NAME

dladdr, dladdr1 − translate address to symbolic information

SYNOPSIS

#define _GNU_SOURCE
#include <dlfcn.h>

int dladdr(void ****addr*, Dl_info ****info***);**

int dladdr1(void ****addr*, Dl_info ****info***, void *****extra_info***, int** *flags***);**

Link with *−ldl*.

DESCRIPTION

The function **dladdr**() determines whether the address specified in *addr* is located in one of the shared objects loaded by the calling application. If it is, then **dladdr**() returns information about the shared object and symbol that overlaps *addr*. This information is returned in a *Dl_info* structure:

```
typedef struct {
    const char *dli_fname;  /* Pathname of shared object that
                               contains address */
    void       *dli_fbase;  /* Base address at which shared
                               object is loaded */
    const char *dli_sname;  /* Name of symbol whose definition
                               overlaps addr */
    void       *dli_saddr;  /* Exact address of symbol named
                               in dli_sname */
} Dl_info;
```

If no symbol matching *addr* could be found, then *dli_sname* and *dli_saddr* are set to NULL.

The function **dladdr1**() is like **dladdr**(), but returns additional information via the argument *extra_info*. The information returned depends on the value specified in *flags*, which can have one of the following values:

RTLD_DL_LINKMAP
Obtain a pointer to the link map for the matched file. The *extra_info* argument points to a pointer to a *link_map* structure (i.e., *struct link_map* **), defined in <*link.h*> as:

```
struct link_map {
    ElfW(Addr) l_addr;  /* Difference between the
                           address in the ELF file and
                           the address in memory */
    char      *l_name;  /* Absolute pathname where
                           object was found */
    ElfW(Dyn) *l_ld;    /* Dynamic section of the
                           shared object */
    struct link_map *l_next, *l_prev;
                        /* Chain of loaded objects */

    /* Plus additional fields private to the
       implementation */
};
```

RTLD_DL_SYMENT
Obtain a pointer to the ELF symbol table entry of the matching symbol. The *extra_info* argument is a pointer to a symbol pointer: *const ElfW(Sym) ***. The *ElfW*() macro definition turns its argument into the name of an ELF data type suitable for the hardware architecture. For example, on a 64-bit platform, *ElfW(Sym)* yields the data type name *Elf64_Sym*, which is defined in <*elf.h*> as:

```
typedef struct  {
    Elf64_Word    st_name;     /* Symbol name */
```

```
    unsigned char st_info;     /* Symbol type and binding */
    unsigned char st_other;    /* Symbol visibility */
    Elf64_Section st_shndx;    /* Section index */
    Elf64_Addr    st_value;    /* Symbol value */
    Elf64_Xword   st_size;     /* Symbol size */
} Elf64_Sym;
```

The *st_name* field is an index into the string table.

The *st_info* field encodes the symbol's type and binding. The type can be extracted using the macro **ELF64_ST_TYPE(st_info)** (or **ELF32_ST_TYPE()** on 32-bit platforms), which yields one of the following values:

Value	Description
STT_NOTYPE	Symbol type is unspecified
STT_OBJECT	Symbol is a data object
STT_FUNC	Symbol is a code object
STT_SECTION	Symbol associated with a section
STT_FILE	Symbol's name is file name
STT_COMMON	Symbol is a common data object
STT_TLS	Symbol is thread-local data object
STT_GNU_IFUNC	Symbol is indirect code object

The symbol binding can be extracted from the *st_info* field using the macro **ELF64_ST_BIND(st_info)** (or **ELF32_ST_BIND()** on 32-bit platforms), which yields one of the following values:

Value	Description
STB_LOCAL	Local symbol
STB_GLOBAL	Global symbol
STB_WEAK	Weak symbol
STB_GNU_UNIQUE	Unique symbol

The *st_other* field contains the symbol's visibility, which can be extracted using the macro **ELF64_ST_VISIBILITY(st_info)** (or **ELF32_ST_VISIBILITY()** on 32-bit platforms), which yields one of the following values:

Value	Description
STV_DEFAULT	Default symbol visibility rules
STV_INTERNAL	Processor-specific hidden class
STV_HIDDEN	Symbol unavailable in other modules
STV_PROTECTED	Not preemptible, not exported

RETURN VALUE

On success, these functions return a nonzero value. If the address specified in *addr* could be matched to a shared object, but not to a symbol in the shared object, then the *info->dli_sname* and *info->dli_saddr* fields are set to NULL.

If the address specified in *addr* could not be matched to a shared object, then these functions return 0. In this case, an error message is *not* available via **dlerror**(3).

VERSIONS

dladdr() is present in glibc 2.0 and later. **dladdr1**() first appeared in glibc 2.3.3.

ATTRIBUTES

For an explanation of the terms used in this section, see **attributes**(7).

Interface	Attribute	Value
dladdr(), **dladdr1**()	Thread safety	MT-Safe

CONFORMING TO

These functions are nonstandard GNU extensions that are also present on Solaris.

BUGS

Sometimes, the function pointers you pass to **dladdr**() may surprise you. On some architectures (notably i386 and x86-64), *dli_fname* and *dli_fbase* may end up pointing back at the object from which you called **dladdr**(), even if the function used as an argument should come from a dynamically linked library.

The problem is that the function pointer will still be resolved at compile time, but merely point to the *plt* (Procedure Linkage Table) section of the original object (which dispatches the call after asking the dynamic linker to resolve the symbol). To work around this, you can try to compile the code to be position-independent: then, the compiler cannot prepare the pointer at compile time any more and **gcc**(1) will generate code that just loads the final symbol address from the *got* (Global Offset Table) at run time before passing it to **dladdr**().

SEE ALSO

dl_iterate_phdr(3), **dlinfo**(3), **dlopen**(3), **dlsym**(3), **ld.so**(8)

COLOPHON

This page is part of release 5.02 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man−pages/.