NAME

dllockinit - register thread locking methods with the dynamic linker

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <dlfcn.h>

void
dllockinit(*void *context, void *(*lock_create)(void *context), void (*rlock_acquire)(void *lock),*
*void (*wlock_acquire)(void *lock), void (*lock_release)(void *lock),*
*void (*lock_destroy)(void *lock), void (*context_destroy)(void *context)*);

DESCRIPTION

Due to enhancements in the dynamic linker, this interface is no longer needed. It is deprecated and will be removed from future releases. In current releases it still exists, but only as a stub which does nothing.

Threads packages can call **dllockinit**() at initialization time to register locking functions for the dynamic linker to use. This enables the dynamic linker to prevent multiple threads from entering its critical sections simultaneously.

The *context* argument specifies an opaque context for creating locks. The dynamic linker will pass it to the *lock_create* function when creating the locks it needs. When the dynamic linker is permanently finished using the locking functions (e.g., if the program makes a subsequent call to **dllockinit**() to register new locking functions) it will call *context_destroy* to destroy the context.

The *lock_create* argument specifies a function for creating a read/write lock. It must return a pointer to the new lock.

The *rlock_acquire* and *wlock_acquire* arguments specify functions which lock a lock for reading or writing, respectively. The *lock_release* argument specifies a function which unlocks a lock. Each of these functions is passed a pointer to the lock.

The *lock_destroy* argument specifies a function to destroy a lock. It may be NULL if locks do not need to be destroyed. The *context_destroy* argument specifies a function to destroy the context. It may be NULL if the context does not need to be destroyed.

Until **dllockinit**() is called, the dynamic linker protects its critical sections using a default locking mechanism which works by blocking the SIGVTALRM, SIGPROF, and SIGALRM signals. This is

sufficient for many application level threads packages, which typically use one of these signals to implement preemption. An application which has registered its own locking methods with **dllockinit**() can restore the default locking by calling **dllockinit**() with all arguments NULL.

SEE ALSO

rtld(1), signal(3)

HISTORY

The **dllockinit**() function first appeared in FreeBSD 4.0.