N2019
Revised Proposal for DR469

Blaine Garst
2016-03-10

While searching **7.26.4 Mutex functions** for an appropriate place to place the Suggested TC of DR469 revealed that the required matching behavior of a recursive mx_unlock was simply missing. N1907 was proposed, but not deemed sufficient, and so this is a revision based on feedback at WG14 meetings.

I. Recursive mtx_lock underspecification

Recursive mutexes have thread specific behaviors, namely ongoing success on locking an already locked recursive mutex, as specified in 7.26.4.3p2. Each successive lock operation must also be matched with a call to **`mtx_unlock`**, yet this is not stated.

Proposed Technical Corrigenda

Add as new paragraph to 7.26.4 *Mutex functions*

> After initialization and before destruction a mutex shall be locked and subsequently unlocked by one calling thread at a time. In the case of a recursive mutex additional matched pairs of lock and unlock requests shall also succeed by that calling thread, but with no observable effect.

In 7.26.4.3 *The **`mtx_lock`** function* p2

Replace

> The mtx_lock function blocks until it locks the mutex pointed to by **`mtx`**. If the mutex is non-recursive, it shall not be locked by the calling thread. Prior calls to **`mtx_unlock`** on the same mutex shall synchronize with this operation.

with

> If the mutex pointed by by **`mtx`** is non-recursive, the mutex shall not already be locked by the calling thread, and the **`mtx_lock`** function shall block until it the mutex is locked. If the mutex is recursive and is not already locked by the calling thread **`mtx_lock`** shall block until the mutex is locked, and the recursion count is set to one. Prior calls to **`mtx_unlock`** that unlock the same mutex shall

synchronize with this operation.

If the mutex is recursive and already locked by the calling thread, **mtx_lock** shall succeed after incrementing the recursion count.

In 7.26.4.4 *The* ***mtx_timedlock*** *function* p2

replace

The **mtx_timedlock** function endeavors to block until it locks the mutex pointed to by **mtx** or until after the **TIME_UTC**-based calendar time pointed to by **ts**. The specified mutex shall support timeout. If the operation succeeds, prior calls to **mtx_unlock** on the same mutex shall synchronize with this operation.

with

The mutex pointed to by **mtx** shall support timeout. If the mutex pointed to by **mtx** is recursive and is already locked by the calling thread, the recursion count is incremented and the call succeeds. Otherwise, The **mtx_timedlock** function endeavors to block until it locks the mutex pointed to by **mtx** or until after the **TIME_UTC**-based calendar time pointed to by **ts**. If the mutex was locked by this operation and the mutex is recursive, the recursion count is set to one.

In 7.26.4.5 *The* ***mtx_trylock*** *function* p2

replace

The **mtx_trylock** function endeavors to lock the mutex pointed to by **mtx**. If the * mutex already locked, the function returns without blocking. If the operation succeeds, prior calls to **mtx_unlock** on the same mutex shall synchronize with this operation.

with

If the mutex pointed to by **mtx** is recursive and already locked by the calling thread, the recursion count is incremented and the call succeeds. Otherwise, the **mtx_trylock** function endeavors to lock the mutex pointed to by **mtx**. If the mutex is already locked by another thread, the function returns without blocking. If the operation locks the mutex, prior calls to **mtx_unlock** on the same mutex shall synchronize with this operation, and further if the mutex is recursive its

recursion count is set to one.

In 7.26.4.6 The **mtx_unlock** function p2

replace

> The **mtx_unlock** function unlocks the mutex pointed to by **mtx**. The mutex pointed to by **mtx** shall be locked by the calling thread.

with

> The mutex pointed to by **mtx** shall be locked by the calling thread. If the mutex is non-recursive, it is unlocked, If the mutex is recursive, the recursion count is decremented, and if non-zero, the call succeeds, and if zero, the mutex shall be unlocked and the call shall synchronize with subsequent locking calls.

2 Underspecification of thread termination

From DR469, we wish to state explicitly that an operation on a mutex that remains locked after thread termination results in undefined behavior. The rationale is that the implementation of the thread waiting may require a finite resource.

Proposed Technical Corrigendum

To 7.26.5.1 *The **thrd_create** function* add after paragraph 2

> The thread terminates when either the function pointed by **func** returns and sets the result code to the returned value, or a call is made by the thread to **thrd_exit**.
>
> The behavior of a program is undefined if a thread terminates without unlocking every mutex that the thread has locked.