How can I emulate the REG_NOTIFY_THREAD_AGNOSTIC flag on systems that don't support it? part 4

devblogs.microsoft.com/oldnewthing/20201224-00

December 24, 2020

Raymond Chen

We continue our exercise of emulating the `REG_NOTIFY_THREAD_AGNOSTIC` flag by making the whole thing a coroutine, assuming you're willing to take the anachronism to an extreme by using C++20 features in code intended to run on Windows XP.

```
auto RegNotifyChangeKeyValueAsync(
  HKEY hkey,
  BOOL bWatchSubtree,
  DWORD dwNotifyFilter,
  HANDLE hEvent)
{
  struct awaiter
  {
    HKEY m_hkey;
    BOOL m_bWatchSubtree;
    DWORD m_dwNotifyFilter;
    HANDLE m_hEvent;
    LONG m_result;
    std::experimental::coroutine_handle<> m_handle;

    bool await_ready() const noexcept { return false; }

    bool await_suspend(std::experimental::coroutine_handle<> handle)
    {
      m_handle = handle;
      if (!QueueUserWorkItem(
          Callback,
          this,
          WT_EXECUTEINPERSISTENTTHREAD)) {
        m_result = static_cast<LONG>(GetLastError());
        return false;
      }
      return true;
    }

    LONG await_ready() const noexcept { return m_result; }

    DWORD CALLBACK Callback(void* param)
    {
      auto self = reinterpret_cast<awaiter*>(param);
      self->m_result = RegNotifyChangeKeyValueArgs(
        self->m_hkey,
        self->m_bWatchSubtree,
        self->m_dwNotifyFilter,
        self->m_hEvent,
        TRUE);
      self->m_handle();
      return 0;
    }
  };

  return awaiter(hkey, bWatchSubtree, dwNotifyFilter, hEvent);
}
```

The catch here is that the coroutine continues on the persistent thread, and you're not supposed to run long operations on the persistent thread, so the caller should probably `resume_background` to get onto a non-persistent thread pool thread.

We can't do the work ourselves of resuming on a non-persistent thread pool thread, say, by doing another `QueueUserWorkItem`, because if the second call fails, we are stuck on the persistent thread. If we are willing to bump the minimum system requirements to Windows Vista, we could preallocate the work items and remove the possibility of getting stuck halfway through.

So let's go all the way with this absurd exercise, next time.

Raymond Chen

Follow