

Re: Obscure mutex problem

Source: <http://linux.derkeiler.com/Newsgroups/comp.os.linux.development.apps/2007-09/msg00037.html>

- *From:* Rainer Weikusat <rweikusat@xxxxxxxxxxx>
 - *Date:* Thu, 06 Sep 2007 12:50:52 +0200
-

David Given <dg@xxxxxxxxxxx> writes:

Rainer Weikusat wrote:

David Given <dg@xxxxxxxxxxx> writes:

– when initially writing the code, I discovered that I had to set
PTHREAD_MUTEX_RECURSIVE to make the mutexes
work at all... but then later
found this was no longer necessary. No doubt this is due to
another
change I made, but I still don't understand it.

D'oh. I'm sorry, I managed to commit a typo in that paragraph, to vast confusion.

What I actually meant to say was that I had to set
PTHREAD_PROCESS_SHARED. I was finding that thread A would lock the
mutex fine, and then thread B would attempt to lock it, but instead
of blocking would instead deadlock. *That* I didn't
understand. PTHREAD_PROCESS_SHARED made it work. Later when I went
back to try and figure out why it was necessary, I discovered it
wasn't any more.

Hmm ... could it be possible that you were keeping the mutex locked
across a fork?

I use PTHREAD_MUTEX_RECURSIVE because I'm used to recursive mutexes from
elsewhere and would rather deal with mutexes set up the way I expect.

[...]

– my test machines include an ARM-based NSLU2 with
linuxthreads (one kernel
process per thread) and a i386-based PC with NPTL (real
kernel

Re: Obscure mutex problem

threads).

Please don't perpetuate stupid myths about Linux threading support.

Possibly slightly less inflammatory language, please?

That pre-NPTL-Linux does not support 'real' multithreading 'but some weird emulation using processes' is a myth. It is a stupid myth, because such a thing cannot possibly exist: A process is comprised of a set of resource shared among its threads and it always runs in an address space of its own. Real 'genetic UNIX(*)' used to have a specific representation for processes (proc struct/ u area) and the concept of 'light-weight processes' as something different than 'real processes' exists because of technical properties of the inherited representation and the way it was handled by the kernel. Since Linux is not derived from any 'genetic UNIX(*)' codebase, it never had this particular problem, which is an additional reason for the adjective I was using.

I do not quite understand why this would be regarded as 'inflammatory'. It's a technical fact. How can one get upset about this? And I neither wrote about you-the-person nor did I mean to. On the other hand, I have had one discussion with a 'NT-kiddie' running around and telling everyone that 'Linux is inferior because it does not support threads, but only processes', which had absolutely no clue about anything, and I don't need another.

[...]

linuxthreads and NPTL are actually have some fairly major differences under the hood, especially when it comes to TLS, and I thought it was worth emphasizing that I'd tried the program on both implementations and it worked fine both ways.

They are both using the same basic kernel support (clone(2)) and the notion of what is regarded as a (single-threaded) process and what as one of several threads belonging to a process has not changed.

[...]

I should point out that the only difference between the foreground (working) version and the background (non-working) version is that the background version calls daemon(0,0).

Re: Obscure mutex problem

A guess would be that some thread locks the mutex before daemon forks and because this thread isn't the one executing the fork, the locked mutex in the child can never be unlocked again. At least that's the specified behaviour. 'Some NPTL versions' support unlocking mutexes from arbitrary threads, though.

.