

Re: semaphore question

Source: <http://linux.derkeiler.com/Newsgroups/comp.os.linux.development.system/2006-08/msg00183.html>

- *From:* "David Schwartz" <davids@xxxxxxxxxxxxxx>
 - *Date:* 16 Aug 2006 04:35:37 -0700
-

Josef Moellers wrote:

David Schwartz wrote:

Josef Moellers wrote:

You protect neither files nor global data.

Mutexes protect data.

Yes and no.

Mutexes serialize execution of critical sections.
What you do inside these critical sections is up to you.
You might modify global data, you might modify the internal state of a device in multiple steps, ...

It is only when you modify data that might be accessed or modified elsewhere that you need a critical section. (Data being loosely defined to include things like the state of a device or communications channel.)

I think this is a very bad way to think about mutexes. Mutexes are associated with the objects they protect, not the code that accesses them. Code should not acquire a mutex that protects that code but a

Re: semaphore question

mutex that protects the object that code manipulates.

That is a specific mutex.

Generally speaking (and that was what I was referring to), a mutex is a semaphore initialized with 1.

Huh? What other kind of mutex is there? Whether it's a mutex, critical section, semaphore, or whatever, its point is to prevent concurrent access to the same "thing", whether by the same code or different code. That "thing" is not code. It is data or state.

Thinking about mutexes as protecting code leads to people putting mutexes around code that has no need to be protected. Only shared data (or resources of some kind) requires protection.

Any global state that might be modified concurrently requires protection.

Right, the **state** needs protection.

Code never needs to be protected. Concurrent accesses of the same code is totally safe and permitted.

It depends on what the code does.

Right, if the code accesses something that needs to be protected, then that thing needs to be protected.

But I guess I'm thinking of a generic mutex (1-Semaphore) and you're thinking of a specific use of a mutex.

How would you use a mutex or semaphore to protect something other than shared state of some kind? Shared state has nothing to do with the particular code that accesses it, for example, the same protection is required whether the state is accessed by the same code or different code in multiple threads.

It is extremely bad practice and does in fact lead to misunderstandings and bad code to think about synchronization mechanisms of any kind as

Re: semaphore question

Re: semaphore question

protecting code. Synchronization mechanisms protect the things that more than one thread might want to access or modify, and that is not code.

Concurrent access to code is perfectly safe so long as there isn't conflicting access to the same data.

It is impossible to create a problem with conflicting code unless there is also conflicting data.

If there's conflicting data, there will be a problem whether or not there's concurrent use of the same code.

DS

.