What is this race condition that the OpenMutex documentation is trying to warn me about?

devblogs.microsoft.com/oldnewthing/20161208-00

December 8, 2016

Raymond Chen

A customer asked for clarification on what they considered an enigmatic sentence in the documentation for the `OpenMutex` function:

> If your multithreaded application must repeatedly create, open, and close a named mutex object, a race condition can occur. In this situation, it is better to use **CreateMutex** instead of **OpenMutex**, because **CreateMutex** opens a mutex if it exists and creates it if it does not.

"What is this race condition the documentation is talking about? Our program uses `OpenMutex` and we are wondering if we should switch to `CreateMutex`."

Consider two threads. One thread calls `CreateMutex`, then `CloseHandle`, then `CreateMutex`, then `CloseHandle`, then `CreateMutex`, then `CloseHandle`, and so on.

The other thread calls `OpenMutex`.

The race condition is that the second thread's call to `OpenMutex` will fail if it takes place after the first thread calls `CloseHandle` and before it gets to make its next call to `CreateMutex`.

One of my colleagues couldn't understand why MSDN bothers to say anything about this situation at all. "You can't open a mutex that doesn't exist. Duh. I think this adds more confusion than it helps."

I suspect the reason why MSDN bothers to say anything about this is that there was a customer who had two threads. One thread calls `CreateMutex`, then `CloseHandle`, then `CreateMutex`, then `CloseHandle`, then `CreateMutex`, then `CloseHandle`, and so on. The other thread calls `OpenMutex`.

This customer found that if the second thread calls `OpenMutex` at an inopportune time, the call fails. They then insisted that something be added to the documentation to state explicitly that a bad idea is a bad idea. Probably because they needed something in writing to show their management in order to justify the time they are going to need to spend fixing the bug.

Once again, MSDN has been forced into being a pawn in some company's internal politics.

Raymond Chen

Follow