CS3204 Operating Systems - Fall 2000
Instructor: Dr. Craig A. Struble
Shared Libraries

Assigned: Thursday, Sep. 7 **Due:** Midnight, Friday, Sep. 22

1 Introduction

Years ago, linkers used to generate binaries with a **copy** of standard library object code. This contributed not only to wasteful disk space usage, but memory usage as well. Having copies of standard library code included in each executable was easier for operating system designers and implementors, however. An executable was loaded into memory and any machine code needed for execution was readily available.

The *Win32* platform and modern versions of *Unix* now support reusable libraries of code. Only one copy of standard libraries exist on disk and in memory. When a library is needed, the operating system loads it into memory once, and any new executables use the already loaded library. The *Win32* name for these reusable libraries is *Dynamically Linked Library (DLL)* and the *Unix* name is *shared library* or *shared object*.

The goal of this assignment is to introduce the interface to the system loader under Unix. With this interface, executable code, in the form of shared objects, can be loaded and executed *at runtime*. Furthermore, the code can be unloaded (i.e., removed from memory) by the application. This interface is primarily used to implement plug-in systems such as ones used by Java, Netscape, and the Linux/FreeBSD kernels.

2 Specification

You are to implement an interactive program for loading, unloading, and executing code in shared object files. In addition, you must implement two different shared object files to demonstrate that all of the features of your interactive program function properly.

Your interactive program must provide the following commands:

Command	Function
`load <filename>`	Loads the shared object file named `<filename>` into memory. You may assume filename will not contain any spaces or other non-printable characters.
`unload <filename>`	Removes the shared object file named `<filename>`.
`list`	Lists the shared object files loaded and displays the memory added where they are located.
`call <filename> <function>`	Call the C function `<function>` contained in the shared object named `<filename>`. The function will have the prototype `void <function>()`.
`help`	List the available commands with a brief description of each command. The command syntax should also be included.
`quit`	Unload any loaded shared objects and exit your program.
`exit`	Unload any loaded shared objects and exit your program.

Your program must also display a brief introductory message and provide a command line prompt for the user. Upon success, the commands `load`, `unload`, and `call` should print out a message including the memory location where the shared object was loaded, verification that the shared object was unloaded, or the addresses of the function executed and the shared object containing the function respectively. Upon failure of any command, your program should indicate that a failure occurred, and a description of what caused the error. Be sure to indicate if a user types in an improper command.

Note, it is perfectly valid to load a shared object file more than once. Read the manual pages referenced in Section 3 for details.

When implementing shared objects, you must have the object print out a message when it is loaded into memory for the first time and a message when the object is unloaded from memory. These messages will not correspond to each execution of `load` and `unload` in your interactive program. Each shared object must have at least two functions to call. The functions must indicate, by printed messages, that they have been successfully called.

3 References

To implement your project, you will need to use the `dlopen(...)`, `dlerror(...)`, `dlsym(...)`, and `dlclose(...)` functions. Executing `man` on any one of those functions provides you with the reference material you need to implement your interactive program. Furthermore, details elaborating on special features of shared objects are included in the manual page. You will need to link against the `dl` library for these functions to be available (i.e., the `-ldl` flag is needed during compilation).

In addition, you should read the manual pages for `ld` and `ld.so` for more information regarding the linker and loader. To construct your shared object, you will need to first compile it so that the loader can place it into an arbitrary location in memory. This is accomplished by using *position independent code (PIC)* during the compilation process. The correct compilation command is

```
cc -fpic -c sharedobj.c
```

which will generate a `sharedobj.o` object file, appropriately compiled. To generate a shared object under Linux once the object file has been created, you must use the linker. The correct link command is

```
ld -shared sharedobj.o -o sharedobj.so
```

which will generate `sharedobj.so`, the shared object file.

4 Discussion

As part of the documentation requirement, you must provide a brief discussion for the following scenario. *The interface provided by the dynamic loader does not include a method for gathering the function names available in a shared object. Using only the current dynamic loader interface, design a specification and briefly discuss a possible avenue for implementation to list the available functions in a shared object when the shared object is loaded into memory.*

Keep your answer to one or two paragraphs. You do not need to implement your specification, just provide a general idea of what is necessary and how it might work. Simply providing an implementation is not acceptable.

5 Submission

We will use the Curator, `http://ei.cs.vt.edu/~eags/Curator.html` to collect program submissions. The URL for submission is `http://spasm.cs.vt.edu:8080/curator/`. Only the servlet interface to the Curator is supported. No grading will be done by the Curator.

You are to submit a single tarred (`man tar`) and gzipped (`man gzip`) archive containing

- A `README` text file describing the program, describing the contents of the archive, providing building instructions (including the platform you used for development), a user's guide (including how to start the program), and examples of usage with your shared objects;
- A `DISCUSSION` text file containing your answer to the discussion question;
- The source code for your program and shared objects;
- A script named `build` or a suitable `Makefile` for building your program and shared objects.

Be sure to include your name in all files submitted. **DO NOT** include executables or object files of any type in the archive.

6 Programming Environment

As stated in the syllabus, you may use either FreeBSD or Linux and ANSI C/C++ to implement this project. The linker command to create shared objects under FreeBSD may be slightly different, but FreeBSD provides the same interface to the system loader. **All data structures used in your program must be student implemented. Using the standard template library (STL) or other third party libraries for data structure implementations is strictly prohibited.**