Operating Systems, 5dv171, Fall 2017

Project Linux Kernel Module - part 1

Teacher
Jan-Erik Moström

Teaching assistent
Adam Dahlgren

Group members
Igor Ramazanov, ens17irv@cs.umu.se
Erik Ramos, c03ers@cs.umu.se
Bastien Harendarczyk, ens17bhk@cs.umu.se

Gitlab repository
https://git.cs.umu.se/c03ers/5dv171-project

2017-09-28

User's guide

The project consists of two parts - the kernel module and the user space test program. All files required to build both parts can be found in the gitlab repository.

To build the project, begin by cloning the repositorybyt typing the following in a terminal:

```
git clone https://git.cs.umu.se/c03ers/5dv171-project
```

In the project directory there are two subdirectories; module and program. Both of these directories have a Makefile for building. Simply enter the directories and type `make` in the terminal.

Once built, a super-user may install the module by typing `insmod moddb.ko` in the terminal and later `rmmod moddb` to uninstall it.

The test program *test*, can be used both for putting data into the module and to retrieve data from it. Inserting data is done like this:

```
./test set key value
```

To retrieve data from the module using the test program, instead write this:

```
./test get key
```

Problem description

The goal with this project is to write a Linux Kernel Module (LKM) that allows different processes running on the same linux machine, to access shared data through key-value mappings, stored in the kernel. The system must be robust enough, that multiple processes, running on multiple processors, can use it without accidentally corrupting the stored data. A user space application must also be implementated to demonstrate the functionality of the LKM.

This first part of the project report describes how the LKM represents these mappings, how the user space applications communicate with the module and what consideratons went into the decisions.

Solution

The following three methods of user to kernel space communication were considered for this project:

1. Device files
2. Proc files
3. Netlink sockets

Using one of the filesystems to solve the problem was appealing, primarly because much of the heavy lifting would be done by system calls already available in Linux. However, since device files usually represent actual, physical devices and proc files are meant to contain information about processes, none of these seemed like a really good fit.

Because the kernel module would provide user space applications with a service, in much the same way that an Internet server would to its clients, Netlink sockets seemed like a much better fit. They would however, turn out to be much more complicated to set up and use.

When it came to storing the key-value pairs in the module, there were two options available; to make a new hashtable datatype, or to find an existing one to use. Not a difficult choice.

A hashtable datatype called *rhashtable* is provided by the Linux kernel and offers all the basic hashtable functionality required at a this stage of the project. It is also designed to run well in multi-threaded environments, which will come in handy later in the project.

Discussion

Developing for the Linux kernel is not an easy task. Much of the information available online is outdated and the kernel in many places, is poorly documented. Most of the time spent on the project, was dedicated to sifting through kernel code, trying to to figure out what the functions do, which functions to use and in what contexts they are safe to use.

Debugging is also difficult, in part because many tools run only in user space. Another complicating factor, is the fact that the operating system can freeze when there is a bug in the code and that error print-outs are lost when rebooting.