## NAME

**dlopen**, **fdlopen**, **dlsym**, **dlvsym**, **dlfunc**, **dlerror**, **dlclose** - programmatic interface to the dynamic linker

## LIBRARY

Standard C Library (libc, -lc)

## SYNOPSIS

**#include <dlfcn.h>**

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

*void **
**fdlopen**(*int fd, int mode*);

*void **
**dlsym**(*void * restrict handle, const char * restrict symbol*);

*void **
**dlvsym**(*void * restrict handle, const char * restrict symbol, const char * restrict version*);

*dlfunc_t*
**dlfunc**(*void * restrict handle, const char * restrict symbol*);

*char **
**dlerror**(*void*);

*int*
**dlclose**(*void *handle*);

## DESCRIPTION

These functions provide a simple programmatic interface to the services of the dynamic linker. Operations are provided to add new shared objects to a program's address space, to obtain the address bindings of symbols defined by such objects, and to remove such objects when their use is no longer required.

The **dlopen**() function provides access to the shared object in *path*, returning a descriptor that can be used for later references to the object in calls to **dlsym**(), **dlvsym**() and **dlclose**(). If *path* was not in the address space prior to the call to **dlopen**(), it is placed in the address space. When an object is first loaded into the address space in this way, its function **_init**(), if any, is called by the dynamic linker. If

*path* has already been placed in the address space in a previous call to **dlopen**(), it is not added a second time, although a reference count of **dlopen**() operations on *path* is maintained. A null pointer supplied for *path* is interpreted as a reference to the main executable of the process. The *mode* argument controls the way in which external function references from the loaded object are bound to their referents. It must contain one of the following values, possibly ORed with additional flags which will be described subsequently:

RTLD_LAZY Each external function reference is resolved when the function is first called.

RTLD_NOW All external function references are bound immediately by **dlopen**().

RTLD_LAZY is normally preferred, for reasons of efficiency. However, RTLD_NOW is useful to ensure that any undefined symbols are discovered during the call to **dlopen**().

One of the following flags may be ORed into the *mode* argument:

RTLD_GLOBAL Symbols from this shared object and its directed acyclic graph (DAG) of needed objects will be available for resolving undefined references from all other shared objects.

RTLD_LOCAL Symbols in this shared object and its DAG of needed objects will be available for resolving undefined references only from other objects in the same DAG. This is the default, but it may be specified explicitly with this flag.

RTLD_TRACE When set, causes dynamic linker to exit after loading all objects needed by this shared object and printing a summary which includes the absolute pathnames of all objects, to standard output. With this flag **dlopen**() will return to the caller only in the case of error.

RTLD_NODELETE Prevents unload of the loaded object on **dlclose**(). The same behaviour may be requested by **-z nodelete** option of the static linker ld(1).

RTLD_NOLOAD Only return valid handle for the object if it is already loaded in the process address space, otherwise NULL is returned. Other mode flags may be specified, which will be applied for promotion for the found object.

RTLD_DEEPBIND Symbols from the loaded library are put before global symbols when resolving symbolic references originated from the library.

If **dlopen**() fails, it returns a null pointer, and sets an error condition which may be interrogated with

**dlerror**().

The **fdlopen**() function is similar to **dlopen**(), but it takes the file descriptor argument *fd*, which is used for the file operations needed to load an object into the address space. The file descriptor *fd* is not closed by the function regardless a result of execution, but a duplicate of the file descriptor is. This may be important if a lockf(3) lock is held on the passed descriptor. The *fd* argument -1 is interpreted as a reference to the main executable of the process, similar to *NULL* value for the *name* argument to **dlopen**(). The **fdlopen**() function can be used by the code that needs to perform additional checks on the loaded objects, to prevent races with symlinking or renames.

The **dlsym**() function returns the address binding of the symbol described in the null-terminated character string *symbol*, as it occurs in the shared object identified by *handle*. The symbols exported by objects added to the address space by **dlopen**() can be accessed only through calls to **dlsym**(). Such symbols do not supersede any definition of those symbols already present in the address space when the object is loaded, nor are they available to satisfy normal dynamic linking references.

If **dlsym**() is called with the special *handle* NULL, it is interpreted as a reference to the executable or shared object from which the call is being made. Thus a shared object can reference its own symbols.

If **dlsym**() is called with the special *handle* RTLD_DEFAULT, the search for the symbol follows the algorithm used for resolving undefined symbols when objects are loaded. The objects searched are as follows, in the given order:

1. The referencing object itself (or the object from which the call to **dlsym**() is made), if that object was linked using the **-Bsymbolic** option to ld(1).

2. All objects loaded at program start-up.

3. All objects loaded via **dlopen**() with the RTLD_GLOBAL flag set in the *mode* argument.

4. All objects loaded via **dlopen**() which are in needed-object DAGs that also contain the referencing object.

If **dlsym**() is called with the special *handle* RTLD_NEXT, then the search for the symbol is limited to the shared objects which were loaded after the one issuing the call to **dlsym**(). Thus, if the function is called from the main program, all the shared libraries are searched. If it is called from a shared library, all subsequent shared libraries are searched. RTLD_NEXT is useful for implementing wrappers around library functions. For example, a wrapper function **getpid**() could access the "real" **getpid**() with dlsym(RTLD_NEXT, "getpid"). (Actually, the **dlfunc**() interface, below, should be used, since **getpid**() is a function and not a data object.)

If **dlsym**() is called with the special *handle* RTLD_SELF, then the search for the symbol is limited to the shared object issuing the call to **dlsym**() and those shared objects which were loaded after it.

The **dlsym**() function returns a null pointer if the symbol cannot be found, and sets an error condition which may be queried with **dlerror**().

The **dlvsym**() function behaves like **dlsym**(), but takes an extra argument *version*: a null-terminated character string which is used to request a specific version of *symbol*.

The **dlfunc**() function implements all of the behavior of **dlsym**(), but has a return type which can be cast to a function pointer without triggering compiler diagnostics. (The **dlsym**() function returns an object pointer; in the C standard, conversions between object and function pointer types are undefined. Some compilers and lint utilities warn about such casts.) The precise return type of **dlfunc**() is unspecified; applications must cast it to an appropriate function pointer type.

The **dlerror**() function returns a null-terminated character string describing the last error that occurred during a call to **dlopen**(), **dladdr**(), **dlinfo**(), **dlsym**(), **dlvsym**(), **dlfunc**(), or **dlclose**(). If no such error has occurred, **dlerror**() returns a null pointer. At each call to **dlerror**(), the error indication is reset. Thus in the case of two calls to **dlerror**(), where the second call follows the first immediately, the second call will always return a null pointer.

The **dlclose**() function deletes a reference to the shared object referenced by *handle*. If the reference count drops to 0, the object is removed from the address space, and *handle* is rendered invalid. Just before removing a shared object in this way, the dynamic linker calls the object's **_fini**() function, if such a function is defined by the object. If **dlclose**() is successful, it returns a value of 0. Otherwise it returns -1, and sets an error condition that can be interrogated with **dlerror**().

The object-intrinsic functions **_init**() and **_fini**() are called with no arguments, and are not expected to return values.

## NOTES

ELF executables need to be linked using the **-export-dynamic** option to ld(1) for symbols defined in the executable to become visible to **dlsym**(), **dlvsym**() or **dlfunc**()

Other ELF platforms require linking with Dynamic Linker Services Filter (libdl, -ldl) to provide **dlopen**() and other functions. FreeBSD does not require linking with the library, but supports it for compatibility.

In previous implementations, it was necessary to prepend an underscore to all external symbols in order to gain symbol compatibility with object code compiled from the C language. This is still the case when

using the (obsolete) **-aout** option to the C language compiler.

## ERRORS

The **dlopen**(), **fdlopen**(), **dlsym**(), **dlvsym**(), and **dlfunc**() functions return a null pointer in the event of errors. The **dlclose**() function returns 0 on success, or -1 if an error occurred. Whenever an error has been detected, a message detailing it can be retrieved via a call to **dlerror**().

## SEE ALSO

ld(1), rtld(1), dladdr(3), dlinfo(3), link(5)