NAME

rwlock, **rw_init**, **rw_init_flags**, **rw_destroy**, **rw_rlock**, **rw_wlock**, **rw_runlock**, **rw_wunlock**, **rw_unlock**, **rw_try_rlock**, **rw_try_upgrade**, **rw_try_wlock**, **rw_downgrade**, **rw_sleep**, **rw_initialized**, **rw_owned**, **rw_assert**, **RW_SYSINIT**, **RW_SYSINIT_FLAGS** - kernel reader/writer lock

SYNOPSIS

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

void
rw_init(*struct rwlock *rw*, *const char *name*);

void
rw_init_flags(*struct rwlock *rw*, *const char *name*, *int opts*);

void
rw_destroy(*struct rwlock *rw*);

void
rw_rlock(*struct rwlock *rw*);

void
rw_wlock(*struct rwlock *rw*);

int
rw_try_rlock(*struct rwlock *rw*);

int
rw_try_wlock(*struct rwlock *rw*);

void
rw_runlock(*struct rwlock *rw*);

void
rw_wunlock(*struct rwlock *rw*);

void
rw_unlock(*struct rwlock *rw*);

int
rw_try_upgrade(*struct rwlock *rw*);

void
rw_downgrade(*struct rwlock *rw*);

int
rw_sleep(*void *chan, struct rwlock *rw, int priority, const char *wmesg, int timo*);

int
rw_initialized(*const struct rwlock *rw*);

int
rw_wowned(*const struct rwlock *rw*);

options INVARIANTS
options INVARIANT_SUPPORT
void
rw_assert(*const struct rwlock *rw, int what*);

#include <sys/kernel.h>

RW_SYSINIT(*name, struct rwlock *rw, const char *desc*);

RW_SYSINIT_FLAGS(*name, struct rwlock *rw, const char *desc, int flags*);

DESCRIPTION

Reader/writer locks allow shared access to protected data by multiple threads, or exclusive access by a single thread. The threads with shared access are known as *readers* since they only read the protected data. A thread with exclusive access is known as a *writer* since it can modify protected data.

Although reader/writer locks look very similar to sx(9) locks, their usage pattern is different. Reader/writer locks can be treated as mutexes (see mutex(9)) with shared/exclusive semantics. Unlike sx(9), an **rwlock** can be locked while holding a non-spin mutex, and an **rwlock** cannot be held while sleeping. The **rwlock** locks have priority propagation like mutexes, but priority can be propagated only to writers. This limitation comes from the fact that readers are anonymous. Another important property is that readers can always recurse, and exclusive locks can be made recursive selectively.

Macros and Functions

rw_init(*struct rwlock *rw, const char *name*)

Initialize structure located at *rw* as reader/writer lock, described by name *name*. The description is used solely for debugging purposes. This function must be called before any other operations on the lock.

rw_init_flags(*struct rwlock *rw, const char *name, int opts*)

Initialize the rw lock just like the **rw_init**() function, but specifying a set of optional flags to alter the behaviour of *rw*, through the *opts* argument. It contains one or more of the following flags:

RW_DUPOK Witness should not log messages about duplicate locks being acquired.

RW_NOPROFILE Do not profile this lock.

RW_NOWITNESS
Instruct witness(4) to ignore this lock.

RW_QUIET Do not log any operations for this lock via ktr(4).

RW_RECURSE Allow threads to recursively acquire exclusive locks for *rw*.

RW_NEW If the kernel has been compiled with **option INVARIANTS**, **rw_init_flags**() will assert that the *rw* has not been initialized multiple times without intervening calls to **rw_destroy**() unless this option is specified.

rw_rlock(*struct rwlock *rw*)

Lock *rw* as a reader. If any thread holds this lock exclusively, the current thread blocks, and its priority is propagated to the exclusive holder. The **rw_rlock**() function can be called when the thread has already acquired reader access on *rw*. This is called "recursing on a lock".

rw_wlock(*struct rwlock *rw*)

Lock *rw* as a writer. If there are any shared owners of the lock, the current thread blocks. The **rw_wlock**() function can be called recursively only if *rw* has been initialized with the RW_RECURSE option enabled.

rw_try_rlock(*struct rwlock *rw*)

Try to lock *rw* as a reader. This function will return true if the operation succeeds, otherwise 0 will be returned.

rw_try_wlock(*struct rwlock *rw*)

Try to lock *rw* as a writer. This function will return true if the operation succeeds, otherwise 0 will be returned.

rw_runlock(*struct rwlock *rw*)
This function releases a shared lock previously acquired by **rw_rlock**().

rw_wunlock(*struct rwlock *rw*)
This function releases an exclusive lock previously acquired by **rw_wlock**().

rw_unlock(*struct rwlock *rw*)
This function releases a shared lock previously acquired by **rw_rlock**() or an exclusive lock previously acquired by **rw_wlock**().

rw_try_upgrade(*struct rwlock *rw*)
Attempt to upgrade a single shared lock to an exclusive lock. The current thread must hold a shared lock of *rw*. This will only succeed if the current thread holds the only shared lock on *rw*, and it only holds a single shared lock. If the attempt succeeds **rw_try_upgrade**() will return a non-zero value, and the current thread will hold an exclusive lock. If the attempt fails **rw_try_upgrade**() will return zero, and the current thread will still hold a shared lock.

rw_downgrade(*struct rwlock *rw*)
Convert an exclusive lock into a single shared lock. The current thread must hold an exclusive lock of *rw*.

rw_sleep(*void *chan, struct rwlock *rw, int priority, const char *wmesg, int timo*)
Atomically release *rw* while waiting for an event. For more details on the parameters to this function, see sleep(9).

rw_initialized(*const struct rwlock *rw*)
This function returns non-zero if *rw* has been initialized, and zero otherwise.

rw_destroy(*struct rwlock *rw*)
This functions destroys a lock previously initialized with **rw_init**(). The *rw* lock must be unlocked.

rw_wowned(*const struct rwlock *rw*)
This function returns a non-zero value if the current thread owns an exclusive lock on *rw*.

rw_assert(*const struct rwlock *rw, int what*)
This function allows assertions specified in *what* to be made about *rw*. If the assertions are not true and the kernel is compiled with **options INVARIANTS** and **options INVARIANT_SUPPORT**, the kernel will panic. Currently the following base assertions are supported:

RA_LOCKED Assert that current thread holds either a shared or exclusive lock of *rw*.

RA_RLOCKED Assert that current thread holds a shared lock of *rw*.

RA_WLOCKED Assert that current thread holds an exclusive lock of *rw*.

RA_UNLOCKED Assert that current thread holds neither a shared nor exclusive lock of *rw*.

In addition, one of the following optional flags may be specified with RA_LOCKED, RA_RLOCKED, or RA_WLOCKED:

RA_RECURSED Assert that the current thread holds a recursive lock of *rw*.

RA_NOTRECURSED Assert that the current thread does not hold a recursive lock of *rw*.

SEE ALSO

locking(9), mutex(9), panic(9), sema(9), sx(9)

HISTORY

These functions appeared in FreeBSD 7.0.

AUTHORS

The **rwlock** facility was written by John Baldwin. This manual page was written by Gleb Smirnoff.

BUGS

A kernel without WITNESS cannot assert whether the current thread does or does not hold a read lock. RA_LOCKED and RA_RLOCKED can only assert that *any* thread holds a read lock. They cannot ensure that the current thread holds a read lock. Further, RA_UNLOCKED can only assert that the current thread does not hold a write lock.

Reader/writer is a bit of an awkward name. An **rwlock** can also be called a "Robert Watson" lock if desired.