Win32 Named mutex not released when process crashes

Asked 11 years, 10 months ago Modified 11 years, 10 months ago
Viewed 9k times

12

I have 2 processes (A, B) sharing the same mutex (using WaitForSingleObject / ReleaseMutex calls). Everything works fine, but when process A crashes, process B is humming along happily. When I restart process A, there's a deadlock.

Deeper investigation reveals that process B can successfully call ReleaseMutex() twice after process A crashes.

My interpretation: After process A crashes, the mutex is still locked, but ownership of the mutex transfers readily to process B (which is a bug). That's why it's humming along happily, calling WaitForSingleObject (getting WAIT_OBJECT_0 in return) and ReleaseMutex (getting TRUE in return).

Is it possible to use a named synchronization primitive similar to Mutex in such a way that a crash in process A will release the mutex?

One solution is to use SEH and catch the crash and release mutex, but I really hope Windows has a robust primitive that doesn't deadlock like that on process crash.

winapi win32-process

Share

Improve this question

Follow

asked Feb 20, 2013 at 16:43

Sergiy Migdalskiy

1,086 ● 1 ● 11 ● 22

2 This may be an interesting article for you to read: blogs.msdn.com/b/oldnewthing/archive/2005/09/12/463977.aspx – Tony The Lion Feb 20, 2013 at 16:52

Fixed link as of 2024: devblogs.microsoft.com/oldnewthing/20050912-14/?p=34253 – Jack G Jul 17 at 15:48

2 Answers

Sorted by: Highest score (default)

29

Some basic assumptions you have to make here about how a mutex works on Windows:

- a mutex is an operating system object that's reference-counted. It will not disappear until the last handle on the mutex is closed
- any handle that's left unclosed when a process terminates is closed by the operating system, decrementing the reference count
- a mutex is *re-entrant*, calling WaitForSingleObject on a mutex on the same thread succeeds and needs to

be balanced with an equal number of ReleaseMutex calls

- an owned mutex becomes *abandoned* when the thread that owns it terminates without calling ReleaseMutex. Calling WaitForSingleObject on a mutex in this state generates the WAIT_ABANDONED error return code
- it is never a bug in the operating system.

So you can draw conclusions from this by what you observed. Nothing happens to the mutex when A crashes, B still has an handle on it. The only possible way B can notice that A crashed is when A crashed while it owned the mutex. Very low odds for that and easily observed since B will deadlock. Far more likely is that B will happily motor on since it is now completely unobstructed, nobody else is going to acquire the mutex anymore.

Furthermore, a deadlock when A starts back proves something you already knew: B owns the mutex permanently for some reason. Possibly because it acquired the mutex recursively. You know this because you noticed you had to call ReleaseMutex twice. This is a bug you need to fix.

You'll need to protect yourself against a crashing sibling process and you need to write explicit code for that. Call OpenProcess on the sibling to obtain a handle on the process object. A WaitForSingleObject call on the handle will complete when the process terminates.

Share Improve this answer

Follow

edited Feb 20, 2013 at 22:28

answered Feb 20, 2013 at 18:10

Hans Passant

940k ● 148 ● 1.7k ● 2.6k

1 Sorry I didn't flag this as an answer but.. better late then never. Thanks, I didn't know the meaning of the abandoned state. – Sergiy Migdalskiy Nov 26, 2017 at 4:03

11

If the process holding the mutex crashes, then it becomes abandoned. It's up to the other application how it deals with this state returned from the wait functions.

If it gets `WAIT_ABANDONED` back then it can either carry on as if all was ok (presumably what it does now) or "potentially unstable data, proceed with caution". The ownership is not passed to another process automatically.

Share Improve this answer

Follow

answered Feb 20, 2013 at 17:25

Deanna

24.2k ● 7 ● 73 ● 159

12 My guess is that Process B receives `WAIT_ABANDONED` and thinks that it means `WAIT_FAILED` (rather than "succeeded in an unexpected way") so it tries to acquire the mutex a second time. – Raymond Chen Feb 20, 2013 at 17:44