You can name your car, and you can name your kernel objects, but there is a qualitative difference between the two

devblogs.microsoft.com/oldnewthing/20141001-00

October 1, 2014

Raymond Chen

A customer reported that the `WaitForSingleObject` appeared to be unreliable.

> We have two threads, one that waits on an event and the other that signals the event. But we found that sometimes, signaling the event does not wake up the waiting thread. We have to signal it twice. What are the conditions under which `WaitForSingleObject` will ignore a signal?
>
> ```
> // cleanup and error checking elided for expository purposes
> void Thread1()
> {
>   // Create an auto-reset event, initially unsignaled
>   HANDLE eventHandle = CreateEvent(NULL, FALSE, FALSE, TEXT("MyEvent"));
>   // Kick off the background thread and give it the handle
>   CreateThread(..., Thread2, eventHandle, ...);
>   // Wait for the event to be signaled
>   WaitForSingleObject(eventHandle, INFINITE);
> }
> DWORD CALLBACK Thread2(void *eventHandle)
> {
>  ResetEvent(eventHandle); // start with a clean slate
>  DoStuff();
>  // All the calls to SetEvent succeed.
>  SetEvent(eventHandle); // this does not always wake up Thread1
>  SetEvent(eventHandle); // need to add this line
>  return 0;
> }
> ```

Remember, you generally shouldn't start with the conspiracy theory. The problem is most likely close to home.

People offered a variety of theories as to what may be wrong. One possibility is that some other code in the process is calling `ResetEvent` on the event handle. Another is that some other code in the process has a bug where it is calling `ResetEvent` on the wrong event

handle.

I asked about the name.

I have a friend who names her car. Whenever she gets a new car, she agonizes over what to call it. She'll drive it for a few days to see what its personality is and eventually choose a name that suits the vehicle. And thereafter, whenever she refers to her car, she uses the name. (She also assigns the car a gender.)

If you like naming your car, then that's great. But there's a difference between naming your car and naming your kernel objects. When you give your car a name, that name is just for your private use. On the other hand, if you give your kernel object a name, other people can use that name to access your object. And once they have access to your object, they can do funky things to it, like reset it.

Imagine if you decided to name your car Clara, and any time somebody shouted, "Clara, where are you?" your car horn honked. I'm assuming your car has voice recognition software. Also that your car has the personality of a puppy. Work with me here.

Even scarier: Any time somebody shouted, "Clara, open the trunk," your car trunk unlocked.

That's what happens when you name your kernel objects. Anybody who knows the name (and has appropriate access) can open the object and start doing things to it. Presumably that's why you named your kernel object in the first place: You *want* this to happen. You gave your object a name specifically to allow other people to come in and access the same object.

In the above example, I saw that the event had a very generic-sounding name, *MyEvent*. That sounds like the name that some other similarly uncreative application developer might have chosen.

And indeed, that was the reason. There was another application which was creating an event that coincidentally has the same name, so instead of creating a new object, the kernel returned a handle to the existing one. The other application called `WaitForSingleObject` on the event, and so when the customer's program called `SetEvent`, it woke the other application instead. So this bug has a double-whammy: Not only does it cause your program to miss a signal, it causes the other program to receive a signal *when it wasn't expecting one*. Two bugs for the price of one.

Note that no matter how clever you are at choosing a name for your event, you will always have this problem, because even if you called it *SuperSecretNeverGonnaFindIt75*, there's a program out there that knows the secret name: Namely your own program! If you run two copies of your program, they will both be manipulating the same *SuperSecretNeverGonna-FindIt75*, and then you're back where you started. When the first copy of the program calls `SetEvent`, it may wake up the second copy.

(This is the same principle behind the conclusion that a single-instance program is its own denial of service.)

Kernel objects should not be named unless you intend them to be shared, because once you name them, you open yourself to issues like this. If you name a kernel object, it must be because you *want* another process to access it, not because you think giving it a name is kind of cute.

I suspect a lot of people give their kernel objects names not because they intend them to be shared, but because they see that the `CreateEvent` function has a `lpName` parameter, and they think, "Well, I guess giving it a name would be nice. Maybe I can use it for debugging purposes or something," not realizing that giving it name actually *introduced* a bug. Another possibility is that they see that there is a `lpName` parameter and think, "Gosh, I *must* give this event a name."

Kernel object names are optional. Don't give them a name unless you intend them to be shared.

Raymond Chen

Follow