giving people some slack after Christmas. You will get a Doodle where each group will get to decide when they want to demo their project and submit code.

3. You are supposed to use the Pis, but if you do not, and just show us that it works on the real computer, we will not penalize you. On the other hand, showing it on the Pi can get you some bonus.

4. You will submit

   (a) By email, a report of no more than 10 pages describing your design, your decisions, anything that you find worth mentioning, and a paragraph on what you liked about the project, and a paragraph on what you hated about the project :).

   (b) Running code that you will need to demonstrate in the lab. Uploaded to github!