

Re: recursive mutexes

Source: <http://unix.derkeiler.com/Newsgroups/comp.unix.programmer/2005-05/0349.html>

From: Uenal Mutlu (520001085531-0001_at_t-online.de)

Date: 05/15/05

Date: Sun, 15 May 2005 08:05:49 +0200

"David Schwartz" wrote

> *red floyd* wrote:

>

> > *May I ask why?*

>

> *There are two possibilities:*

>

> *1) You don't know the mutex is being used recursively. In this case, the*

> *recursive mutex will hide a serious problem.*

>

> *2) You know the mutex is being recursively. In this case, just don't*

> *lock it since you know it's already locked anyway.*

>

> *The problem is that recursive mutexes hide something from you and that*

> *something is extremely important. Consider code like this:*

>

> *A) Lock mutex*

> *B) Unlock mutex*

> *C) Do something, assuming the mutex is unlocked*

>

> *What happens if the mutex is recursive?*

Recursive locks work only for the same (current) thread.

The code above is IMO wrong, obviously it should be:

A) Lock mutex

B) Do something

C) Unlock mutex

and this is ok too (here the recursive feature is in action):

A) Lock mutex

B) Do something

C) Lock mutex

D) Do something2

E) Unlock mutex

F) Unlock mutex

C and E are redundant, however application code (the current thread)

comp.unix.programmer: Re: recursive mutexes

cannot always know that it already has the lock, though it could test for it. But testing is unnecessary because calling such a Lock() will already do the testing implicitly, so then there is no need for an explicit testing in applic code, it just would make code longer and slow down the performance.

To clarify:

there is Lock() without recursive feature,
and there is Lock() with recursive feature.

And: such lock objects are initialized at creation.

- > *The only value of a recursive mutex is that it allows you to write a*
- > *function that works properly whether or not a mutex is locked. But the*
- > *caller must know that the mutex is locked anyway, so why not just have two*

usually the calling thread must not know that; it simply can issue a Lock() request of its own. If it is already locked by this thread then recursion is in effect and the lock will be granted very fast.

- > *versions of the function? (With the one you call without a lock possibly*
- > *just being a wrapper that grabs the lock and calls the other function.)*
- >
- > *It's just too hard and dangerous to write sensible code that works with*
- > *a mutex that might or might not start out with that mutex locked and with no*
- > *way to tell which. And the only value of recursive mutexes is that they let*
- > *you do this.*

Lock objects are initialized at creation, they don't have a random state.

Recursive locks are very practical. It saves coding and prevents from self-deadlocking.