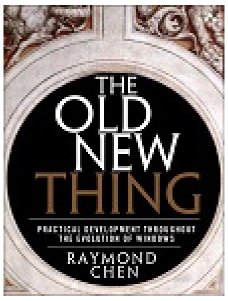
How can I wait until a named object (say a mutex) is created?

devblogs.microsoft.com/oldnewthing/20250724-00/?p=111406

July 24, 2025



A customer used a named mutex as a way to detect that another instance of the program is already running. This is pretty standard.

They also used the presence of the mutex to indicate that the program is ready to receive work requests: When the program creates the mutex, this prevents new instances from running, and it also announces that the program is open for business.

The customer had a manager program that launched this program and wants the manager to wait until the mutex is created before it starts submitting work. They wanted to know if there is a way (other than polling) to wait for a mutex to be created.

No, there is no general way to receive a notification when a named kernel object is created.

One idea is to use a named manual-reset event (initially unsignaled), and have the program signal the event when it is ready to accept work. The manager program waits on that event. The program resets the event when it shuts down to indicate that it cannot accept work.

Unfortunately, this solution doesn't work if the program crashes before it can reset the event. Handles are automatically closed when a program crashes. Closing a mutex handle implicitly releases it (marking the mutex as abandoned) but closing an event does not reset it. This means that the next time the manager launches the program, it will think that the program is ready, even though the program hasn't even started.

I tried to come up with a solution for this, since Windows doesn't have "auto-set events".

One idea was to use a named shared memory block (protected by a named mutex) that contains the worker program's PID and creation time (because the PID and creation time uniquely identify a program on a system), as well as the name of an event that the manager wants the worker program to set. The manager program opens (or creates if necessary) the shared memory block and checks the PID/time. If not valid, then the previous worker program exited, so the manager generates a new event name and writes it to the shared memory block, then launches the worker program and waits for that event to be set (or for the worker program to exit). If the PID/time is valid, then the worker is still running, and it can just wait on the event.

I also considered using an opportunistic lock as a signal: The manager program opens a file with an opportunistic lock, and the worker program opens that file for writing when it's ready. Opening for writing breaks the lock.

But then I realized that Windows already has an entire infrastructure for "launching a program and waiting for it to be ready": COM local servers.

You can `CoCreateInstance` a COM local server, and COM will launch the server process and wait for it to finish initializing and call `CoRegisterClassObject`, at which point it will use that class object to obtain an instance and return it. Somebody else has already done the work of ensuring that only one copy is running and waiting for it to be ready. You can use the returned COM object to communicate with the server program, and you can release the COM object to tell the server that you don't need it any more.

[1] Note that this overloading of the mutex is already a problem, because it means that the program cannot detect whether it should run until it is ready to run. The rule that the mutex cannot be created until the program is ready means that if two copies start at the same time, both of them will think that they are "the one" and will prepare to receive work. Whoever becomes ready first will claim the mutex, and the other will realize, at the end of all its hard work, that it is no longer needed.