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 'dlsym.3p' command

$ man dlsym.3p

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

dlsym ? get the address of a symbol from a symbol table handle

SYNOPSIS

```
#include <dlfcn.h>
void *dlsym(void *restrict handle, const char *restrict name);
```

DESCRIPTION

The dlsym() function shall obtain the address of a symbol (a function identifier or a data object identifier) defined in the symbol table identified by the handle argument. The handle argument is a symbol ta? ble handle returned from a call to dlopen() (and which has not since been released by a call to dlclose()), and name is the symbol's name as a character string. The return value from dlsym(), cast to a pointer to the type of the named symbol, can be used to call (in the case of a function) or access the contents of (in the case of a data object) the named symbol.

The dlsym() function shall search for the named symbol in the symbol table referenced by handle. If the symbol table was created with lazy

loading (see RTLD_LAZY in dlopen()), load ordering shall be used in dl?
sym() operations to relocate executable object files needed to resolve
the symbol. The symbol resolution algorithm used shall be dependency
order as described in dlopen().
The RTLD_DEFAULT and RTLD_NEXT symbolic constants (which may be defined
in <dlfcn.h>) are reserved for future use as special values that appli?
cations may be allowed to use for handle.

RETURN VALUE

Upon successful completion, if name names a function identifier, dl?
sym() shall return the address of the function converted from type
pointer to function to type pointer to void; otherwise, dlsym() shall
return the address of the data object associated with the data object
identifier named by name converted from a pointer to the type of the
data object to a pointer to void. If handle does not refer to a valid
symbol table handle or if the symbol named by name cannot be found in
the symbol table associated with handle, dlsym() shall return a null
pointer.
More detailed diagnostic information shall be available through dler?
ror().

ERRORS

No errors are defined.
The following sections are informative.

EXAMPLES

The following example shows how dlopen() and dlsym() can be used to ac?
cess either a function or a data object. For simplicity, error checking
has been omitted.

```
void *handle;
int (*fptr)(int), *iptr, result;
/* open the needed symbol table */
handle = dlopen("/usr/home/me/libfoo.so", RTLD_LOCAL | RTLD_LAZY);
/* find the address of the function my_function */
fptr = (int (*)(int))dlsym(handle, "my_function");
/* find the address of the data object my_object */
```

```
iptr = (int *)dlsym(handle, "my_OBJ");
/* invoke my_function, passing the value of my_OBJ as the parameter */
result = (*fptr)(*iptr);
```

APPLICATION USAGE

The following special purpose values for handle are reserved for future use and have the indicated meanings:

RTLD_DEFAULT

The identifier lookup happens in the normal global scope; that is, a search for an identifier using handle would find the same definition as a direct use of this identifier in the program code.

RTLD_NEXT Specifies the next executable object file after this one that defines name. This one refers to the executable ob? ject file containing the invocation of dlsym(). The next executable object file is the one found upon the applica? tion of a load order symbol resolution algorithm (see dlopen()). The next symbol is either one of global scope (because it was introduced as part of the original process image or because it was added with a dlopen() operation in? cluding the RTLD_GLOBAL flag), or is in an executable ob? ject file that was included in the same dlopen() operation that loaded this one.

The RTLD_NEXT flag is useful to navigate an intentionally created hier? archy of multiply-defined symbols created through interposition. For example, if a program wished to create an implementation of malloc() that embedded some statistics gathering about memory allocations, such an implementation could use the real malloc() definition to perform the memory allocation ? and itself only embed the necessary logic to imple? ment the statistics gathering function.

Note that conversion from a void * pointer to a function pointer as in:

```
fptr = (int (*)(int))dlsym(handle, "my_function");
```

is not defined by the ISO C standard. This standard requires this con? version to work correctly on conforming implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

dlclose(), dlerror(), dlopen()

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

COPYRIGHT