A Proposed Implementation of Shared Libraries on DOS Platforms
by Mercury Thirteen, May 2021

1. Introduction

Shared libraries provide many benefits to an operating system, including a reduction of application size and complexity, and, by extension, increased security and stability. As such, some form of the shared library concept is implemented by all modern operating systems, and even some classic ones - DOS-style systems, however, never fell under this umbrella. Remedying this would ease the burden of coding on the DOS platform, making it a more attractive target for developers and aid in the creation of a new generation of applications.

Imparting this feature to a DOS system in the most succinct manner would involve implementing support for the libraries themselves, likely in ELF format, at the system level. However, there are a some obstacles to consider with such an approach:

- The majority of DOS-style systems are no longer under development, making modifications to them impossible.
- Getting support integrated into FreeDOS, the DOS-style system which is most actively developed, is unlikely to garner widespread support and/or priority within the developer community.
- Adding support directly to the OS itself would add unnecessary bloat for those who do not need or want the feature.

Considering the above points, the best alternative method of implementing a shared library mechanism in a DOS system would by through using a terminate-and-stay-resident program (TSR) to provide applications a "bridge" of sorts to access the libraries. As this is a feature never formally included in DOS systems – and there is therefore no official standard to follow - this document outlines one possible method of implementation to open the floor for discussion.

2. Details of Components and Operation

Libraries

For simplicity, libraries would be installed simply by adding them to a specified directory; `/LIB` for example. The format of the library files could be one of two possibilities; ELF-format binaries, or simple ZIP archives containing files, each of which are a single function offered by the library. The latter is likely the path to be taken first, as the ZIP format is relatively simple, allows for compression, and there are already a number of tools available on DOS platforms for working with ZIP files.

Manager Application

This application would be used only for managing the library files themselves and has no part in the process of calling library code at run time. Its primary purpose is to be invoked manually by the user after any files are added, deleted, or replaced in the directory containing the libraries. On launch, it will scan all library files in the specified directory, get the names of all functions made available in each library, and add this information to a single "database" file which would be kept in the same directory. In the event of a conflict – e.g. the same function being made available from two different library files – this application will be able to

alert the user and, at the user's choosing, either delete one of the files or keep the file but simply disregard the older version of the specified function. In this way, its operation is akin to Apple's Extension Manager in the classic MacOS.

Database File Format

The database file is a simple text file containing an alphabetical listing of the names of all functions made available across all library files, along with the name of the library file containing the function. The entries in the file will be equally spaced to ease locating the start of all functions beginning with a specific letter.

Main TSR

The TSR would be executed at boot and would stay memory resident to provide dynamic loading and unloading of functions listed in the database file compiled by the manager application. The following basic functions would be made available via an interrupt number determined by the AMIS specification:

- A basic "installation check" which returns data about the TSR.
- Querying the version number of a specified library.
- Querying if a specified function is available.
- Loading a function.
- Unloading a function.

The TSR would also support being unloaded by a parameter passed on the command line, most likely `/U`.

3. Possible Use Cases

Image editors would particularly benefit from a shared library architecture, as they would be able to load and unload individual tools as needed - and even have entirely new tools added at a later date - all without recompiling the applications themselves. Extending this principle to files, applications would also be able to open and save newer formats as they emerge by simply modifying the underlying library file which handles these operations instead of changing each individual application to update its compatibility. This flexibility could be extended to word processors, encryption applications, compression, and any other common features which may be needed by a variety of applications.

Additionally, this would make possible GUI applications with consistent, unified themes. Libraries could exist which would contain all possible elements of a given user interface, and developers would be free to use that UI at will instead of reinventing this wheel over and over again on a per-application basis. Standardized dialog boxes would be possible for everyday operations like file selection, option confirmation, and more.

Indeed, even the drawing of UI elements to the screen could be relegated to yet other shared libraries designed to handle various video modes and image buffers. Typical image operations like rotation, scaling, and shifting between various bit depths would now easily be accessible to any application which needed the ability, all without the developer being forced to write complex code from scratch.

4. Conclusion

This document is in RFQ status, and is provided as an outline only, to allow for discussion and additional changes to be proposed prior to any code being written. Constructive commentary is welcomed.

5. Revision History

2023-04-28	Corrected some grammar. Updated Section 2 to indicate that the Main TSR would use the AMIS specification for determining its interrupt number.
2021-05-25	Initial release.