NAME

pthread_spin_init, pthread_spin_destroy − initialize or destroy a spin lock

LIBRARY

POSIX threads library (*libpthread*, *−lpthread*)

SYNOPSIS

```
#include <pthread.h>

int pthread_spin_init(pthread_spinlock_t *lock, int pshared);
int pthread_spin_destroy(pthread_spinlock_t *lock);
```

Feature Test Macro Requirements for glibc (see **feature_test_macros**(7)):

pthread_spin_init(), **pthread_spin_destroy**():
_POSIX_C_SOURCE >= 200112L

DESCRIPTION

General note: Most programs should use mutexes instead of spin locks. Spin locks are primarily useful in conjunction with real-time scheduling policies. See NOTES.

The **pthread_spin_init**() function allocates any resources required for the use of the spin lock referred to by *lock* and initializes the lock to be in the unlocked state. The *pshared* argument must have one of the following values:

PTHREAD_PROCESS_PRIVATE
The spin lock is to be operated on only by threads in the same process as the thread that calls **pthread_spin_init**(). (Attempting to share the spin lock between processes results in undefined behavior.)

PTHREAD_PROCESS_SHARED
The spin lock may be operated on by any thread in any process that has access to the memory containing the lock (i.e., the lock may be in a shared memory object that is shared among multiple processes).

Calling **pthread_spin_init**() on a spin lock that has already been initialized results in undefined behavior.

The **pthread_spin_destroy**() function destroys a previously initialized spin lock, freeing any resources that were allocated for that lock. Destroying a spin lock that has not been previously been initialized or destroying a spin lock while another thread holds the lock results in undefined behavior.

Once a spin lock has been destroyed, performing any operation on the lock other than once more initializing it with **pthread_spin_init**() results in undefined behavior.

The result of performing operations such as **pthread_spin_lock**(3), **pthread_spin_unlock**(3), and **pthread_spin_destroy**() on *copies* of the object referred to by *lock* is undefined.

RETURN VALUE

On success, there functions return zero. On failure, they return an error number. In the event that **pthread_spin_init**() fails, the lock is not initialized.

ERRORS

pthread_spin_init() may fail with the following errors:

EAGAIN
The system has insufficient resources to initialize a new spin lock.

ENOMEM
Insufficient memory to initialize the spin lock.

VERSIONS

These functions were added in glibc 2.2.

STANDARDS

POSIX.1-2001.

Support for process-shared spin locks is a POSIX option. The option is supported in the glibc implementation.

NOTES

Spin locks should be employed in conjunction with real-time scheduling policies (**SCHED_FIFO**, or possibly **SCHED_RR**). Use of spin locks with nondeterministic scheduling policies such as **SCHED_OTHER** probably indicates a design mistake. The problem is that if a thread operating under such a policy is scheduled off the CPU while it holds a spin lock, then other threads will waste time spinning on the lock until the lock holder is once more rescheduled and releases the lock.

If threads create a deadlock situation while employing spin locks, those threads will spin forever consuming CPU time.

User-space spin locks are *not* applicable as a general locking solution. They are, by definition, prone to priority inversion and unbounded spin times. A programmer using spin locks must be exceptionally careful not only in the code, but also in terms of system configuration, thread placement, and priority assignment.

SEE ALSO

pthread_mutex_init(3), **pthread_mutex_lock**(3), **pthread_spin_lock**(3), **pthread_spin_unlock**(3), **pthreads**(7)