Function interposition

You are given a Linux program called expire that runs on the Rutgers iLab Linux systems. Download the file here and unzip it.

Imagine that it is the first part of a program that you received for evaluation and it has an expiration time coded into it. In this program, it exits (expires) if the date is after June 15, 2018. It will also refuse to run with any date earlier than Jan 1 2016. However, you wish to continue using this program and you want to defeat its check for the time.

Your assignment is to replace the program's call to a system library to get the time of day with one of your own – an alternate version that will return a suitable date for the program's time check. This is called function interposition. After the time-based check for validity is done, you want the program to use the actual time, so it will return the correct time of day.

Groups

This is an individual assignment. You are expected to do this on your own and submit your own version of the code and output.

Environment

Your submissions will be tested on Rutgers iLab Linux systems. You can most likely develop this on any other Linux system but you are responsible for making sure that it will work on the iLab systems. You cannot do this assignment on macOS; the BSD functions it uses for dynamic linking are somewhat different.

What you need to do

This program happens to use the standard C library (glibc) time function to get the system time. This returns the number of seconds since the Linux Epoch (the start of January 1, 1970 UTC).

Your assignment is to create an alternate version of the time() C library function that will return a value within a time window that will pass the program's validation check.

You will use Linux's `LD_PRELOAD` mechanism. This is a shell environment variable that forces the listed shared libraries to be loaded for programs you run. The dynamic linker will search these libraries first when it resolves symbols in the program. This allows you to take control and replace standard library functions with your own versions.

You need to write just one function in one file. Let's call the file `newtime.c`.

The function will be your implementation of time. You'll give it the same name and it should take the same parameters and return the same data type as the original version. The proram validates the time only the first time it calls time. After that, you want calls to time to return the correct system time. Unfortunately, your time function will continue to be called by the program so you will need to have your function dynamically link the original time function and pass successive calls to it.

You will then compile this into a shared library called `newtime.so`.

You can then preload this shared library by setting the environment variable:

```
export LD_PRELOAD=$PWD/newtime.so
```

and then run the program normally:

```
./expire
```

If your implementation is correct, you will get a message stating:

```
PASSED! You reset the time successfully!
```

If not, you should get a message stating what part of your implementation failed.

Hints

The program should be quite short: approximately 15 lines of code. Test an initial version that does not link in the original time function just to make sure you have that first part working before you move on.

Some of you will no doubt finish this assignment in under an hour. Some, however, may not have any experience with the various Linux time functions and may struggle a bit finguring out how to convert a date. Allow yourself sufficient time and realize that we have an exam coming up.

References

There are many tutorials on function interposition on the web. You'll want to follow the instructions for dynamically linking original functions as well as the proper flags to use to compile the shared library (`newtime.so`).

Some reference you may find useful are:

- man page for the time function
- manual page for ldsym
- NetSPI Blog, Function Hooking Part I
- CatonMat, A Simple LD_PRELOAD Tutorial
- CatonMat, A Simple LD_PRELOAD Tutorial, Part 2

What to submit

When you have completed your assignment, you will need to submit only two files via sakai:

1. `newtime.c`, which contains your implementation of time. Make sure your name, netID, and RUID are in the comments.
2. `output.txt`, which contains the exact output of everything printed by running the expire program. For example: ./expire >output.txt

Validate that the contents are correct before submitting. We will use a script to generate and test your programs, and you will be penalized for any deviation. Do not submit anything else.