Full credit is given to the above companies including the OS that this PDF file was generated!

Red Hat Enterprise Linux Release 9.2 Manual Pages on 'dlopen.3p' command

$ man dlopen.3p

DLOPEN(3P) POSIX Programmer's Manual DLOPEN(3P)

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

dlopen ? open a symbol table handle

SYNOPSIS

#include <dlfcn.h>

void *dlopen(const char *file, int mode);

DESCRIPTION

The dlopen() function shall make the symbols (function identifiers and data object identifiers) in the executable object file specified by file available to the calling program.

The class of executable object files eligible for this operation and the manner of their construction are implementation-defined, though typically such files are shared libraries or programs.

Implementations may permit the construction of embedded dependencies in executable object files. In such cases, a dlopen() operation shall load those dependencies in addition to the executable object file specified by file. Implementations may also impose specific constraints on the construction of programs that can employ dlopen() and its related ser?

vices.

A successful dlopen() shall return a symbol table handle which the caller may use on subsequent calls to dlsym() and dlclose().

The value of this symbol table handle should not be interpreted in any way by the caller.

The file argument is used to construct a pathname to the executable ob? ject file. If file contains a <slash> character, the file argument is used as the pathname for the file. Otherwise, file is used in an imple? mentation-defined manner to yield a pathname.

If file is a null pointer, dlopen() shall return a global symbol table handle for the currently running process image. This symbol table han? dle shall provide access to the symbols from an ordered set of exe? cutable object files consisting of the original program image file, any executable object files loaded at program start-up as specified by that process file (for example, shared libraries), and the set of executable object files loaded using dlopen() operations with the RTLD_GLOBAL flag. As the latter set of executable object files can change during execution, the set of symbols made available by this symbol table han? dle can also change dynamically.

Only a single copy of an executable object file shall be brought into the address space, even if dlopen() is invoked multiple times in refer? ence to the executable object file, and even if different pathnames are used to reference the executable object file.

The mode parameter describes how dlopen() shall operate upon file with respect to the processing of relocations and the scope of visibility of the symbols provided within file. When an executable object file is brought into the address space of a process, it may contain references to symbols whose addresses are not known until the executable object file is loaded.

These references shall be relocated before the symbols can be accessed.

The mode parameter governs when these relocations take place and may have the following values:

RTLD_LAZY Relocations shall be performed at an implementation-defined

time, ranging from the time of the dlopen() call until the first reference to a given symbol occurs. Specifying RTLD_LAZY should improve performance on implementations supporting dynamic symbol binding since a process might not reference all of the symbols in an executable object file. And, for systems supporting dynamic symbol resolution for normal process execution, this behavior mimics the normal handling of process execution.

RTLD_NOW All necessary relocations shall be performed when the exe? cutable object file is first loaded. This may waste some processing if relocations are performed for symbols that are never referenced. This behavior may be useful for ap? plications that need to know that all symbols referenced during execution will be available before dlopen() returns.

Any executable object file loaded by dlopen() that requires relocations against global symbols can reference the symbols in the original process image file, any executable object files loaded at program start-up, from the initial process image itself, from any other exe? cutable object file included in the same dlopen() invocation, and any executable object files that were loaded in any dlopen() invocation and which specified the RTLD_GLOBAL flag. To determine the scope of visi? bility for the symbols loaded with a dlopen() invocation, the mode pa? rameter should be a bitwise-inclusive OR with one of the following val? ues:

RTLD_GLOBAL The executable object file's symbols shall be made avail? able for relocation processing of any other executable ob? ject file. In addition, symbol lookup using dlopen(NULL,mode) and an associated dlsym() allows exe? cutable object files loaded with this mode to be searched.

RTLD_LOCAL The executable object file's symbols shall not be made available for relocation processing of any other executable object file.

If neither RTLD_GLOBAL nor RTLD_LOCAL is specified, the default behav?

ior is unspecified.

If an executable object file is specified in multiple dlopen() invoca?

tions, mode is interpreted at each invocation.

If RTLD_NOW has been specified, all relocations shall have been com?

pleted rendering further RTLD_NOW operations redundant and any further

RTLD_LAZY operations irrelevant.

If RTLD_GLOBAL has been specified, the executable object file shall

maintain the RTLD_GLOBAL status regardless of any previous or future

specification of RTLD_LOCAL, as long as the executable object file re?

mains in the address space (see dlclose()).

Symbols introduced into the process image through calls to dlopen() may

be used in relocation activities. Symbols so introduced may duplicate

symbols already defined by the program or previous dlopen() operations.

To resolve the ambiguities such a situation might present, the resolu?

tion of a symbol reference to symbol definition is based on a symbol

resolution order. Two such resolution orders are defined: load order

and dependency order. Load order establishes an ordering among symbol

definitions, such that the first definition loaded (including defini?

tions from the process image file and any dependent executable object

files loaded with it) has priority over executable object files added

later (by dlopen()). Load ordering is used in relocation processing.

Dependency ordering uses a breadth-first order starting with a given

executable object file, then all of its dependencies, then any depen?

dents of those, iterating until all dependencies are satisfied. With

the exception of the global symbol table handle obtained via a dlopen()

operation with a null pointer as the file argument, dependency ordering

is used by the dlsym() function. Load ordering is used in dlsym() oper?

ations upon the global symbol table handle.

When an executable object file is first made accessible via dlopen(),

it and its dependent executable object files are added in dependency

order. Once all the executable object files are added, relocations are

performed using load order. Note that if an executable object file or

its dependencies had been previously loaded, the load and dependency

orders may yield different resolutions.

The symbols introduced by dlopen() operations and available through dl? sym() are at a minimum those which are exported as identifiers of global scope by the executable object file. Typically, such identifiers shall be those that were specified in (for example) C source code as having extern linkage. The precise manner in which an implementation constructs the set of exported symbols for an executable object file is implementation-defined.

RETURN VALUE

Upon successful completion, dlopen() shall return a symbol table han? dle. If file cannot be found, cannot be opened for reading, is not of an appropriate executable object file format for processing by dlopen(), or if an error occurs during the process of loading file or relocating its symbolic references, dlopen() shall return a null pointer. More detailed diagnostic information shall be available through dlerror().

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Refer to dlsym().

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

dlclose(), dlerror(), dlsym()

The Base Definitions volume of POSIX.1?2017, <dlfcn.h>

COPYRIGHT

table Operating System Interface (POSIX), The Open Group Base Specifi? cations Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.ker? nel.org/doc/man-pages/reporting_bugs.html .