Concepts and misconceptions of (L)GPL installation obligations

Carsten Emde
Open Source Automation Development Lab (OSADL) eG

COMPACT
OSADL
ONLINE
LECTURES

The author of the General Public Licenses

To understand the “spirit” of the General Public License (and particularly the obligations with respect to installation), we should know a bit about the author:

“Richard Matthew Stallman [..] was born in New York in 1953. He is a physicist, computer scientist, philosopher, and a passionate champion for software freedom.”

“In 1971, while still a student at Harvard, he started working as a programmer at the MIT Artificial Intelligence Lab (currently, CSAIL). In 1983, he launched the GNU Project with the goal of building a free software operating system (known today as GNU/Linux). Soon after that, in January 1984, Stallman quit his job at MIT so that the institution could not claim any rights on his work on GNU.”

From https://stallmansupport.org/who-is-richard-stallman.html

The importance of freedom 0

- “Since I am not a pacifist, I would also disagree with a ‘no military use’ provision. I condemn wars of aggression but I don't condemn fighting back.”
- “Since I am not against business in general, I would oppose a restriction against commercial use. A system that we could use only for recreation, hobbies and school is off limits to much of what we do with computers.”
- “The conclusion is clear: **a program must not restrict what jobs its users do with it. Freedom 0 must be complete**. We need to stop torture, but we can't do it through software licenses. The proper job of software licenses is **to establish and protect users' freedom.**”

From: https://www.gnu.org/philosophy/programs-must-not-limit-freedom-to-run.en.html

The importance of freedom 0

- "Since I am not a pacifist, I would also disagree with a 'no military use' provision. I condemn wars of aggression but I don't condemn fighting back."
- "Since I am not against business in general, I would oppose a restriction against commercial use. A system that we could use only for recreation, hobbies and school is off limits to much of what we do with computers."
- "The conclusion is clear: **a program must not restrict what jobs its users do with it. Freedom 0 must be complete.** We need to stop torture, but we can't do it through software licenses. **The proper job of software licenses is to establish and protect users' freedom.**"

From: https://www.gnu.org/philosophy/programs-must-not-limit-freedom-to-run.en.html

- The freedom to run the program as you wish, for any purpose (freedom 0).
- The freedom to study how the program works, and change it so it does your computing as you wish (freedom 1). Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help others (freedom 2).
- The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.

Quoting Richard Stallman's ethics

"Things like freedom and the expansion of knowledge are beyond success, beyond the personal. Personal success is not wrong, but it is limited in importance, and once you have enough of it, it is a shame to keep striving for that instead of for truth, beauty, or justice."

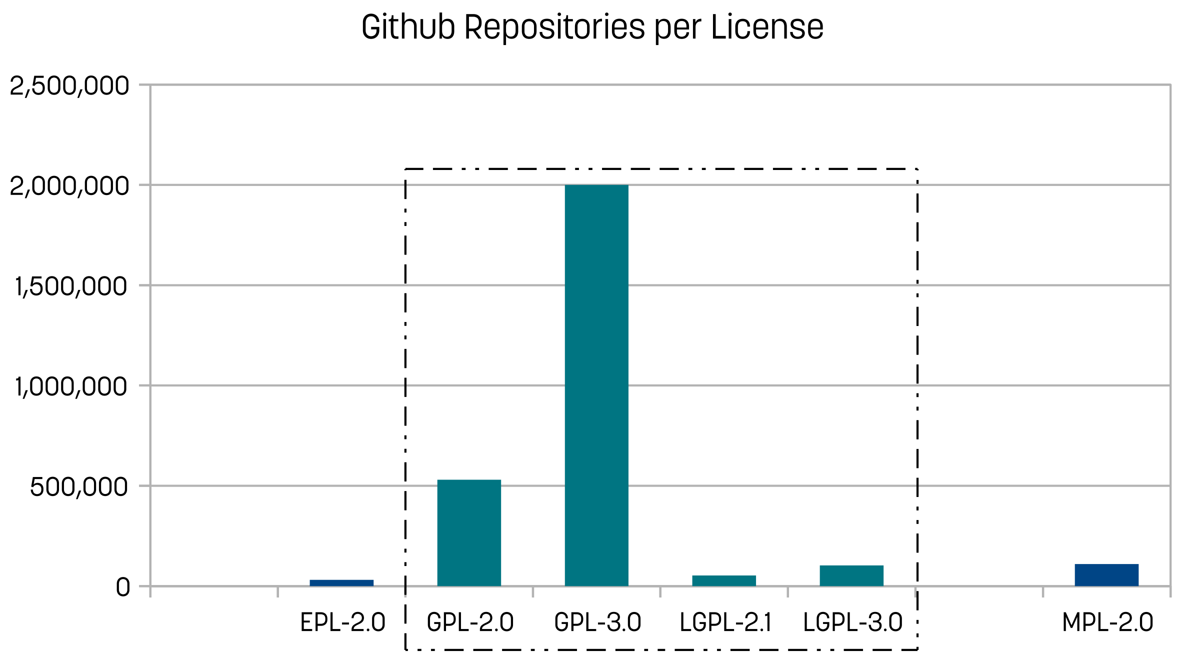
(Richard Stallman in "Free Software as a Social Movement", December 18, 2005)
from https://stallmansupport.org/who-is-richard-stallman.html

History of the General Public Licenses

License	Published	Remarks
GPL-1.0	February 1989	No longer used for new projects
GPL-2.0	June 1991	In use (e.g. Linux kernel)
GPL-3.0	June 2007	In use (e.g. bash)
LGPL-2.0	June 1991	No longer used for new projects, “L” stands for “Library”
LGPL-2.1	June 1999	In use (e.g. GNU C library), “L” stands for “Lesser”
LGPL-3.0	June 2007	In use (e.g. GNU gzip library), “L” stands for “Lesser”

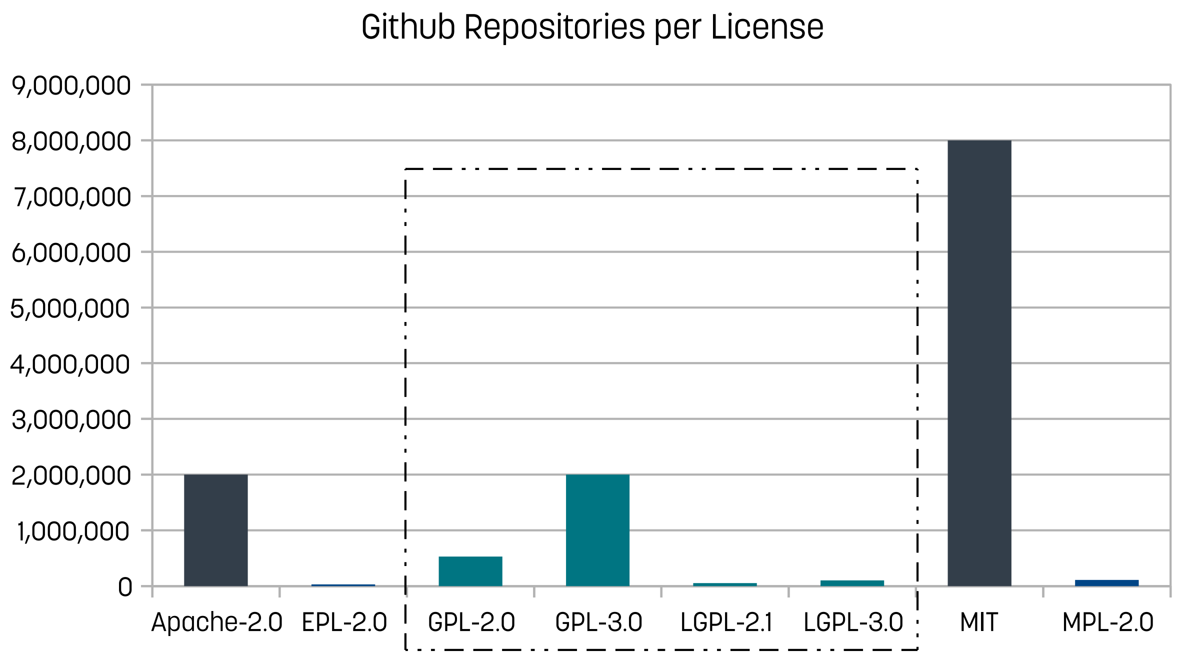
Acceptance of the General Public Licenses

Comparison with other copyleft licenses



Acceptance of the General Public Licenses

Comparison with other licenses



Quick refresher on GPL-2.0 and GPL-3.0

- The General Public Licenses **GPL-2.0** and **GPL-3.0** are Open Source software licenses with an **unrestricted ("strong")** copyleft:
  - Any modification or extension of an existing file must be licensed under the original license.
  - New files with code that depends on original files or vice versa **must be licensed <u>under the original license</u> when the original and new material are supplied together.**

Quick refresher on LGPL-2.1 and LGPL-3.0

- The Lesser General Public Licenses **LGPL-2.1** and **LGPL-3.0** are Open Source software licenses with a **restricted ("weak")** copyleft:
  - Any modification or extension of an existing file must be licensed under the original license.
  - New files with code that depends on original files **may be licensed independently** **as long as the original and new material can be separated at any time or the new code is provided as a linkable object (e.g. for static linking).**

Quick refresher on LGPL-2.1 and LGPL-3.0

- The Lesser General Public Licenses **LGPL-2.1** and **LGPL-3.0** are Open Source software licenses with a **restricted (“weak”)** copyleft:
  - Any modification or extension of an existing file must be licensed under the original license.
  - New files with code that depends on original files **may be licensed independently as long as the original and new material can be separated at any time or the new code is provided as a linkable object (e.g. for static linking).**
  - Exceptionally, this license also imposes obligations on the otherwise independently licensed new material.

Quick refresher on the non-difference between GPL-2.0 and GPL-3.0

- Obligation to allow the recipient to re-install the software if it comes installed
  - GPL-2.0: Yes
  - GPL-3.0: Yes

Quick refresher on the non-difference between GPL-2.0 and GPL-3.0

- Obligation to allow the recipient to re-install the software if it comes installed
  - GPL-2.0: Yes
  - GPL-3.0: Yes

'The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, **plus the scripts used to control compilation and installation of the executable.**'

Quick refresher on the non-difference between GPL-2.0 and GPL-3.0

- Obligation to allow the recipient to re-install the software if it comes installed
  - GPL-2.0: Yes
  - GPL-3.0: Yes

'"Installation Information" for a **User Product** means any **methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work** in that User Product from a modified version of its Corresponding Source.'

Quick refresher on the non-difference between GPL-2.0 and GPL-3.0

GPL-2.0: Yes

‘The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, **plus the scripts used to control compilation and installation of the executable.**’

GPL-3.0: Yes

‘"Installation Information" for a **User Product** means any **methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work** in that User Product from a modified version of its Corresponding Source.’

Quick refresher on the <u>difference</u> between GPL-2.0 and GPL-3.0

- Duration of the source code disclosure obligations

Quick refresher on the difference between GPL-2.0 and GPL-3.0

- Duration of the source code disclosure obligations
  - GPL-2.0: Three years after
    the most recent public release

Quick refresher on the difference between GPL-2.0 and GPL-3.0

- Duration of the source code disclosure obligations
  - GPL-2.0: Three years after the most recent public release
  - GPL-3.0: **At least** three years after the most recent public release **plus** as long as the software is considered “a product”, i.e. as long as spare parts can be purchased from the original vendor or the vendor provides support

The GPLs

Installation related obligations and freedom

- Let's take a look at a realistic scenario:
  - A manufacturer of a Linux-based router with Secure Boot is discontinuing support after five years.
  - Shortly after, a vulnerability was discovered in the Linux kernel that allowed an attacker to gain access to the router by sending a maliciously manipulated network packet to the router.
  - The Linux community has released a patch.
  - The owner of the router should be free to apply a patched kernel to the device (for sustainability reasons alone).
  - The license therefore stipulates that this **freedom** is granted.

Concerns raised by manufacturers

- Safety regulations do not allow to permit individual installation.
- Security regulations do not allow to permit individual installation.
- Only a minority of users will ever be able to compile the Linux kernel and install it on the device, so this effort is not justified.
- A user lacks knowledge and procedures to ensure safe and secure operation of the patched device.
- Users must be forced to buy new devices from time to time to ensure innovation and keep the economy running.
- We need to encrypt the code to protect our IP.
- The device is ready for the market, retroactively enabling installation sets us back many months.

Safety regulations do not allow to permit individual installation

- Safety regulations primarily protect users, not manufacturers. Therefore, in certain markets a device may not be given to a user, if it is not appropriately certified.
- However, a person may apply modifications to a device that violate safety requirements and use it on their own risk. This is not limited by a safety regulation or by law. Accordingly, in Germany and in many other countries, self-injurious behavior or even suicide is not punishable.
- The GPLs nowhere require that the device must be able to be certified for safety after the modification.

Security regulations do not allow to permit individual installation

- Security regulations primarily protect users, not manufacturers. Therefore, in certain markets a device may not be given to a user, if it is not appropriately certified.
- However, a person may apply modifications to a device that violate security requirements and use it on their own risk. This is not limited by a security regulation or by law.
- The GPLs nowhere require that the device must be able to be certified for security after the modification.

Only a minority of users will ever be able to compile the kernel

- It is not the responsibility of the original manufacturer to provide training or special support. The only requirement is that the materials indispensable for reloading new software, such as scripts and keys, are provided, and no contractual prohibitions on software reloading are imposed.
- If a buyer is unable to repair the unit, they can ask others for assistance. If they don't get support: Bad luck.

A user lacks knowledge to ensure safe and secure operation

- It is not the responsibility of the original equipment manufacturer to provide training on safety and security. If the user feels that they cannot ensure safe and secure operation, they are well advised not to make any changes to the device.

Users must be forced to buy new devices from time to time

- At first glance, this may sound a bit strange, but the entire list of concerns would not be complete without mentioning this. There is no question that cold discontinuation and planned obsolescence are parts of the toolbox of capitalism – whether we appreciate it or not.
- It is hardly to be expected that the increasing use of Open Source software will change this. The distribution of GPLs is too low for this, and if these licenses are actually used, then mostly in conjunction with other proprietary software, which the manufacturer can discontinue at any time.

Users must be forced to buy new devices from time to time

- At first glance, this may sound a bit strange, but the entire list of concerns would not be complete without mentioning this. There is no question that cold discontinuation and planned obsolescence are parts of the toolbox of capitalism – whether we appreciate it or not.
- It is hardly to be expected that the increasing use of Open Source software will change this. The distribution of GPL3 is too low for this, and if these licenses are actually used, then mostly in conjunction with other proprietary software, which the manufacturer can discontinue at any time.

For example, since September 1 of this year, many smartphones equipped with GPL licensed software, for which update was discontinued, are no longer accepted for electronic banking and must be disposed of. Users are forced to buy new devices.

We need to encrypt the code to protect our IP

- IP that needs to be protected is better not included in GPL licensed software, e.g. not in a Linux driver, but in a user-space program.
- Instead of the entire image, encryption can be applied only to individual areas of the image, e.g. to the proprietary software that is to be protected.

Enabling installation sets us back many months

- The answer to this concern sounds very know-it-all, but it is no question that equipping a device with a mechanism to let users reinstall GPL-licensed software is much easier if this is done from the very beginning of the device development than having to do so retroactively.
- By the way: It should be part of a company's FOSS policy that whenever it is decided to use a particular Open Source software in a product, the requirements of the license are evaluated and care is taken to ensure that they can be met.

Concerns raised by manufacturers

- Safety regulations do not allow to permit individual installation. ✓
- Security regulations do not allow to permit individual installation. ✓
- Only a minority of users will ever be able to compile the Linux kernel and install it on the device, so this effort is not justified. ✓
- A user lacks knowledge and procedures to ensure safe and secure operation of the patched device. ✓
- Users must be forced to buy new devices from time to time to ensure innovation and keep the economy running. ✓
- We need to encrypt the code to protect our IP. ✓
- The device is ready for the market, retroactively enabling installation sets us back many months. ✓

The LGPLs

Quick refresher of LGPL-2.1 additional provisions

- LGPL-2.1 exceptionally not only imposes obligations on the licensed work, but even on a work linked to it, although the other work may be licensed under a completely different license:
- Article 6: As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms **permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.**

Quick refresher of LGPL-3.0 additional provisions

- LGPL-3.0 exceptionally not only imposes obligations on the licensed work, but even on a work linked to it, although the other work may be licensed under a completely different license:
- Article 4 Combined Works: You may convey a Combined Work under terms of your choice that, taken together, effectively **do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications** …

Obligation to permit modification and freedom

- Let's take a look at another scenario, this time involving LGPL and let’s assume we are in the year 2035:
  - A very old and long neglected 32-bit device is still indispensable and will probably be used after January 19, 2038.
  - If no measures are taken, the system will crash at this date, because on Unix systems the time is stored in a signed 32-bit variable that contains the number of seconds after January 1, 1970, 00:00. It will reach its maximum on January 19, 2038 at 04:14:07 CET and wrap to the smallest negative 32-bit number.

Check the Y2038 bug

```c
#include <stdio.h>
#include <time.h>
#include <string.h>

int main(void)
{
  long l32 = 0x7fffffff;
  time_t t = (time_t) l32;
  char *c;

  printf("Bit length of time_t: %ld\n",
    (long) sizeof(time_t) * 8);
printf("%s", ctime(&t));
t++;
c = ctime(&t);
printf("%s", c);
if (strstr((const char *)c, "2038") == NULL)
  printf("Result: NOT OK\n");
else
  printf("Result: ok\n");

  return 0;
}
```

Running the test program ...

... on an affected system

```
Bit length of time_t: 32
Tue Jan 19 04:14:07 2038
Fri Dec 13 21:45:52 1901
Result: NOT OK
```

... on a sane system

```
Bit length of time_t: 64
Tue Jan 19 04:14:07 2038
Tue Jan 19 04:14:08 2038
Result: ok
```

Running the test program ...

We run the program regularly on the OSADL QA Farm devices and display the result in the profiles.

... on an affected system

```
Bit length of time_t: 32
Tue Jan 19 04:14:07 2038
Fri Dec 13 21:45:52 1901
Result: NOT OK
```

... on a sane system

```
Bit length of time_t: 64
Tue Jan 19 04:14:07 2038
Tue Jan 19 04:14:08 2038
Result: ok
```

Running the test program ...

It may be interesting to know that the 64-bit variable will wrap in about 292 billion years from now.

... on an affected system

```
Bit length of time_t: 32
Tue Jan 19 04:14:07 2038
Fri Dec 13 21:45:52 1901
Result: NOT OK
```

... on a sane system

```
Bit length of time_t: 64
Tue Jan 19 04:14:07 2038
Tue Jan 19 04:14:08 2038
Result: ok
```

Repairing the GNU C library

- To fix the Y2038 bug, the C library must be recompiled with the Gnulib module 'year2038' and reinstalled.
- Replacing an important language library like the C library requires extensive knowledge about the library and the architecture:
  - The time variable may be used by proprietary applications and libraries.
  - Bugs may have been fixed in library functions, making earlier workarounds obsolete.
  - To overcome the above challenges, it may be required to modify the proprietary application and reverse engineer it for debugging the modifications.

Modify the proprietary (binary) application?

- The proprietary application linked to the C library is only available in binary form, and it is the important exception of the LGPLs that they do not impose the obligation to disclose its source code.
- How can a program in binary form be modified?

Modify the proprietary (binary) application?

- The proprietary application linked to the C library is only available in binary form, and it is the important exception of the LGPLs that they do not impose the obligation to disclose its source code.
- How can a program in binary form be modified?

Binary patching!

How does binary patching work?

- Let's assume the following code sequence of a call into a library:
  - Prepare the stack
  - Call a subroutine in the C library that was exchanged
  - Evaluate the return value

How does binary patching work?

- Let’s assume the following code sequence of a call into a library:
  - Prepare the stack
  - Call a subroutine in the C library that was exchanged
  - Evaluate the return value

We may need to insert code here in order to consider changes in the function call.

How does binary patching work?

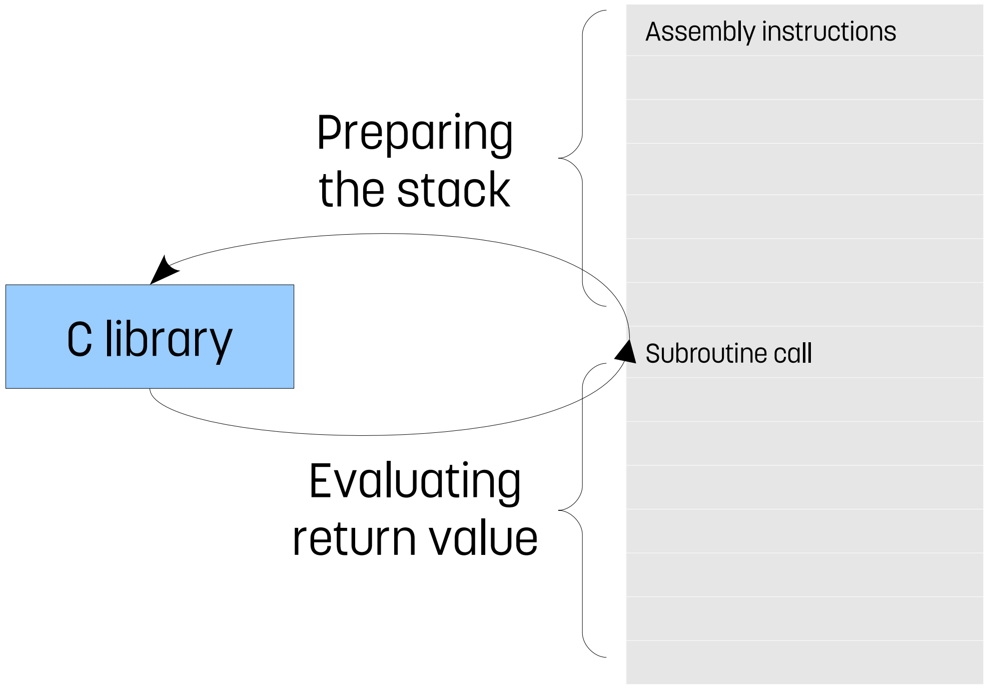
- Let's assume the following code sequence of a call into a library:
  - Prepare the stack
  - Call a subroutine in the C library that was exchanged
  - Evaluate the return value

We may need to insert code here in order to consider changes in the function call.

We may need to insert code here in order to consider changes in the return value

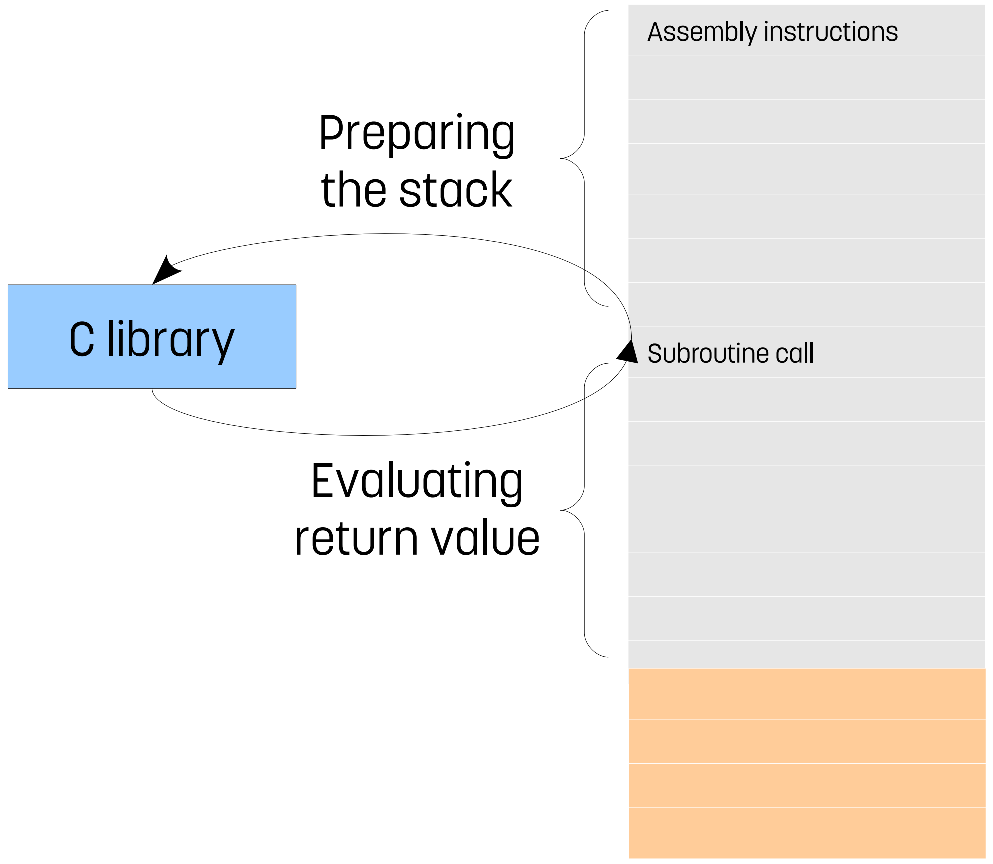
How does binary patching work?

- Steps to do



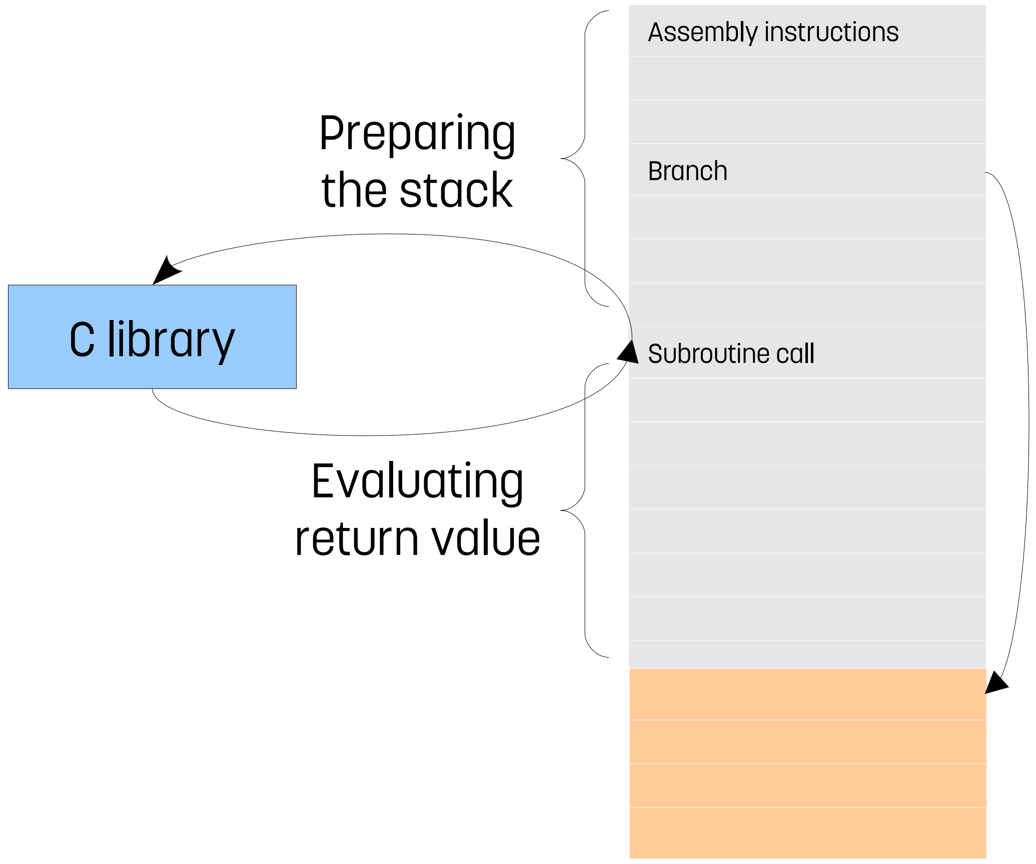
How does binary patching work?

- Steps to do
  - Make the program longer



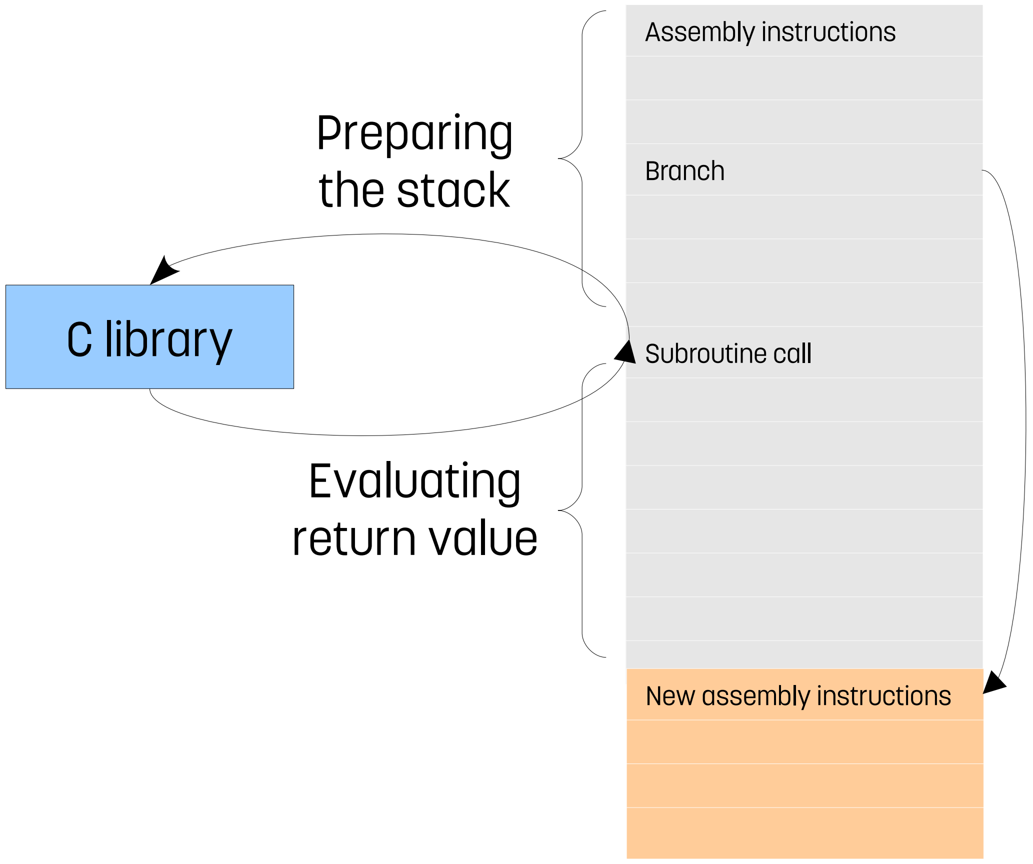
How does binary patching work?

- Steps to do
  - Make the program longer
  - Overwrite the code that needs to be extended with an unconditional branch operation to the new space at the end of the program



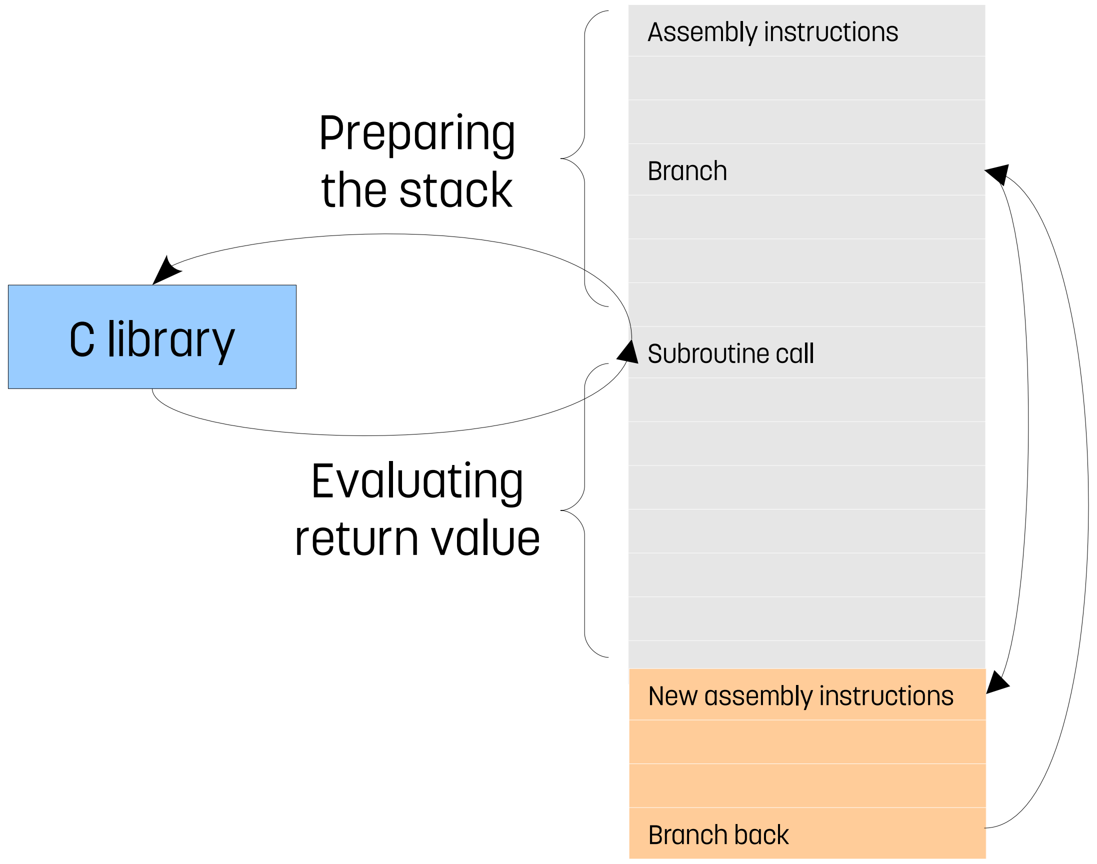
How does binary patching work?

- Steps to do
  - Make the program longer
  - Overwrite the code that needs to be extended with an unconditional branch operation to the new space at the end of the program
  - Add code that fixes the library incompatibility



How does binary patching work?

- Steps to do
  - Make the program longer
  - Overwrite the code that needs to be extended with an unconditional branch operation to the new space at the end of the program
  - Add code that fixes the library incompatibility
  - Finish the work with a branch command to jump back to the initial program location



Is binary patching still feasible?

- At the time LGPL-2.1 was written, code was less optimized and – especially for complex instruction sets – binary patching was possible, although always rather tedious.
- Today, binary patching is usually no longer possible because binary code is normally highly optimized and reduced instruction set architectures are more common.
- This is probably the reason why LGPL-3.0 no longer requires that modifications to the binary code of the proprietary application be allowed.

Conclusion GPLs

- The GPLs' obligations to allow and enable reinstallation of the software was not created primarily to annoy companies that decided to copy and distribute such software.
- The purpose of this regulation is rather to give the user of the software the greatest possible freedom and independence – in the same way as if the software had been delivered in source code form.

Conclusion LGPLs

- The LGPLs' additional obligations that relate to the linked proprietary software are also intended to give the user of the software the greatest possible freedom and independence.
- The requirement in LGPL-2.1 that modifications to proprietary software must be permitted, e.g. by binary patching, is now out of date and has therefore been abandoned in LGPL-3.0.
- Considering the fact that binary patching is hardly possible nowadays and also not reasonable without seriously endangering the function of the software, it should actually not be a big hurdle to give this permission.

Conclusion LGPLs

- The LGPLs' additional obligations that relate to the linked proprietary software are also intended to give the user of the software the greatest possible freedom and independence.
- The requirement in LGPL-2.1 that modifications to proprietary software must be permitted, e.g. by binary patching, is now out of date and has therefore been abandoned in LGPL-3.0.
- Considering the fact that binary patching is hardly possible nowadays and also not reasonable without seriously endangering the function of the software, it should actually not be a big hurdle to give this permission.

Finally, denying this permission certainly would not deter criminals from doing it.