NAME

mtx_pool, **mtx_pool_alloc**, **mtx_pool_find**, **mtx_pool_lock**, **mtx_pool_lock_spin**, **mtx_pool_unlock**, **mtx_pool_unlock_spin**, **mtx_pool_create**, **mtx_pool_destroy** - mutex pool routines

SYNOPSIS

#include <sys/param.h>
#include <sys/lock.h>
#include <sys/mutex.h>

*struct mtx **
mtx_pool_alloc(*struct mtx_pool *pool*);

*struct mtx **
mtx_pool_find(*struct mtx_pool *pool*, *void *ptr*);

void
mtx_pool_lock(*struct mtx_pool *pool*, *void *ptr*);

void
mtx_pool_lock_spin(*struct mtx_pool *pool*, *void *ptr*);

void
mtx_pool_unlock(*struct mtx_pool *pool*, *void *ptr*);

void
mtx_pool_unlock_spin(*struct mtx_pool *pool*, *void *ptr*);

*struct mtx_pool **
mtx_pool_create(*const char *mtx_name*, *int pool_size*, *int opts*);

void
mtx_pool_destroy(*struct mtx_pool **poolp*);

DESCRIPTION

Mutex pools are designed to be used as short term leaf mutexes; i.e., the last mutex one might acquire before calling mtx_sleep(9). They operate using a shared pool of mutexes. A mutex may be chosen from the pool based on a supplied pointer, which may or may not point to anything valid, or the caller may allocate an arbitrary shared mutex from the pool and save the returned mutex pointer for later use.

The shared mutexes in the *mtxpool_sleep* mutex pool, which is created by default, are standard, non-

recursive, blockable mutexes, and should only be used in appropriate situations. The mutexes in the *mtxpool_lockbuilder* mutex pool are similar, except that they are initialized with the MTX_NOWITNESS flag so that they may be used to build higher-level locks. Other mutex pools may be created that contain mutexes with different properties, such as spin mutexes.

The caller can lock and unlock mutexes returned by the pool routines, but since the mutexes are shared, the caller should not attempt to destroy them or modify their characteristics. While pool mutexes are normally leaf mutexes (meaning that one cannot depend on any ordering guarantees after obtaining one), one can still obtain other mutexes under carefully controlled circumstances. Specifically, if one has a private mutex (one that was allocated and initialized by the caller), one can obtain it after obtaining a pool mutex if ordering issues are carefully accounted for. In these cases the private mutex winds up being the true leaf mutex.

Pool mutexes have the following advantages:

1. No structural overhead; i.e., they can be associated with a structure without adding bloat to it.
2. Mutexes can be obtained for invalid pointers, which is useful when one uses mutexes to interlock destructor operations.
3. No initialization or destruction overhead.
4. Can be used with mtx_sleep(9).

And the following disadvantages:

1. Should generally only be used as leaf mutexes.
2. Pool/pool dependency ordering cannot be guaranteed.
3. Possible L1 cache mastership contention between CPUs.

mtx_pool_alloc() obtains a shared mutex from the specified pool. This routine uses a simple rover to choose one of the shared mutexes managed by the **mtx_pool** subsystem.

mtx_pool_find() returns the shared mutex associated with the specified address. This routine will create a hash out of the pointer passed into it and will choose a shared mutex from the specified pool based on that hash. The pointer does not need to point to anything real.

mtx_pool_lock(), **mtx_pool_lock_spin**(), **mtx_pool_unlock**(), and **mtx_pool_unlock_spin**() lock and unlock the shared mutex from the specified pool associated with the specified address; they are a combination of **mtx_pool_find**() and mtx_lock(9), mtx_lock_spin(9), mtx_unlock(9), and mtx_unlock_spin(9), respectively. Since these routines must first find the mutex to operate on, they are not as fast as directly using the mutex pointer returned by a previous invocation of **mtx_pool_find**() or **mtx_pool_alloc**().

mtx_pool_create() allocates and initializes a new mutex pool of the specified size. The pool size must be a power of two. The *opts* argument is passed to mtx_init(9) to set the options for each mutex in the pool.

mtx_pool_destroy() calls mtx_destroy(9) on each mutex in the specified pool, deallocates the memory associated with the pool, and assigns NULL to the pool pointer.

SEE ALSO

locking(9), mutex(9)

HISTORY

These routines first appeared in FreeBSD 5.0.