

Name: Archit Benipal

Subject: Software Supply Chain Security (CY 653)

Assignment 10: Execution by utilizing the LD LIBRARY PATH environment variable

What is ldconfig

ldconfig is a command used in Unix and Unix-like operating systems such as Linux. It is used to create, update, and manage the symbolic links and cache (typically /etc/ld.so.cache) for the shared libraries. When a program is started, the dynamic linker is invoked to load any dynamic libraries that the program needs. The dynamic linker uses the cache file to find these libraries efficiently. When libraries are added, removed, or relocated, the cache must be updated, typically with ldconfig.

If ldconfig is not run when necessary, programs starting may not find the libraries they require and can fail to start or operate incorrectly.

Here's how it might be used:

After installing new libraries in a directory tracked by the linker (such as /usr/local/lib), ldconfig can be run to update the cache.

If a directory that contains shared libraries is added, ldconfig can be run with the -n option and the directory path to add these to the cache. For example, ldconfig -n /new/path/to/lib.

Uses of ldconfig:

- **Update the cache:** When new libraries are added, removed, or relocated, the cache, typically /etc/ld.so.cache, is updated with the **ldconfig** command so that the dynamic linker can find them efficiently.
- **Adding new library paths:** If you add a new directory that contains shared libraries, you can run **ldconfig** with the -n option followed by the directory path to add these to the cache. For example, **ldconfig -n /path/to/new/library**.
- **Checking library info:** You can use **ldconfig -p** to print the current shared library cache, which can be helpful for debugging or verifying that libraries have been installed correctly.
- **Updating library links:** **ldconfig** also checks that symbolic links are properly set up for libraries. For instance, a library **libfoo.so.1.2.3** might have symbolic links like **libfoo.so.1** and **libfoo.so** pointing to it. If these are missing, **ldconfig** can create them.
- **Reducing start-up time for many programs:** By maintaining an updated cache file for libraries, **ldconfig** helps the dynamic linker to load these libraries more efficiently when a program is started, which can reduce the start-up time for programs that depend on many libraries.

Step-by-step process: Method 1: Hard link

For this demo I will use the original files and folders which I used for assignment_4 (dynamic_linking)

Step 1: Created lib1 and copied all the contents of lib1 to lib2 and remove the calc(executable) and calc.o



Step 2: Removed the original .o and .so files and created them for lib2



Step 3: Created a link between lib2 and /usr/lib



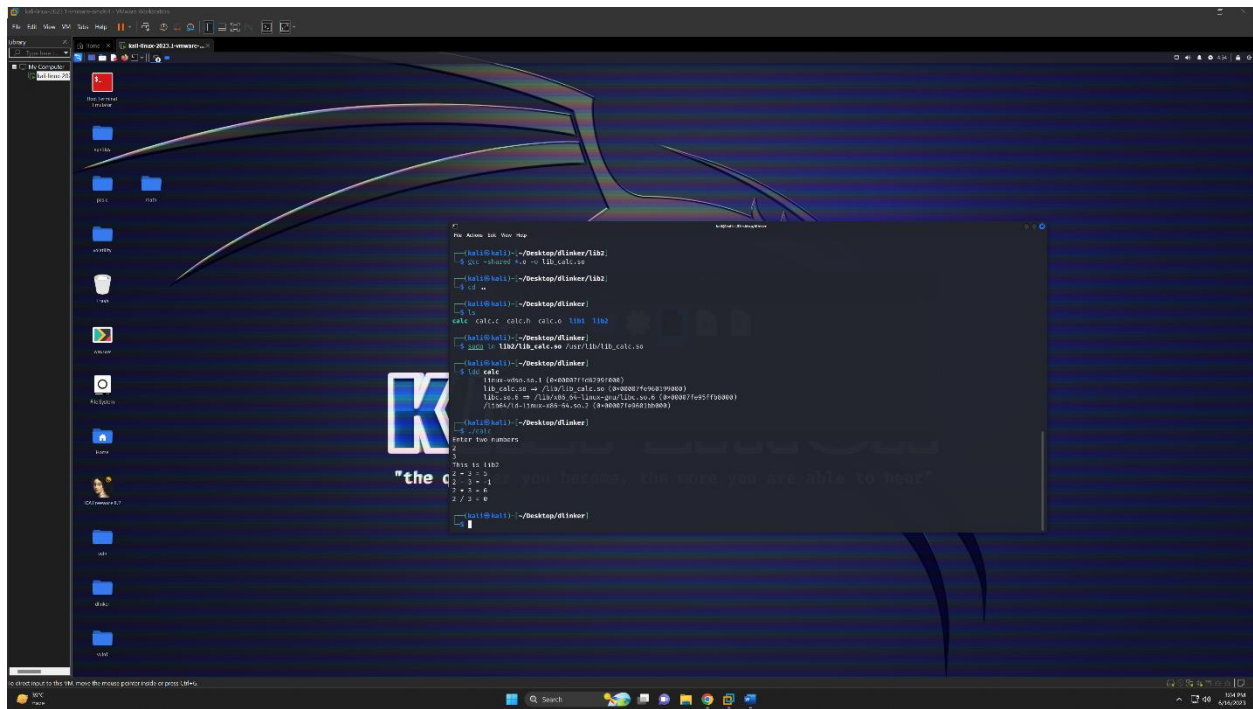
Breakdown of the command used:

- **sudo:** This runs the following command with root privileges. Typically need these privileges to make changes in the **/usr/lib** directory.
- **ln:** This is the command that creates links.
- **lib2/lib_calc.so:** This is the source file — the file you want to link to.
- **/usr/lib/lib_calc.so:** This is the link name — the location where you want to create the link.

After running this command, when programs look for **lib_calc.so** in the **/usr/lib** directory, they'll be redirected to the actual library at **lib2/lib_calc.so**. This is helpful if a program that's looking for a library in a specific location, but we don't want to (or can't) move the actual library file to that location.

Note: We need to run '**sudo ldconfig**' to update the library cache after creating the link. This will ensure the dynamic linker knows about the new library.

Step 4: Executed the calc (executable)



Step 5: Removed the previous link and linked it to lib1



Method 2: Easy and fast way to switch library is to use LD_PATH_ENVIRONMENT variable

How is this faster than linking?

The method of setting the LD_LIBRARY_PATH environment variable to include a custom directory for shared libraries is "faster" than creating symbolic links in terms of the time it takes to make the change. Specifically, by setting LD_LIBRARY_PATH we tell the dynamic linker to also search in the specified directory for required shared libraries when we launch a program. This change can be made immediately and only requires modifying an environment variable, which can be done quickly.

On the other hand, creating symbolic links to each individual library file (with the **ln -s** command) and then updating the library cache (with **ldconfig**) involves more steps and thus takes more time.

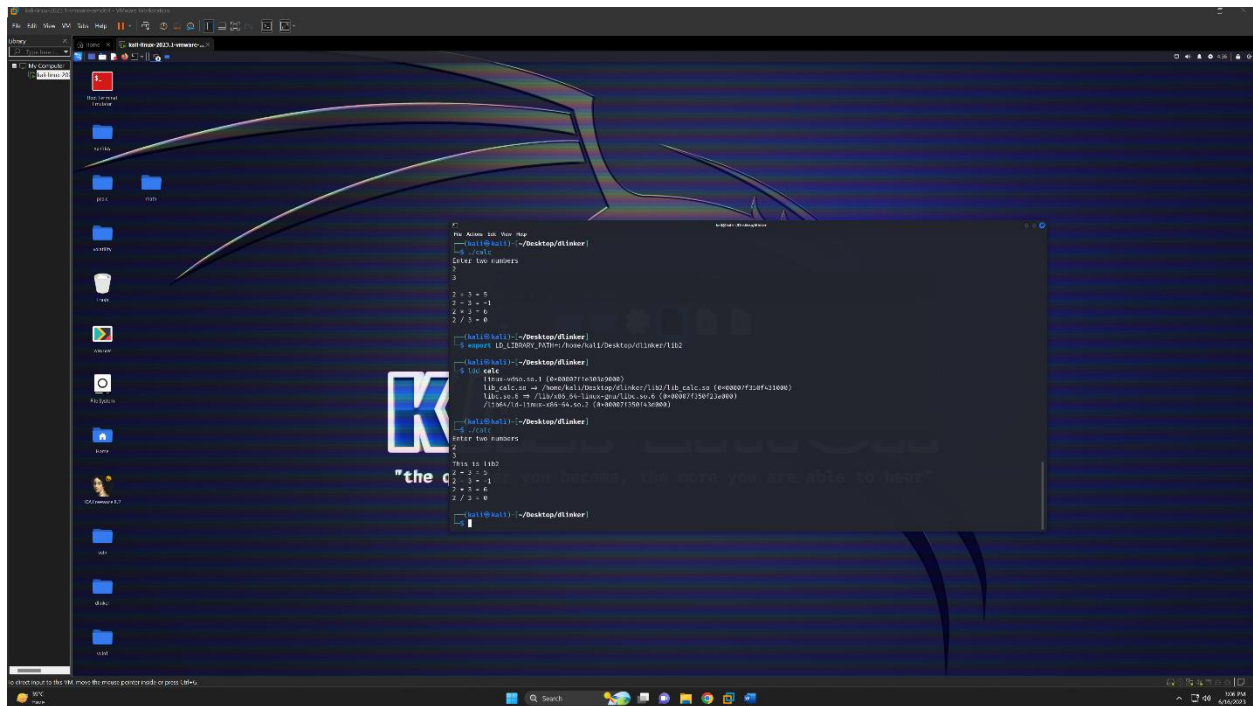
Cons of using this method:

- The **LD_LIBRARY_PATH** setting is only effective in the shell session where we set it (and its child sessions). If we need the change to be effective system-wide or persist across reboots, we will need to modify a system-wide configuration file or shell profile script, which might not be faster or easier than creating symbolic links.
- The dynamic linker checks directories in the order listed in **LD_LIBRARY_PATH**. If there are many directories or the directories contain many files, this could slow down the startup of programs. By contrast, the **ldconfig** cache is designed for fast lookups.
- Using **LD_LIBRARY_PATH** can have unintended side effects if a program needs different versions of the same library, or if a malicious user can add their own libraries to the directories in **LD_LIBRARY_PATH**.

Therefore, while setting LD_LIBRARY_PATH can be a faster method in terms of immediate changes, the traditional method of using ldconfig to manage shared libraries can be more robust, efficient, and secure for managing shared libraries in the long term.

Step-by-Step process:

Step 1: Set the LD_LIBRARY_PATH variable to /lib2 folder

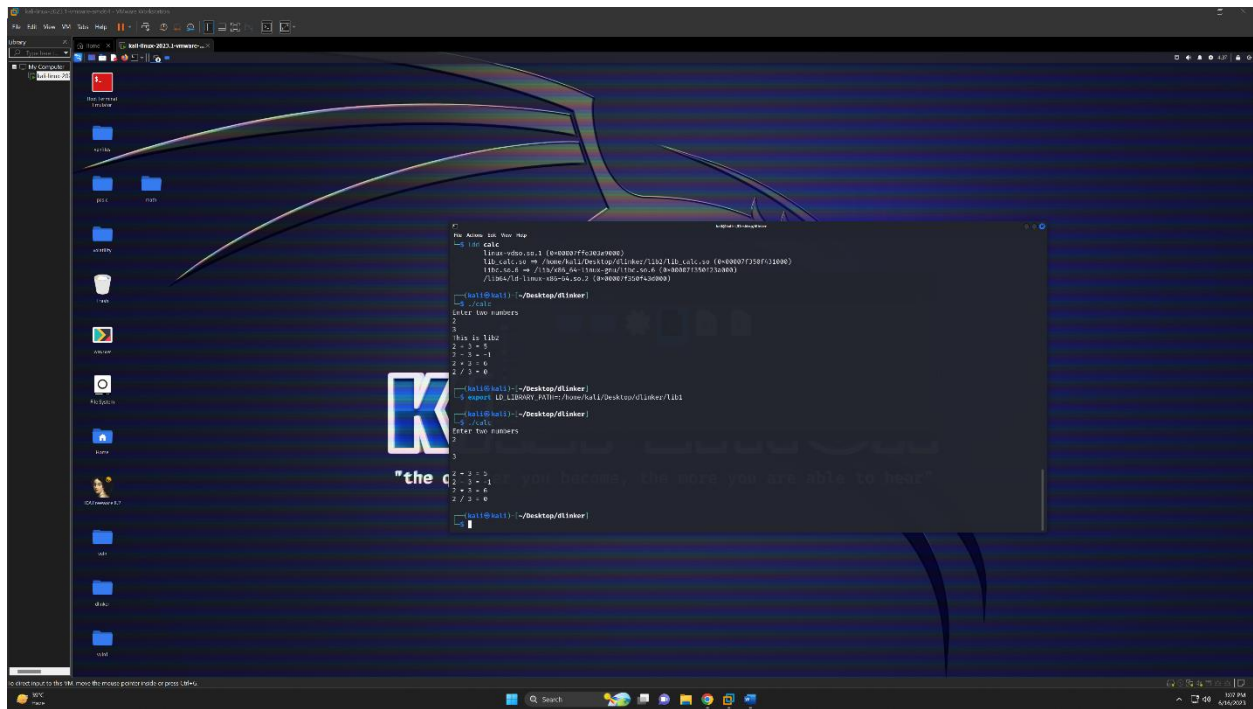


Breakdown of the command used:

- **export:** This command makes the environment variable available to child processes of the current shell session.
- **LD_LIBRARY_PATH:** This is the environment variable that's being set.
- **:/home/kali/Desktop/dlinker/lib2:** This is the value that's being assigned to **LD_LIBRARY_PATH**. The colon at the beginning of the value adds this directory to any existing directories in **LD_LIBRARY_PATH** (rather than replacing them). If **LD_LIBRARY_PATH** is currently unset or empty, this will be equivalent to setting it to just **/home/kali/Desktop/dlinker/lib2**.

Note: When modifying **LD_LIBRARY_PATH**, as it can affect the operation of system programs and can be a security risk if misused. It's generally recommended to use other methods of managing libraries, like adding them to the standard library directories and running **ldconfig**, when possible.

Step 2: Switched the library to lib1



Step 3: Viewed the contents of the current shared library cache with command 'ldconfig -p'

