Log Book Week 13

Jose Devian Hibono
1706039603
System Programming - C

This week's material is about Kernel Module and Compilation. 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.

Module/Driver Roles in the System

Module

Linux modules are lumps of code that, at any point after booting the machine, can be dynamically connected to the kernel. When they are no longer needed, they can be unlinked from the kernel and removed. Device drivers, pseudo-device drivers such as network drivers, and file-systems are mainly Linux kernel modules.

The Loadable Kernel Module (LKM) is an object file containing code to extend an operating system's running kernel or so-called base kernel. Usually, LKMs are used to add support to new hardware (such as computer drivers) and/or file systems or to add system calls.

Usually, LKM is used to add support or to add device calls to new hardware and/or filesystems. If the LKM feature is no longer required, it can be unloaded to free up memory and other resources.

Driver

A driver is a bit of code that runs to speak to any hardware device in the kernel. Driver code is a big part of a running kernel, and the rest of the code offers generic services such as memory management, IPC, scheduling, etc.

Early in the boot phase, the driver was loaded into RAM by the boot loader at boot time. Before having the capacity to load another module/driver, it helps to jumpstart the system. Some drivers can also be designed as a kernel module so that they can be loaded dynamically and then maybe unloaded later (the one that you compile the kernel with [M])

Unix Design

17 Rules of Unix Programming (taken from https://paulvanderlaken.com/2019/09/17/17-principles-of-unix-software-design/)

1. **Rule of Modularity**: Write simple parts connected by clean interfaces.
2. **Rule of Clarity**: Clarity is better than cleverness.
3. **Rule of Composition**: Design programs to be connected to other programs.
4. **Rule of Separation**: Separate policy from mechanism; separate interfaces from engines.
5. **Rule of Simplicity**: Design for simplicity; add complexity only where you must.
6. **Rule of Parsimon**y: Write a big program only when it is clear by demonstration that nothing else will do.
7. **Rule of Transparency**: Design for visibility to make inspection and debugging easier.
8. **Rule of Robustness**: Robustness is the child of transparency and simplicity.
9. **Rule of Representation**: Fold knowledge into data so program logic can be stupid and robust.
10. **Rule of Least Surprise**: In interface design, always do the least surprising thing.
11. **Rule of Silence**: When a program has nothing surprising to say, it should say nothing.
12. **Rule of Repair**: When you must fail, fail noisily and as soon as possible.
13. **Rule of Economy**: Programmer time is expensive; conserve it in preference to machine time.
14. **Rule of Generation**: Avoid hand-hacking; write programs to write programs when you can.
15. **Rule of Optimization**: Prototype before polishing. Get it working before you optimize it.
16. **Rule of Diversity**: Distrust all claims for “one true way”.
17. **Rule of Extensibility**: Design for the future, because it will be here sooner than you think.