NAME

dladdr - find the shared object containing a given address

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <dlfcn.h>

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

DESCRIPTION

The **dladdr**() function queries the dynamic linker for information about the shared object containing the address *addr*. The information is returned in the structure specified by *info*. The structure contains at least the following members:

const char *dli_fname	The pathname of the shared object containing the address.
void *dli_fbase	The base address at which the shared object is mapped into the address space of the calling process.
const char *dli_sname	The name of the nearest run-time symbol with a value less than or equal to *addr*. When possible, the symbol name is returned as it would appear in C source code. If no symbol with a suitable value is found, both this field and *dli_saddr* are set to NULL.
void *dli_saddr	The value of the symbol returned in dli_sname.

The **dladdr**() function is available only in dynamically linked programs.

ERRORS

If a mapped shared object containing *addr* cannot be found, **dladdr**() returns 0. In that case, a message detailing the failure can be retrieved by calling **dlerror**().

On success, a non-zero value is returned.

SEE ALSO

rtld(1), dlopen(3)

HISTORY

The **dladdr**() function first appeared in the Solaris operating system.

BUGS

This implementation is bug-compatible with the Solaris implementation. In particular, the following bugs are present:

- If *addr* lies in the main executable rather than in a shared library, the pathname returned in *dli_fname* may not be correct. The pathname is taken directly from *argv[0]* of the calling process. When executing a program specified by its full pathname, most shells set *argv[0]* to the pathname. But this is not required of shells or guaranteed by the operating system.

- If *addr* is of the form *&func*, where *func* is a global function, its value may be an unpleasant surprise. In dynamically linked programs, the address of a global function is considered to point to its program linkage table entry, rather than to the entry point of the function itself. This causes most global functions to appear to be defined within the main executable, rather than in the shared libraries where the actual code resides.

- Returning 0 as an indication of failure goes against long-standing Unix tradition.