The Soon to be Unmaintainable Linux Kernel

Jason Butz

Computer Science 390

Dr. D. Hwang

October 29, 2008
The Soon to be Unmaintainable Linux Kernel

Linux is an open source operating system that is freely available for download over the Internet. Open Source software is often developed by groups of people without pay as a hobby or as a way to be active. The real definition of open source software is software, and code, that is not restricted in its use and is freely available to everyone without discrimination so long as they follow the rules in the license that it uses. [1] This makes Linux an excellent tool for study since it is easy to obtain and it is a large, ongoing project. Linux is often used when there is research into software engineering, and mainly maintainability. Since so much research has been done on Linux, it has been concluded that due to the way the Linux Kernel is constructed, it may become non-maintainable if actions are not taken.

The Linux Operating System is essentially made up of two parts, the user space and the kernel space. The user space is where the user's applications are run and where the user interacts with the system. The kernel space is made up of the kernel, which can be broken down into different subsystems. At the top of the kernel space is the system call interface. The system call interface, or SCI, allows function calls to be made to the kernel space from the user space. Next you have process management, the virtual file system, memory management, and network stack. Process management acts sort of like a “go between” for the user space and the processor. Memory management helps to ensure that the computer's memory is not over loaded and helps to balance things out. It also controls how programs from the user space access the computer's memory. The virtual file system works with the different file systems supported by the kernel to allow access to the hard drive. The network stack provides an interface for the network and allows the use of different protocols supported by the system. After these are the device drivers and the architecture dependent code. The device drivers allow the system to work with different types of hardware, ensuring that a new version of the kernel is not needed for every piece of hardware that could be in a computer. The architecture dependent code is the code that is used depending upon the architecture of the system in use. [3]
Linux has a long history. The Linux Kernel was originally released in September 1991 with a version number of 0.01. In March of 1994 the Linux Kernel was released with a version number of 1.0. The Linux Kernel is currently at version 2.6.27.1. It has been over 17 years since the Linux Kernel was first conceived. Over these 17 years there has been a great deal of work put into it, and most of that has been maintenance and new feature development.

Whenever the maintainability of Linux is brought up it seems to include a discussion of coupling. Coupling is a measure of the degree to which two modules interact. For example the modules $A$ and $B$ are coupled if they both use the same global variable. There are different types of coupling and they all vary on their impact with respect to a piece of software. For example they can make a module take longer to build, or make it harder to read, or make it prone to breaking. Coupling is a necessary part of programming, but some types are considered more detrimental to software, while others are considered less detrimental.

The least detrimental type of coupling is usually considered to be message coupling, also called loose or low coupling. With message coupling one module interacts with another via some sort of interface, like a function call. The next type is data coupling, which is when all the arguments for two modules are the same. Stamp coupling comes next in the hierarchy. Stamp coupling occurs when a data structure is passed as an argument when only a few elements are needed. Next is control coupling, which is when one module controls the logic of another module. Common coupling is considered the second most detrimental type, and the most common type. Common Coupling is when two modules use the same global data. The most detrimental type of coupling is content coupling, which is also known as high or strong coupling. It occurs when one module directly references the internal workings of another module.

Coupling can have different effects on software. More strongly coupled modules are reported to be more fault-prone. This makes sense because strong coupling can lead to regression faults. Regression faults are when one module is modified and it causes problems in a seemingly unrelated module. Regression faults can cause repeated maintenance, which leads to work to try and repair the faults. The
maintenance can introduce new faults due to the complexity of the code, and this can cause the module to become unmaintainable. Also, with stronger couplings other modules can need maintenance as a result of another module's maintenance. This can cause a cycle that can cause an entire project to become unmaintainable. [9]

When people speak of coupling they often refer to the Linux Kernel since it an easily obtainable, ongoing project. The Linux Kernel is composed of 17 primary modules, and then other less primary modules. [8] When people use the Linux Kernel for research they often look at common coupling. Most researches start looking at version 1.0 and go to the most recent version to see the progress of how coupling has grown. The Linux Kernel has grown immensely over the years. In 1994 version 1.0 of the kernel had 165,165 lines of code, and 487 modules. In 2000 version 2.3.51 of the kernel had 1,690,233 lines of code and 3,857 modules. Now if we take a look at common coupling in the module Panic.c there were 914 instances of common coupling in version 2.1.104, but in version 2.1.105 there were 921 instances. By the version 2.1.109 there were 946 instances. [8] The number of lines of code in the Linux Kernel increases linearly with each version, but the instances of common coupling grows exponentially with each new version. [6] This growth in the number of instances of common coupling is alarming to some people, but also unpopular to others.

Since coupling seems to be related to fault-proneness, we can assume that the Linux Kernel is going to become more fault-prone as the instances of common coupling increase. Since this increase will be exponential it can be assumed that the kernel may become so fault-prone that it is no longer feasible to maintain it. It has been estimated that the cost to develop the Linux Kernel as of version 2.6 by traditional approaches would be around $612 million. [10] This makes it seem very unreasonable to start again from the beginning and redevelop the Linux Kernel. However a more feasible solution would be to restructure the kernel with minimal common coupling. Parts of the kernel are already being restructured, but it is doubtful that it is being done with coupling in mind. Restructuring would put development of the kernel
on hold for months, but it would be necessary. [7] There seems to be no estimate as to what the cost of restructuring would be, however if it were put to a monetary amount the cost would be high.

The Linux Kernel is in trouble. It has an incredibly large number of instances of common coupling, which have been shown to correlate to fault-proneness. Common coupling is merely one type of coupling, and it is a possibility that the Linux Kernel contains other negative types of coupling as well. All of these negative types of coupling make the kernel harder to maintain, which puts stress on the developers. If the problem of common coupling is not stopped then the Linux Kernel will become unmaintainable and will need to be rewritten, or restructured.
Works Cited


   <http://www.bookrags.com/wiki/Coupling_(computer_science)>.


