How can I emulate the REG_NOTIFY_THREAD_AGNOSTIC flag on systems that don't support it? part 3

devblogs.microsoft.com/oldnewthing/20201223-00

December 23, 2020

Raymond Chen

We continue our exercise of emulating the `REG_NOTIFY_THREAD_AGNOSTIC` flag, this time trying to address the problem we discovered where our proposed solution can end up starving the thread pool due to its own misbehavior.

What you have to do to fix this problem is make the registration asynchronous, allowing the caller to continue and release the thread pool thread. If the registration turns out to be invalid, you can set the event to let the caller know that the registration failed. You communicate the failure through a secondary output parameter.

```
struct RegNotifyChangeKeyValueAsyncArgs
{
  HKEY hkey;
  BOOL bWatchSubtree;
  DWORD dwNotifyFilter;
  HANDLE hEvent;
  LONG* registrationResult;

  ~RegNotifyChangeKeyValueAsyncArgs()
  {
    if (hkey) RegCloseKey(hkey);
  }
};

DWORD CALLBACK RegNotifyChangeKeyValueOnPersistentThread(
    void* param)
{
  auto args = std::unique_ptr<RegNotifyChangeKeyValueAsyncArgs>(
              reinterpret_cast<
               RegNotifyChangeKeyValueAsyncArgs*>(param));
  LONG result = RegNotifyChangeKeyValue(
    args->hkey,
    args->bWatchSubtree,
    args->dwNotifyFilter,
    args->hEvent,
    TRUE);
  *args->registrationResult = result;
  if (result != ERROR_SUCCESS) SetEvent(args->hEvent);
  return 0;
}

void RegNotifyChangeKeyValueAsync(
  HKEY hkey,
  BOOL bWatchSubtree,
  DWORD dwNotifyFilter,
  HANDLE hEvent,
  LONG* registrationResult)
{
  LONG result;
  auto args = std::unique_ptr<RegNotifyChangeKeyValueAsyncArgs>(
    new(std::nothrow) RegNotifyChangeKeyValueAsyncArgs{
      nullptr, bWatchSubtree, dwNotifyFilter, hEvent,
      registrationResult });
  if (!args) {
    result = ERROR_OUT_OF_MEMORY;
  } else {
    result = RegOpenKeyEx(hkey, nullptr, 0,
                    KEY_NOTIFY, &args->hkey);
    if (result == ERROR_SUCCESS) {
      if (QueueUserWorkItem(
          RegNotifyChangeKeyValueOnPersistentThread,
          args.get(),
```

```
          WT_EXECUTEINPERSISTENTTHREAD)) {
        args.release();
        result = ERROR_SUCCESS;
      } else {
        result = static_cast<LONG>(GetLastError());
      }
    }
  }
  *registrationResult = result;
  if (result != ERROR_SUCCESS) SetEvent(hEvent);
}
```

What we do is record the results in the `registrationResult` pointer, and if the registration failed, we also signal the event to notify the caller that something is ready for inspection.

The caller waits on the event and checks the registration result. If the result is an error code, then it means that the registration was unsuccessful. If the result is `ERROR_SUCCESS`, then the registration succeeded, and the signal means that the key changed.

But now that we're doing things asynchronously, we can try to go all the way, perhaps anachronistically. Next time.

Raymond Chen

Follow