How can I emulate the REG_NOTIFY_THREAD_AGNOSTIC flag on systems that don't support it? part 2

devblogs.microsoft.com/oldnewthing/20201222-00

December 22, 2020

Raymond Chen

We continue our exercise of emulating the `REG_NOTIFY_THREAD_AGNOSTIC` flag, this time for the case where we don't have a readily-available persistent thread. In that case, we can use one from the thread pool, but it'll be a bit trickier because we'll have to wait for the thread pool work item to run in order to get the result.

```
struct RegNotifyChangeKeyValueAsyncArgs
{
  HKEY hkey;
  BOOL bWatchSubtree;
  DWORD dwNotifyFilter;
  HANDLE hEvent;
  LONG result;
  HANDLE hComplete;

  ~RegNotifyChangeKeyValueAsyncArgs()
  {
    if (hComplete) CloseHandle(hComplete);
  }
};

DWORD CALLBACK RegNotifyChangeKeyValueOnPersistentThread(
    void* param)
{
  auto args = reinterpret_cast<
              RegNotifyChangeKeyValueAsyncArgs*>(param);
  args->result = RegNotifyChangeKeyValue(
    args->hkey,
    args->bWatchSubtree,
    args->dwNotifyFilter,
    args->hEvent,
    TRUE);
  SetEvent(args->hComplete);
  return 0;
}

// See discussion before using this function.
LONG RegNotifyChangeKeyValueAsync(
  HKEY hkey,
  BOOL bWatchSubtree,
  DWORD dwNotifyFilter,
  HANDLE hEvent)
{
  RegNotifyChangeKeyValueAsyncArgs args =
    { hkey, bWatchSubtree, dwNotifyFilter, hEvent,
      ERROR_INVALID_PARAMETER,
      CreateEvent(nullptr, TRUE, FALSE, nullptr) };
  if (!args.hComplete) {
    return static_cast<LONG>(GetLastError());
  }
  if (!QueueUserWorkItem(
          RegNotifyChangeKeyValueOnPersistentThread,
          &args,
          WT_EXECUTEINPERSISTENTTHREAD)) {
    return static_cast<LONG>(GetLastError());
  }
  WaitForSingleObject(args->hComplete, INFINITE);
```

```
  return args.result;
}
```

One of the tricks of this exercise is limiting the solution to the features that were available in Windows XP.

Since we need to do the registration from a thread that will remain running indefinitely, we ask the thread pool for help by calling `QueueUserWorkItem` with the `WT_EXECUTEIN-PERSISTENTTHREAD` flag. The callback runs on a persistent thread, and it is there that we call `RegNotifyChangeKeyValue` and pass the result back. Meanwhile, the main thread waits for the work item to finish so it can propagate the results to the caller.

This design seems to work, but it has a trap: If the caller of `RegNotifyChangeKeyValue-Async` is itself on the thread pool, then you end up with the risk of starving the thread pool and preventing it from doing the thing that would relieve the starvation.

We'll look at ways of fixing this next time.

Raymond Chen

Follow