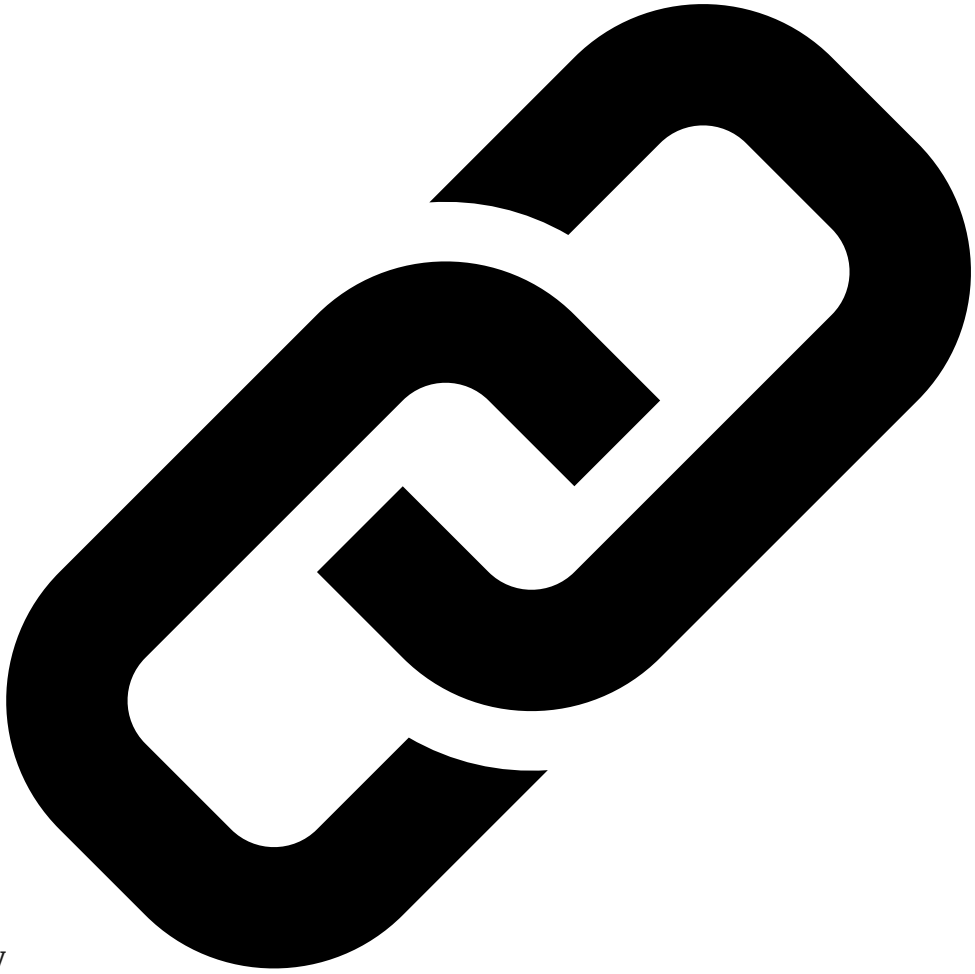
What Is the LD_PRELOAD Trick? | Baeldung on Linux

By baeldung

Published: 2020-08-09 · Archived: 2026-04-05 22:21:58 UTC



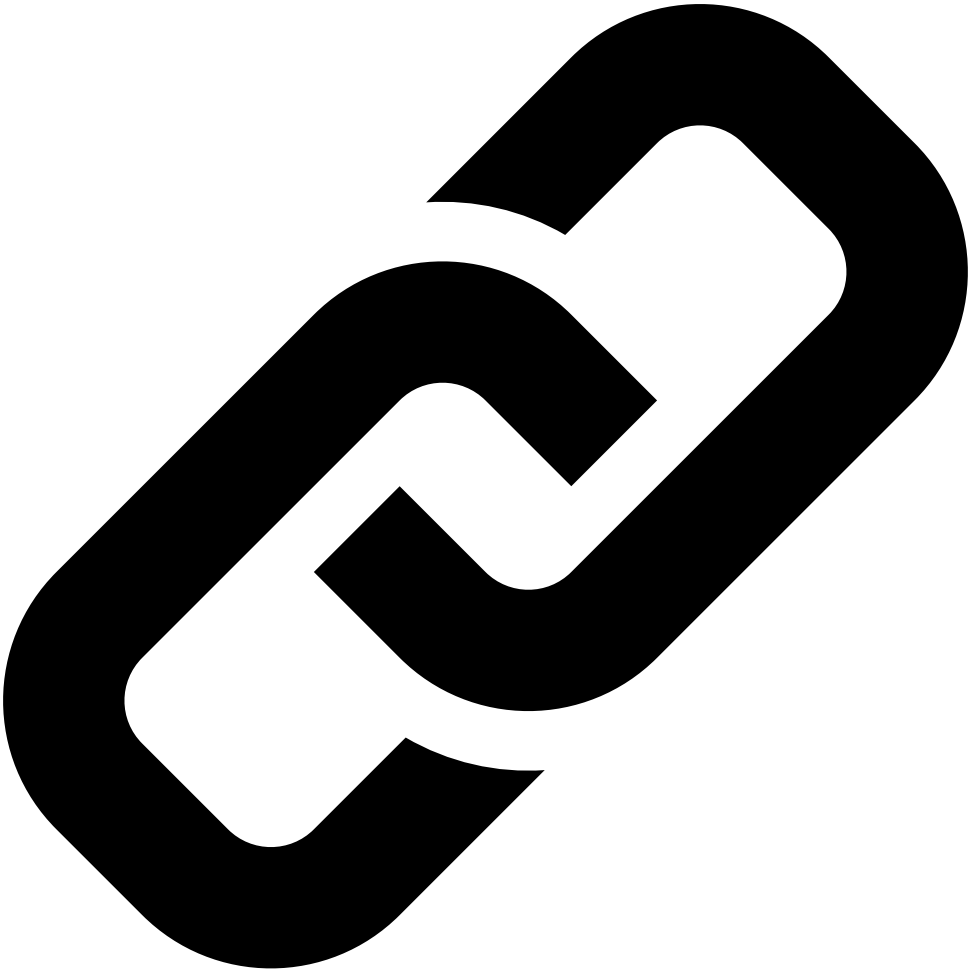
1. Overview

The ***LD_PRELOAD* trick is a useful technique to influence the linkage of shared libraries and the resolution of symbols (functions) at runtime.** To explain *LD_PRELOAD*, let's first discuss a bit about libraries in the Linux system.

In brief, a library is a collection of compiled functions. We can make use of these functions in our programs without rewriting the same functionality. This can be achieved by either including the library code in our program (static library) or by linking dynamically at runtime (shared library).

Using static libraries, we can build standalone programs. On the other hand, programs built with a shared library require runtime linker/loader support. For this reason, **before executing a program, all required symbols are loaded and the program is prepared for execution**.

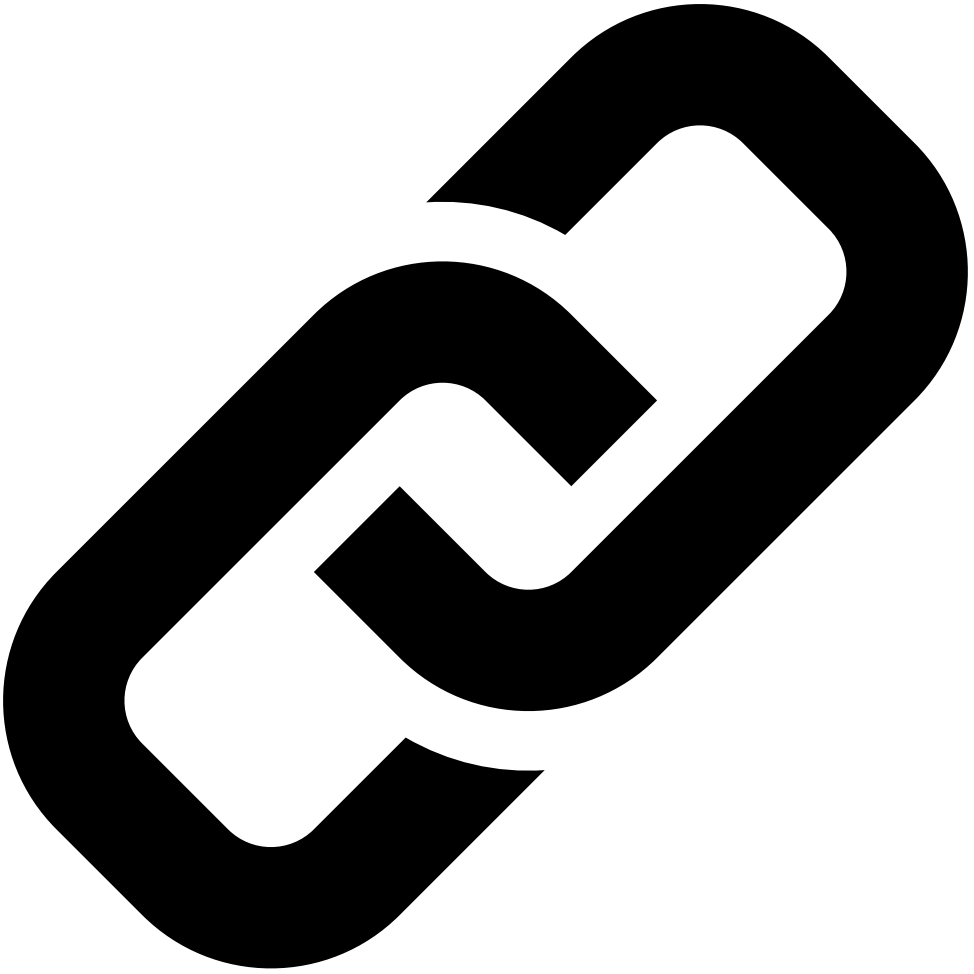
2. Runtime Execution Environment



The *LD_PRELOAD* trick comes handy in the program execution preparation phase. Linux system programs *ld.so* and *ld-linux.so* (dynamic linker/loader) use *LD_PRELOAD* to load specified shared libraries. In particular, before any other library, the dynamic loader will first load shared libraries that are in *LD_PRELOAD*.

It's important to note that in secure-execution mode, entries in *LD_PRELOAD* can be ignored. This happens when a slash appears in the path — meaning the file is not in the default search path. In this case, the dynamic loader will only load a system library if the library has the *setuid* bit set.

3. Use Cases of *LD_PRELOAD*



In this section, we'll try some use cases of the *LD_PRELOAD* trick. Firstly, we will see how libraries can be overridden. Later on, we'll use *LD_PRELOAD* to interpose (wrap around).

Since *LD_PRELOAD* is an environment variable, it affects only the current process. Therefore, we will only use absolute paths.

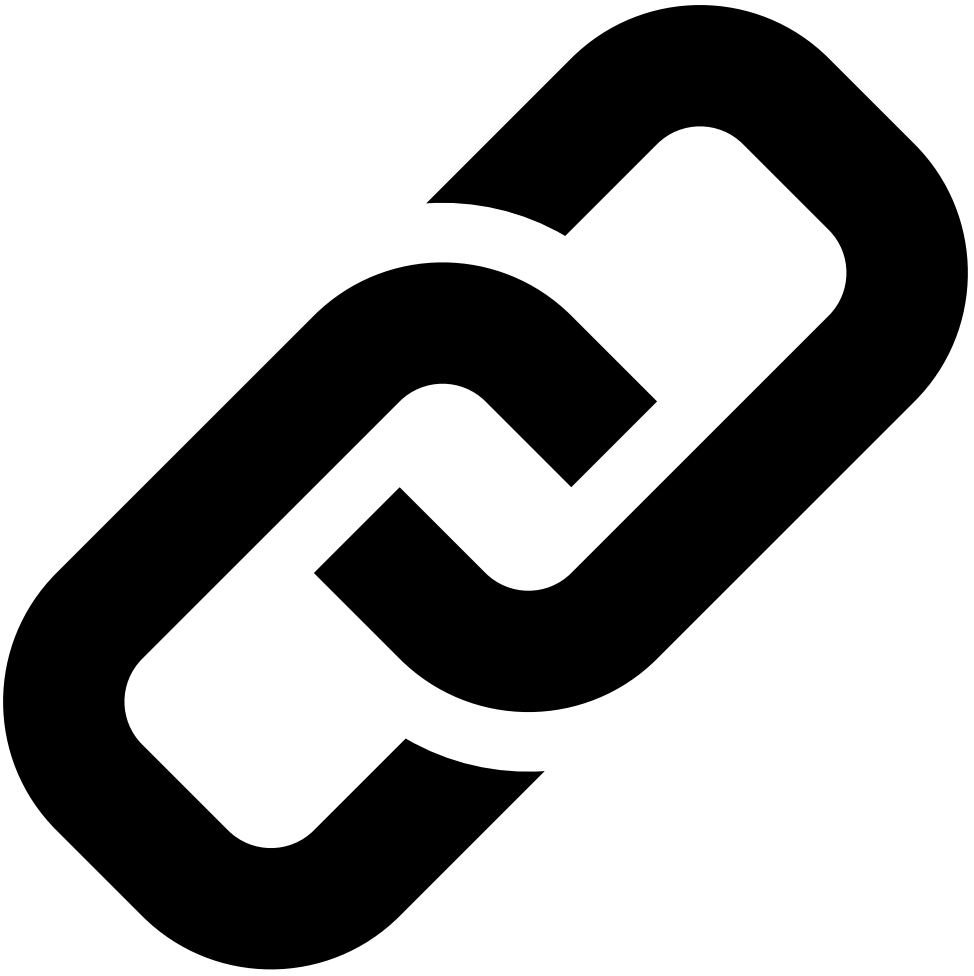
Before getting into the use of *LD_PRELOAD*, let's first use the *ldd* command. **The *ldd* command is useful for listing runtime dependencies of a binary program or shared library**:

```
$ ldd /usr/bin/pigz
        linux-vdso.so.1 (0x00007ffd559d7000)
        libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007fa48bcc1000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007fa48bc9e000)
        libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007fa48bc82000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fa48ba90000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fa48c065000)
```

Using *ldd*, we can see that the *pigz* program depends on several shared libraries.

We also have other options for seeing used shared libraries.

3.1. Replacing Libraries Using *LD_PRELOAD*



Let's now get back to the terminal and see the *LD_PRELOAD* trick in action:

```
$ LD_PRELOAD=/data/preload/lib/libz.so.1.2.7 ldd /usr/bin/pigz
        linux-vdso.so.1 (0x00007ffc1d9c4000)
        /data/preload/lib/libz.so.1.2.7 (0x00007f33877d9000)
        libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f3387674000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f3387651000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f338745f000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f3387a32000)
```

Now, we can see that **the *zlib* (*libz*) dependency of the *pigz* program is changed, from the system default location to the location that we have set in *LD_PRELOAD***.

If we want to run multiple programs with *LD_PRELOAD*, it will be better to use *export* and *unset* to set and clear the environment variable, respectively. Let's do this now and confirm the change by invoking the *pigz* program itself:

```
 $ pigz -vV
pigz 2.4
```

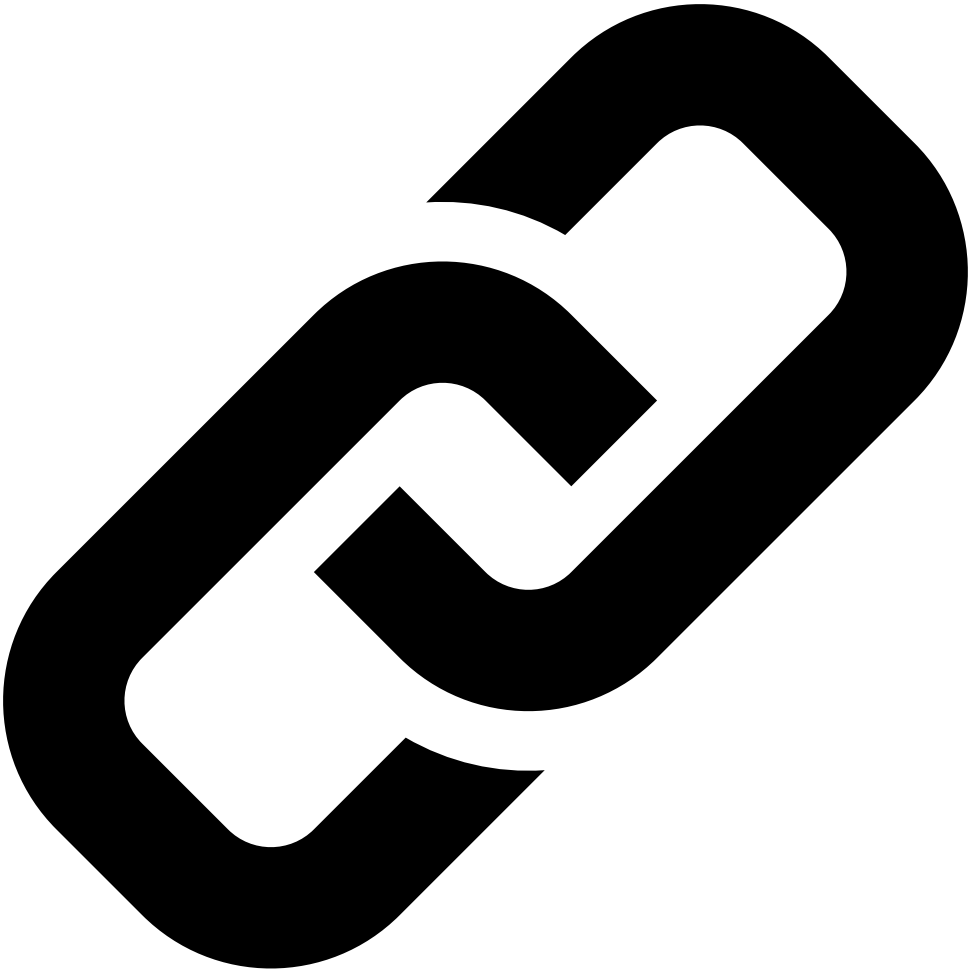
```
zlib 1.2.11

$ export LD_PRELOAD=/data/preload/lib/libz.so.1.2.7
$ pigz -vV
pigz 2.4
zlib 1.2.7

$ unset LD_PRELOAD
$ pigz -vV
pigz 2.4
zlib 1.2.11
```

Here, we invoked *pigz* with the *-vV* options so that it reports both the *pigz* and *zlib* versions.

3.2. Interposing Using *LD_PRELOAD*



Let’s use an interposer library to change the behavior of a system function. For this example, we’ll use an imaginary library that adds some accounting functionality to *malloc*.

Using the *LD_PRELOAD* trick, we can change the behavior of any program that is dependent on an external shared library. For example, the *ls* program depends on *libc*, which provides many system functions, including memory allocation using *malloc*:

```
 $ ls -lh
total 2,8G
-rw-rw-r-- 1 baeldung_user baeldung_user 3,6K Jul  4 20:16 BVF_Density.ipynb
-rwxrwxr-x 1 baeldung_user baeldung_user 2,8G Jul  5 15:59 cuda_11.0.1_450.36.06_linux.run
drwxrwxr-x 2 baeldung_user baeldung_user   25 Jul  8 16:12 h5store
drwxrwxr-x 2 baeldung_user baeldung_user   30 Jul  5 18:47 pandas_processing
drwxrwxr-x 4 baeldung_user baeldung_user  142 Jul 19 15:59 preload
$ LD_PRELOAD=/data/preload/lib/malloc_interpose.so ls -lh
malloc(20000)  call number: 223
malloc(32816)  call number: 226
total 2,8G
-rw-rw-r-- 1 baeldung_user baeldung_user 3,6K Jul  4 20:16 BVF_Density.ipynb
-rwxrwxr-x 1 baeldung_user baeldung_user 2,8G Jul  5 15:59 cuda_11.0.1_450.36.06_linux.run
drwxrwxr-x 2 baeldung_user baeldung_user   25 Jul  8 16:12 h5store
drwxrwxr-x 2 baeldung_user baeldung_user   30 Jul  5 18:47 pandas_processing
drwxrwxr-x 4 baeldung_user baeldung_user  142 Jul 19 15:59 preload
```

With our interposer library preloaded, the output of the *ls* command is different.

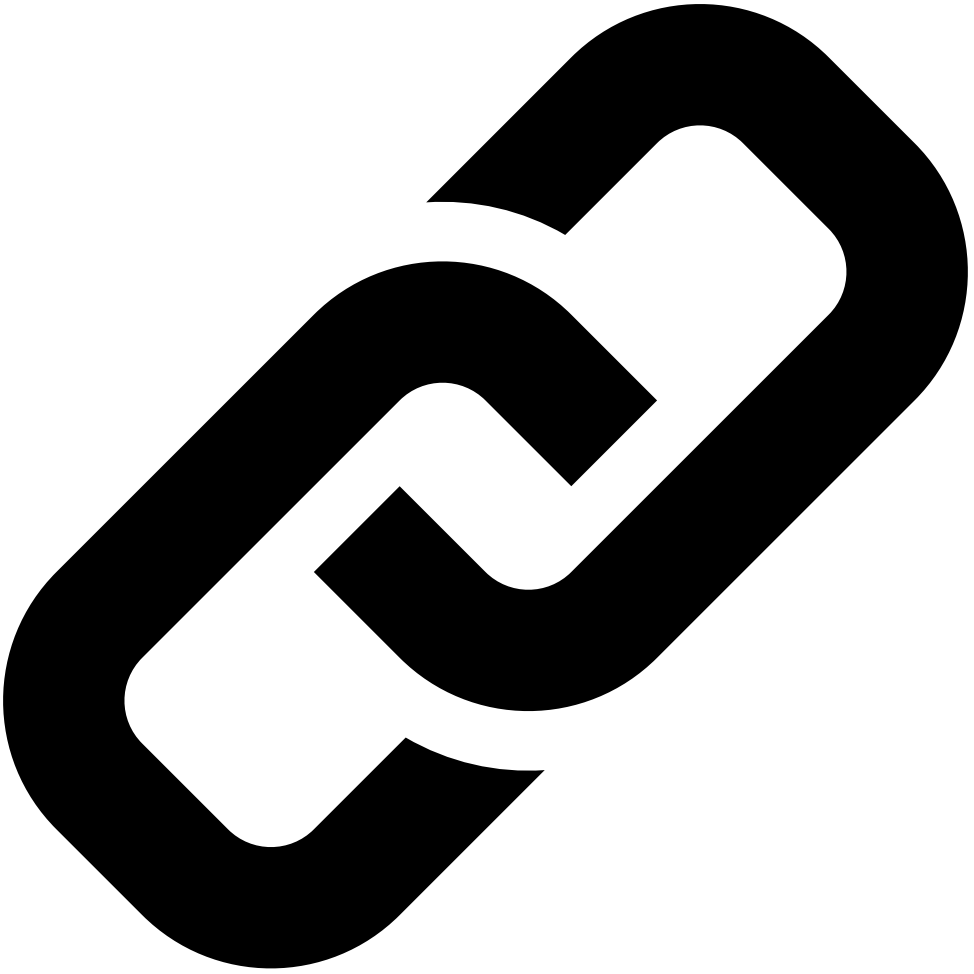
This is because now the *malloc* function in our custom-built library takes precedence over the standard *malloc* function. In this interposer example, the *malloc* function printed the size and call-count at certain memory allocation calls.

It's possible to specify multiple libraries in the *LD_PRELOAD* variable. For this, we can provide a list of libraries, separated using a colon or space:

```
 $ LD_PRELOAD="/data/preload/lib/malloc_interpose.so:/data/preload/lib/free_interpose.so" ls -lh
malloc(20000)  call number: 223
malloc(32816)  call number: 226
total 2,8G
free((nil))  call number: 174
free((nil))  call number: 175
free((nil))  call number: 178
-rw-rw-r-- 1 baeldung_user baeldung_user 3,6K Jul  4 20:16 BVF_Density.ipynb
-rwxrwxr-x 1 baeldung_user baeldung_user 2,8G Jul  5 15:59 cuda_11.0.1_450.36.06_linux.run
drwxrwxr-x 2 baeldung_user baeldung_user   25 Jul  8 16:12 h5store
drwxrwxr-x 2 baeldung_user baeldung_user   30 Jul  5 18:47 pandas_processing
drwxrwxr-x 4 baeldung_user baeldung_user  142 Jul 19 15:59 preload
free((nil))  call number: 180
```

Now, we've added another imaginary interposer library to the *LD_PRELOAD* variable that reports on certain calls to the system function *free*, where a *NULL* pointer is passed.

4. When Is This Helpful?



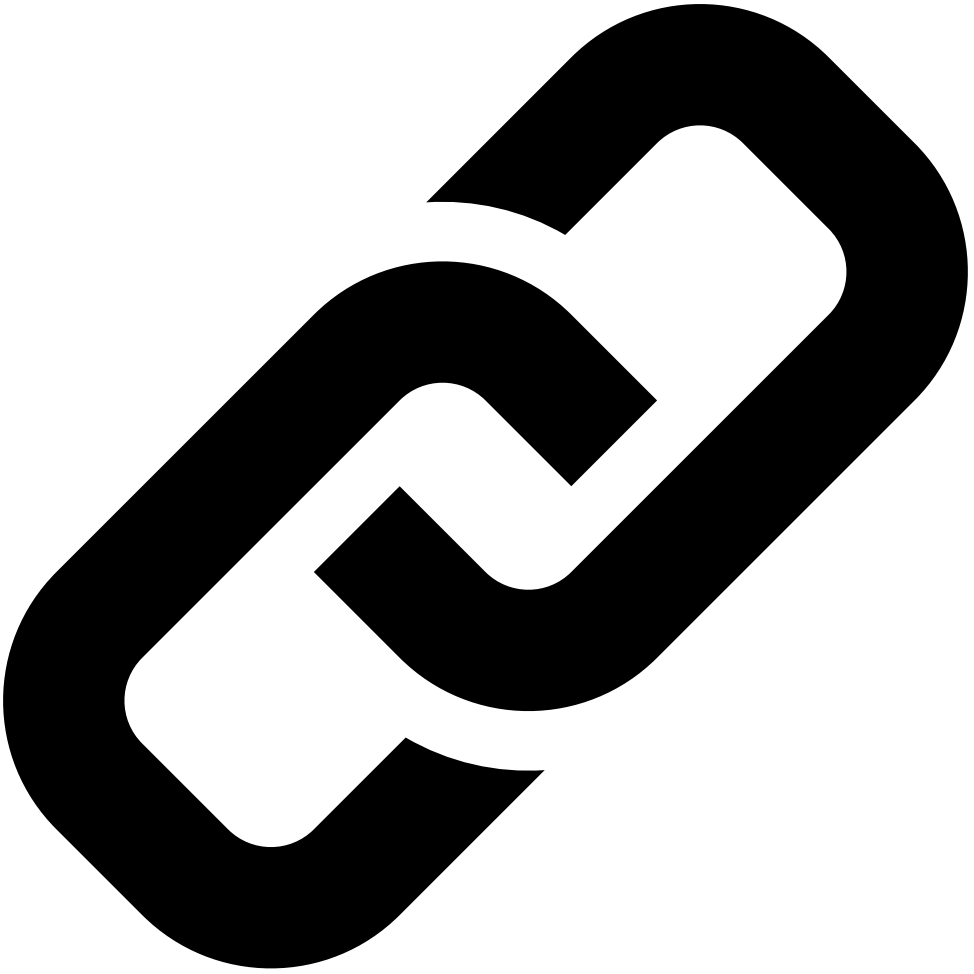
The *LD_PRELOAD* trick can be helpful in some situations.

For example, consider the case **when two libraries export the same symbol and our program links with the wrong one**. In this case, the library with the correct symbol can be preloaded by using *LD_PRELOAD*. Doing this will result in the resolution of the correct symbol.

Another use case is **when an optimized or custom implementation of a library function should be preferred**. We can preload this optimized or custom implementation without changing the original library. Moreover, we can also replace the whole library by providing a different version.

Additionally, **various profiling and monitoring tools widely use *LD_PRELOAD* for instrumenting code**. For instance, a performance profiling application will interpose key system functions. This enables the profiler to collect relevant data from the user application.

5. Alternative to *LD_PRELOAD*



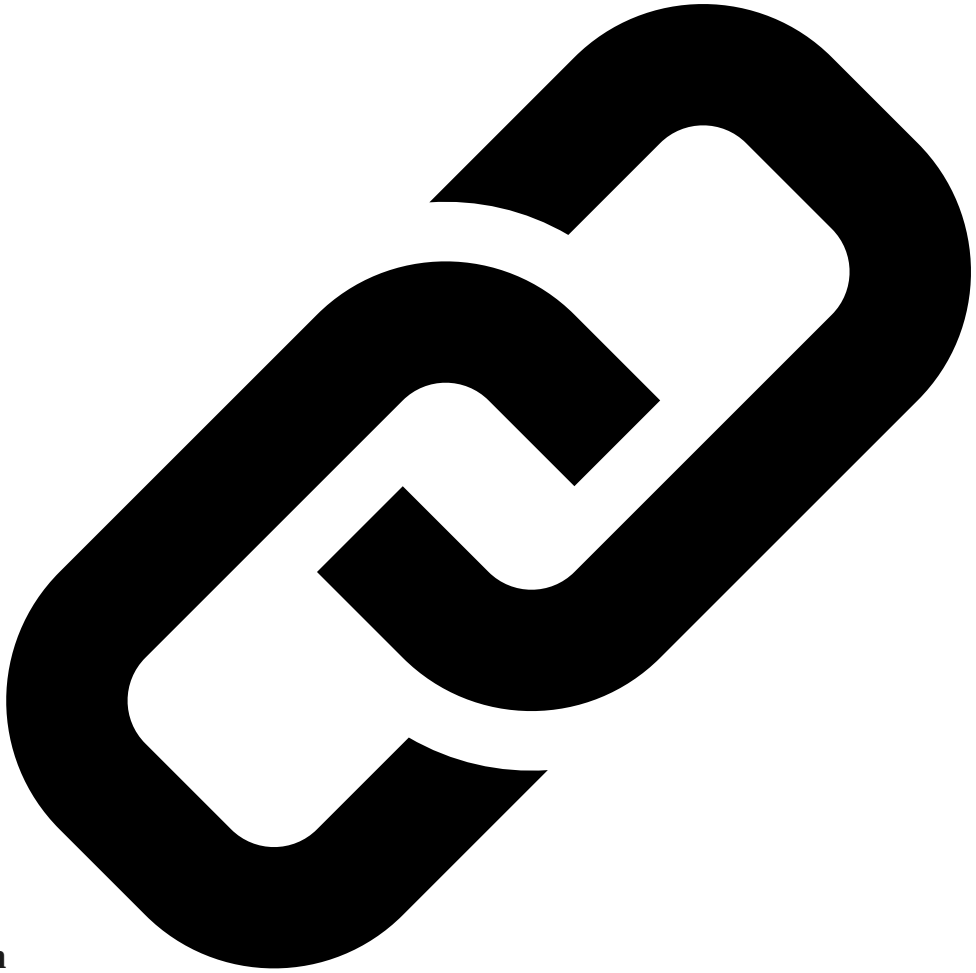
Besides the *LD_PRELOAD* trick, there is an alternative **using the */etc/ld.so.preload* file**. However, **this is a system-wide setting**. In most systems, this file doesn't even exist by default, and a system administrator has to create it.

Let's now create this file and select an older version of *zlib* for preloading, and then we'll test the *pigz* program again:

```
 $ pigz -vV
pigz 2.4
zlib 1.2.11

$ sudo -i
# echo "/data/preload/lib/libz.so.1.2.8" > /etc/ld.so.preload
# exit
logout

$ pigz -vV
pigz 2.4
zlib 1.2.8
```



6. Conclusion

In this article, we discussed various ways to use the *LD_PRELOAD* trick.

It's an advanced technique that can be helpful in certain scenarios. For example, developers can quickly debug and test libraries or some functions within the library.

It's also a well-known way of code instrumentation in profiling and monitoring tools using interposing techniques.

Moreover, system administrators can also use this to control the execution environment for specific users, or for all users.
