Assignment #2: RAID Ramdisk Device Driver

CS 416
Professor - Ricardo Bianchini
TA – William Katsak

Group sizes: exactly 3 people per group

Due date is November 13th, 2011 at 11:55 PM. Late submissions will not be accepted.

Submission: Electronic, via Sakai.

1. Overview

This assignment will give you an opportunity to gain some experience writing operating system level code. You will be required to construct a loadable kernel module (LKM) that can be dynamically loaded into a running Linux kernel. LKM code runs inside the kernel address space (i.e., not in user mode), and can interact with the running kernel in just about any way. Key OS concepts such as execution context, synchronization, and low-level memory management come into play in the construction of LKMs.

The most common usage of LKMs is in the construction of device drivers. There are three main types of device drivers in Linux: character drivers, block drivers, and network drivers. Character and block drivers are accessed via special files, known as “device files”, that live under /dev in the Linux file system, while network drivers are accessed via special APIs. For the purpose of this project, we are most interested in how character and block devices work.

Character device files look just like regular files, and can be “read from” and “written to” using the standard open/close/read/write system calls at the granularity of an individual byte. These types of drivers are used for most devices, including keyboards, mice, webcams, etc.

Block device files also appear under /dev, but these files are different in that they must be read/written at the “block level”, typically 512 bytes at a time. This type of driver is used for mass storage devices like disks, where this block-addressable scheme matches closely with how an actual disk operates.

2. Assignment

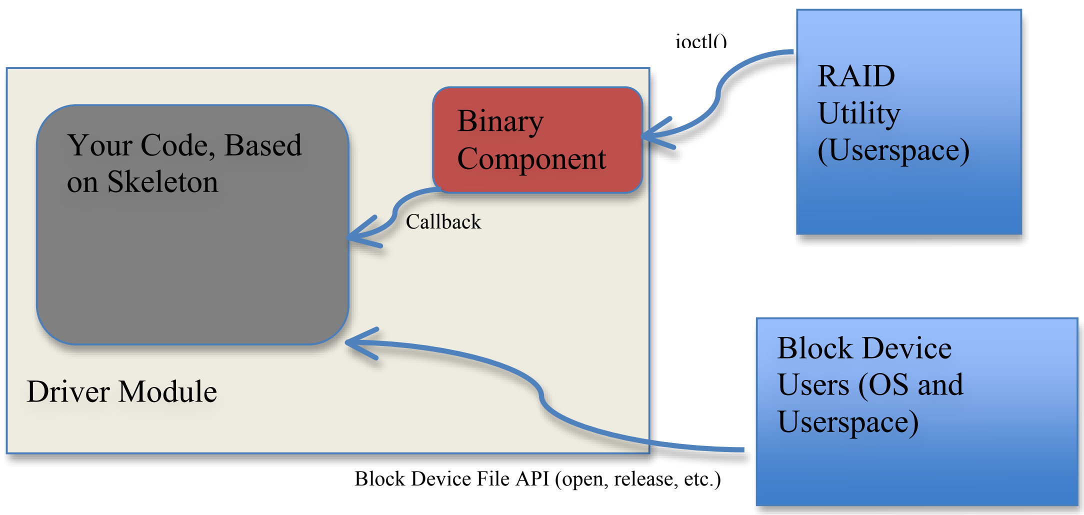
Your assignment will be to complete the implementation of a "RAID Ramdisk" block device driver for the Linux operating system. Your driver will present to the system a single "disk."

We are giving you a skeleton block device driver (heavily commented) that presents you with a set of arrays of memory pages, with each set representing a "virtual disk." You will manage the pages as if each set were an individual hardware disk, and implement RAID5 across the sets. A memory page is 4KB in the x86 architecture, and modern hard disks have 4KB hardware blocks, so this scheme closely mirrors what you would see on a modern disk. Your driver will be compatible with the standard Linux block device API. Because of this, all OS facilities, userspace utilities, etc. that use block devices should work with this driver as if it were an actual hardware disk.

Caveat: In Linux, a "block" is 512 bytes, so you must translate 512-byte block-addresses to the correct location within your array of 4KB blocks.

We are also providing a binary component that is linked into your module. This component will manage the raw pages themselves. In addition, we are providing a command line RAID utility tool that interacts with the binary component through the use of ioctl() system calls, and allows a virtual disk to be "removed" as if a failure occurred. In addition, this component will cause "virtual interrupts" to occur whenever a serious event (such as disk being removed) occurs.

Diagram


ioctl()
RAID Utility (Userspace)
Your Code, Based on Skeleton
Binary Component
Callback
Driver Module
Block Device Users (OS and Userspace)
Block Device File API (open, release, etc.)

To complete the assignment, you need to do the following:

- Implement all functions required by the Linux block driver API.
  - During normal reads and writes, you must maintain whatever metadata is necessary for RAID 5, on the virtual disks themselves (e.g. no additional virtual structures).
- Implement a set of “virtual interrupt handler” functions that will be called from inside the binary component (these will essentially be callbacks).
  - When a disk is removed, you must use your RAID metadata to reconstruct (on the fly), and continue to provide access to the data that was on the missing virtual disk.
  - When the removed disk is replaced with a new, blank one, you must use the metadata to rebuild the new disk.
- Implement whatever locking mechanisms are necessary to ensure consistency and correctness.
- Test your driver under a variety of conditions.
  - You must, at a minimum, be able to partition your virtual disk, create, format, and mount a file system, and verify that data copied into the file system is stored and retrieved correctly.
  - You are also required to verify that all data is maintained after a virtual disk is replaced. This will indicate that your RAID algorithm works.

Honors Students

- Students in the honors section must implement RAID 6 instead of RAID 5. RAID 6 allows for the loss of any two disks by the use of two different parity blocks.
  - Non-honors students may do this for extra credit, provided that the rest of the project is complete and working.

Provided Materials

All necessary materials, including the tarball containing the skeleton and binary component will be attached to the Sakai listing for this assignment.

3. Platform

You will be provided with access to a virtual machine (VM) with an already configured installation of Linux. It is highly recommended that you do all development and experimentation within this VM, due to the high probability that you will crash the kernel at some point.

It is REQUIRED that your final, submitted module work inside the VM to receive credit.

4. Report

Write a report relating the experiences you had, what the key issues that you encountered were, and how you solved each one. The report will be a non-trivial part of your grade, so please prepare it carefully.

At a minimum, the report must include a description of your implementation, including your RAID algorithm and the overall control flow within the driver.

NOTE: the length of the report won't help your grade, but its content will. Be concise, write only what you need to convey.

YOUR REPORT MUST INCLUDE THE NAMES OF ALL MEMBERS OF YOUR GROUP

5. Hint and Recommended Procedure

I recommend that you think carefully about how to implement all functionality before you start. Kernel programming is very unforgiving. An error that would cause a simple “segfault” in user space will usually crash the entire kernel, requiring a reboot of your VM.

6. Submitting Your Assignment

* What? Your report and the entire directory structure that you were provided, with the addition of your source files. Please submit the tar-ed source and the report as two separate files on Sakai.

To re-tar your directory:

```
$ tar cfz yourname.tar.gz source_directory_name
```

* How? Hand in two file called yourname.tar.gz, and yourname-report.txt, respectively, on Sakai:

   - https://sakai.rutgers.edu/portal

PLEASE, ONLY ONE SUBMISSION PER GROUP!

7. Grading

The project will be graded as follows:

(10 points)	No code written, but report is COMPLETE.
(10 points)	Some code. Legitimate effort was made, but driver is non functional.
(20 points)	Driver works at a basic level, RAID not required.
(20 points)	Working driver with support for RAID and virtual disk *removal*.
(40 points)	Fully working driver with FULL RAID 5 (RAID 6 for honors) support, including virtual disk removal and rebuilding of replaced disk.
(25 points) EXTRA CREDIT	Non-honors students may earn an extra 25 points by FULLY implementing RAID 6. No partial extra credit can be earned.

8. Miscellaneous

Useful links

- O'Reilly – Linux Device Drivers, 3rd Edition
  - http://lwn.net/Kernel/LDD3/
- LXR – The Linux Cross Reference
  - http://lxr.linux.no/
- Wikipedia Article: RAID
  - http://en.wikipedia.org/wiki/RAID
  - This page includes a comprehensive explanation of both the RAID 5 and RAID 6 algorithms.