Log Book Week 14

Jose Devian Hibono
1706039603
System Programming - C

This week's material is about writing drivers. First I watched the video and scoured through the internet learning more about the material. Here are the resources I gathered, I tried to write my own understandings in this log book.

Linux Device Drivers

To implement the mechanisms to access the hardware

Types of Devices

1. Network Devices

   They are defined as network interfaces, visible using ifconfig in the user space.

2. Block Devices

   They are used to give access to raw storage devices for user space applications (hard disks, USB keys). In /dev, they are available to the apps as system files.

3. Character Devices

   They are used to provide all other types of devices with access to user space applications (input, sound, graphics, serial, etc.). As system files in /dev, they are also accessible to the applications.

Network Devices

Network devices are called hardware devices that are used to link computers, printers, fax machines and other electronic devices to a network. Such devices transmit data over the same or different networks in a fast, safe and correct manner. Inter-network or intra-network devices can be network devices. Some devices, such as NIC cards or RJ45 connectors, are mounted on the device, while some are part of the network, such as a router, switch, etc.

Block Devices

A block device is one that is designed to function in terms of the Digital UNIX enabled block I/O. It is accessed via the cache of the buffer. A block device has an associated block device driver which uses file system block-sized buffers from a kernel supplied buffer cache to perform I/O. Block device drivers are particularly well-suited to the most popular block devices, disk drives.

Character Devices

A system for characters is any device that can have character streams read from or written to it. A character device has a similar character device driver that can be used with a device that handles one character at a time, such as a line printer. Character drivers, however, are not restricted to executing one single character I/O at a time. Tape drivers often perform I/O in 10K chunks, for example. Where it is appropriate to copy data directly to or from a user process, a character system driver may also be used. Many drivers are character drivers because of their versatility in handling I/Os. Examples of devices requiring character interface drivers are line printers, interactive terminals, and graphics displays.

Security Issues

There are two sides of defense, which can be called intentional and accidental. The harm a user can inflict by misuse of existing programs or by incidentally exploiting bugs is one security issue; a different problem is what kind of (mis)functionality a programmer can intentionally introduce. Obviously, the programmer has much more control than a regular person. In other words, running a program you got from someone else from the root account is as risky as giving him or her a root shell now and then. Although it is not a security vulnerability per se to have access to a compiler, the hole will occur when compiled code is actually executed; everybody should be careful of modules since anything can be achieved by a kernel module. Just as powerful as a superuser shell is a module.

Any system security review is enforced by the kernel code. If there are security holes in the kernel, then there are holes in the device. Only an authorized user can load modules in the official kernel distribution; the `create_module` system call tests whether the invoking method is authorized to load a module into the kernel. Thus, only the superuser, or an attacker who has succeeded in being privileged, can manipulate the power of privileged code while running an official kernel.

Driver writers should, where appropriate, avoid encoding the security policy in their code. Protection is a policy concern that is mostly better managed within the kernel, under the supervision of the system administrator, at higher levels. However, there are still exceptions. As a device driver writer, you should be aware of circumstances where the system as a whole may

be adversely affected by certain forms of device access, and should have appropriate controls. For example, system operations that affect global resources (such as setting an interrupt line) or that may affect other users (such as setting the default size of a block on a tape drive) are typically only accessible to users with appropriate privileges and must be checked by the driver itself.

Of course, driver writers need to be cautious, too, to avoid introducing security bugs. The C programming language makes many types of errors easy to produce. For example, many current security concerns are caused by buffer overrun errors in which the programmer forgets to check how much data is written to a buffer and data ends up being written past the end of the buffer, overwriting unrelated data. The whole system can be corrupted by such errors and must be prevented. Fortunately, it is typically reasonably easy to avoid these errors in the sense of the system driver, where the user interface is narrowly defined and highly controlled.

It's worth keeping any other general security ideas in mind. You should treat any feedback received from user processes with great suspicion; never trust it unless you can check it. Be cautious with uninitialized memory; it is important to zero or otherwise initialize any memory accessed from the kernel before being made available to a user process or computer. Otherwise, leakage of information could result. If your computer interprets data sent to it, make sure that something that might compromise the system can not be sent by the user. Finally, consider the potential impact of computer operations; if there are unique operations that might influence the system (e.g. reloading the firmware on an adapter board, formatting a disk), such operations should ideally be confined to privileged users.

Version Numbering & License

Each package of software used in Linux has a release number. To run a specific version of another program, you need a particular version of one package.

- Must use a GPL-compatible license
- GPL allows anybody to redistribute and sell a product covered by GPL
- GNU General Public License (GPL2)

General Implementation Steps

1. Understanding the device characteristics and supported commands
2. Map device specific operation to unix file operation
3. Select the device name (user interface)
4. Namespace (2-3 characters, /dev/lp0)
5. (optional) Select a major number and minor (a device special file creation) for VFS interface

6. Mapping the number to right device sub-routines
7. Implement file interface subroutines
8. Compile the device driver
9. Install the device driver module with loadable kernel module (LKM) and rebuild/recompile the kernel.

Taken from Course's slide.

Major and minor numbers

A driver associated with the system is identified by the major number. For instance, `/dev/null` and `/dev/zero` are both managed by driver 1, while driver 4 manages virtual consoles and serial terminals; similarly, driver 7 manages both `vcs1` and `vcsa1` machines. In order to dispatch execution to the required driver, the kernel uses the major number at open time.

Only the driver defined by the major number uses the minor number; other parts of the kernel do not use it and simply pass it on to the driver. Multiple devices (as seen in the listing) are popular for a driver to control; the low number provides a way for the driver to distinguish between them.

References

Oreilly. (n.d). *Major and Minor Numbers*. Accessed Dec 22, 2020 from https://www.oreilly.com/library/view/linux-device-drivers/0596000081/ch03s02.html.

Oreilly. (n.d). *Security Issues.* Accessed Dec 22, 2020 from https://www.oreilly.com/library/view/linux-device-drivers/0596000081/ch01s04.html.

System Programming Lecturers (Faculty of Computer Science, Universitas Indonesia). *21 - Writing Drivers*. System Programming Course. Accessed Dec 22, 2020.