2003/07/17

Issues for Discussion about Derivative Works in GPL

1. When a program (e.g. application program) statically links with a GPL program (e.g. a library program), should the whole executable program be covered by GPL?

   I believe that such programs should be analyzed according to whether they are a "derived work." There are several tests under U.S. law for what constitutes a derivative work. However, because the issue – as it relates to the GPL – has never been tested in court, we have to rely on the practices in the programming community, and the views expressed by the leaders of that community. How a program links or loads (statically or dynamically) is one factor that must be considered. But other factors must be examined as well. Generally, if a program is written with original (non-GPL) code, and it is distinguishable as a separate program from other GPL works, then I believe the program can be licensed in any manner. An application that simply runs on top of Linux by linking to an LGPL library is clearly accepted as an independent, non-derived work. An application that links to a GPL library is accepted as something that must be licensed under the GPL. In this area, I do not distinguish between static and dynamic linking. Alternatively, I believe that the method of linking/loading does make a difference when considering kernel modules. This is further addressed below.

   If so, is a this because license is required because the whole executable program is deemed to be the derivative work under the US Copyright Act of the GPL program, and the copyright of the author of the GPL program extends to it?

   Please see above.

   Or, is it because you must observe the term of the GPL (Section 2) since you have agreed to it?

   Section 2 of the GPL only addresses (1) redistribution of a program already subject to the GPL and (2) distribution of programs which are determined to be derivative works. So all Section 2 really says is "you must obey U.S. copyright law which gives the copyright holder the rights to control copying, modification and distribution." Section 2 does not try to define what is a derived work. (In fact, the GPL nowhere tries to define a derived work. The GPL defers entirely to U.S. copyright law.)

2. When a program dynamically links with a GPL program, should the program be covered by the GPL?

   Please see answer #1 above. In the application space, I believe that the method of linking is not determinative. In the kernel module/drive space, I believe that dynamic linking can provide a path to proprietary licensing.

3. Should a dynamic loadable module of the Linux kernel which is licensed under the GPL be covered by the GPL?

I believe that modules which load dynamically -- at runtime -- can be licensed under a non-GPL scheme. Alternatively, I believe that modules which are statically bound to the kernel, loading at boot time, are not really distinguishable as separate works and should be licensed under the GPL along with the whole kernel. The GPL/Linux community in general supports this position.

This practice/requirement has been well-debated in public forums. Perhaps the most influential voice, Linus Torvalds, wrote: ""[For] the run-time loading part, ...at which point is a module a derived work of Linux, and therefore under the GPL? [T]here were people that had written drivers for [another operating system]...but they were willing to recompile to provide binaries for Linux. At that point, for moral reasons, I decided I couldn't apply the GPL in this kind of situation. ... We ended up deciding (or maybe I ended up decreeing) that system calls would not be considered to be linking against the kernel. That is, any program running on top of Linux would not be considered covered by the GPL. ... Because of this commercial vendors can write programs for Linux without having to worry about the GPL. The result for module makers was that you could write a proprietary module if you only used the normal interface for loading"

Linus later wrote: "There is NOTHING in the kernel license that allows modules to be non-GPL'd. The _only_ thing that allows for non-GPL modules is copyright law, and in particular the "derived work" issue." Of course, the opposite is also true: there in nothing in the kernel license that REQUIRES modules to be GPL licensed.

Another noted figure, Eric Raymond, went as far as to publicly ask Mr. Torvalds to add new language to the Linux kernel license, including the following passage: "A kernel module loaded at runtime, after kernel build, *is not* to be considered a derivative work."

While there is still some debate around this matter, I believe that the Linux community generally supports (and certainly practices) the idea of proprietary licenses for loadable modules (assuming that those modules are not otherwise considered derived works).

4. Should a device driver, which often takes the form of dynamic loadable module, be covered by GPL?

I would analyze a driver the same as any module. I see kernel modules generally as programs which extend the kernel's functionality. I see drivers as simply one type of kernel module (other types might be file systems, for example).

5. Should the application programs of Linux be covered by the GPL?

Application programs in Linux should be covered by GPL only if the programs are derived works of GPL programs, or if the author of the program otherwise decides to license the program under the GPL. Programmers can write applications to run on Windows® without giving up their rights to Microsoft. Programmers can write

applications to run on Macintosh® without giving up their rights to Apple. Likewise, programmers can write applications to run on Linux and not be subject to the GPL. This position is also taken by Linus himself in the preamble to the GPL that constitutes the Linux kernel license –this "copying" language states that use of the Linux kernel system call interface (the main entry point for user applications) does not confer derivative work status on programs using it.

Of course, if you write a program for Windows® and that program is a derived work of Microsoft-owned code, then Microsoft will control the licensing. This is nothing new in the software world. And if you write a program that is a derived work of a GPL program, the GPL terms will control. But even the FSF sees the commercial needs of Linux developers, and the existence of the LGPL shows their willingness to allow proprietary distribution of Linux applications.

Mr. Linus Tovalds, copyright holder of the Linux kernel, has declared that application programs of Linux need not be covered by the GPL. Should they be covered by the GPL without his declaration?

Mr. Torvalds does not control the GPL, and does not control U.S. copyright law (or any other copyright law). While his voice is important, it is only a guide point to be considered. U.S. copyright law gives Mr. Torvalds the right to control the copying, modification and redistribution of his copyrighted works. He has chosen the GPL as the particular copyright license to pass on his rights. However, no copyright owner can determine for himself what is a derived work (and so what, by law, must be covered by the GPL, for example). However, by choosing to license a program under the GPL, a copyright holder is saying "IF your program is a derived work, it must be distributed under the terms of this license."

6. It is supposed that an embedded system using Linux takes the form of one executable program including application programs. Should it be covered by the GPL as a whole?

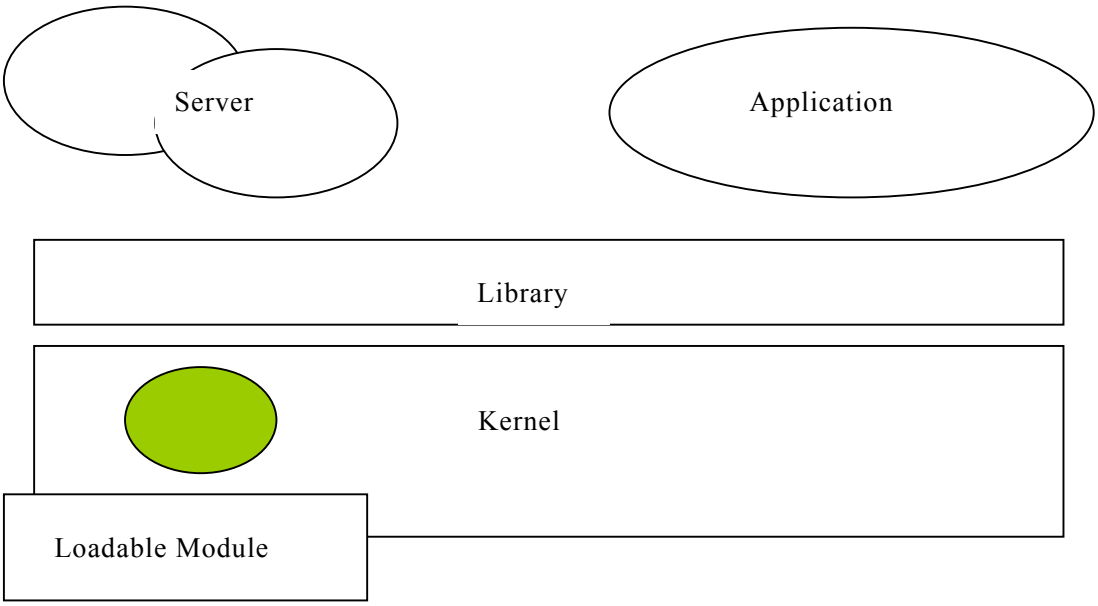
I know of no accepted interpretation of the Linux kernel license or of the GPL that would construe a deployed embedded Linux application, composed of the Linux kernel, modules, libraries, and application code, to be a single program, and it is not MontaVista's position that these components EVER compose a single work per se. This position is defensible by both legal and technical arguments. On the legal front, the various components in such an aggregation each have an independent existence and derivation and are only coincidentally collocated in an end application system – this collocation does not impact their free-standing status as individual works. Technically speaking, the kernel in such an aggregation can run and function without the other components, and in many cases the application code, the libraries and even the device drivers have independent existences and often can execute in other environments – as such, their coincidental joint execution does not negate their independent technical existence.

7. When using the C or C++ language to write programs, “header files” which define macros and data structures are needed. If header files distributed under the GPL are used, should the program be covered by the GPL?

Header files define interfaces and declare other information, including licensing status, about the program to which they belong. Header files themselves are not programs and generally are not attributed the status of intellectual property. Moreover, interfaces themselves do not have IP status nor are licensed nor copyrighted, although the code that implements those interfaces can be copyrighted and licensed. As such, headers cannot have a different license status than the code they describe, and indeed have no licensing status of their own.

If header files are “loaded” so as to contain actual code (and thereby IP), those files represent an aberration, created through generally unaccepted programming practices (or possibly even a direct intent to “pollute” the code that includes them).

A valid set of exceptions includes headers that implement macros and in-line functions in C++ or via extensions to C in the gcc compiler. In those cases, however, the inclusion of program fragments (macros and in-line functions) are an implementation choice and not an indication that the interface must be regarded as IP (derived or otherwise) itself.


Server
Application
Library
Kernel
Loadable Module