COM asynchronous interfaces, part 6: Learning about completion without polling

devblogs.microsoft.com/oldnewthing/20220221-42

February 21, 2022

Raymond Chen

So far, the way we've been waiting for an asynchronous call to complete is by polling for it. Is there a way to be notified directly when the operation completes? For example, while doing work, we may call out to other methods, and if the asynchronous call completes while those other methods are busy, we'd like to cancel those other calls so we can get back to the main work of dealing with the asynchronous call.

One way to do this is to peek at the kernel event handle that is hiding inside the `ISynchronize` interface. The `ISynchronizeHandle` interface lets you do that.

```
HANDLE rawEventHandle = nullptr;
call.as<::ISynchronizeHandle>()->GetHandle(&rawEventHandle);
```

Note that the handle that comes back is still owned by the call, so don't close it. This is not usually a problem because you typically keep the call around since you will want to get the result of the asynchronous call, For example, you might want to suspend the current coroutine until the asynchronous call has completed.

```
co_await winrt::resume_on_signal(rawEventHandle);
// coroutine resumes when the asynchronous call completes

call.Finish_Something();
```

Or you might initiate multiple asynchronous calls, and you want to process the results from whichever call finishes first.

```
HANDLE readyEvents[2];

call1.as<::ISynchronizeHandle>()->GetHandle(&readyEvents[0]);
call2.as<::ISynchronizeHandle>()->GetHandle(&readyEvents[1]);

DWORD index;
auto hr = CoWaitForMultipleHandles(COWAIT_DEFAULT, INFINITE,
                                   2, readyEvents, &index);

if (hr == S_OK) {
  if (index == 0) { /* deal with call1 */ }
  if (index == 1) { /* deal with call2 */ }
}
```

If you want to use the handle in a way unrelated to the call it came from, then duplicate the handle, at which point you become responsible for the lifetime of the duplicate.

```
winrt:handle eventHandle;
winrt::check_bool(DuplicateHandle(
  GetCurrentProcess(), rawEventHandle,
  GetCurrentProcess(), eventHandle.put(),
  SYNCHRONIZE, FALSE, 0));
```

The duplicate handle `eventHandle` is your responsibility to close. You can hand it to some other component which uses it to know when the asynchronous method call has completed. Just remember to close it when you're done.

Next time, we'll look at a way of getting called back *directly* when the asynchronous call completes.

Raymond Chen

Follow