The GetCurrentThread function is like a check payable to Bearer: What it means depends on who's holding it

devblogs.microsoft.com/oldnewthing/20141015-00

October 15, 2014

Raymond Chen

The `GetCurrentThread` function returns a pseudo-handle to the current thread. The documentation goes into significant detail on what this means, but I fear that it may have fallen into the trap of providing so much documentation that people decide to skip it.

Okay, so first of all, what is a pseudo-handle? a pseudo-handle is a sentinel value for `HANDLE` that is not really a handle, but it can act like one.

The pseudo-handle returned by `GetCurrentThread` means, "The thread that this code is running on." Note that *this is a context-sensitive proposition*. All the text in MSDN is explaining the consequences of that sentence.

In a sense, the `GetCurrentThread` function is like a GPS: It tells you where you *are*, not where you *were*.

Imagine somebody who had never seen a GPS before. When they park their car in a parking garage, they look at the parking level number printed on the wall and write it down. Then when somebody asks, "Where did you park your car?" they can look in their notebook and say, "Oh, it's on level 3." Some parking garages even have little pads of paper with the parking level pre-printed on it. When you park your car, you tear a piece of paper off the pad and tuck it into your purse or wallet. Then when you want to return to your car, you see the slip of paper, and it tells you where you parked.

Now suppose you hand that person a GPS device, and tell them, "This will tell you your current location."

"Great! My current location is level 3 of the parking garage."

They go out shopping or whatever they were planning on doing, and now it's time to go home. They can't remember where they parked, so they look at the device, and it says, "Your current location is 512 Main Street."

"Hey, you told me this told me where I parked my car."

“No, I said that it told your current location.”

“That’s right. And at the time you gave it to me, my current location was level 3 of the parking garage. So I expect it to say ‘Level 3 of the parking garage’.”

“No, I mean it tells you your current location at the time you look at it.”

“Well, that’s stupid. I can do that too.” (Scribble.) “There, it’s a piece of paper that says ‘You are right here’.”

The value returned by the `GetCurrentThread` function is like the GPS, or the slip of paper that says “You are right here.” When you hand that value to the kernel, it substitutes the current thread at the time you use it. It’s like a check payable to Bearer: The money goes to whoever happens to take it to the bank.

This means, for example, that if you call `GetCurrentThread` from one thread and pass the result to another thread, then when that other thread uses the value, the kernel will interpret to mean the thread that is using it, not the thread that called `GetCurrentThread`.

It also means that the value cannot be meaningfully cached to remember the thread that created the value, because the point of `GetCurrentThread` is to adapt to whoever happens to be using it.

The theory behind `GetCurrentThread` and its friend `GetCurrentProcess` is that it gives you a convenient way to refer to the current thread from the current thread or the current process from the current process. For example, you might pass the return value of `GetCurrentProcess` to `DuplicateHandle` in order to duplicate a handle into or out of the current process. Or you might pass the return value of `GetCurrentThread` to `SetThreadPriority` to change the priority of the current thread. The “current thread” pseudo-handle let you simplify this:

```
BOOL SetCurrentThreadPriority(int nPriority)
{
 BOOL fSuccess = FALSE;
 HANDLE hCurrentThread = OpenThread(THREAD_SET_INFORMATION,
                            FALSE, GetCurrentThreadId());
 if (hCurrentThread)
 {
  fSuccess = SetThreadPriority(hCurrentThread, nPriority);
  CloseHandle(hCurrentThread);
}
```

to

```
BOOL SetCurrentThreadPriority(int nPriority)
{
 return SetThreadPriority(GetCurrentThread(), nPriority);
}
```

If you want to convert a pseudo-handle to a real handle, you can use the `DuplicateHandle` function.

```
BOOL ConvertToRealHandle(HANDLE h,
                         BOOL bInheritHandle,
                         HANDLE *phConverted)
{
 return DuplicateHandle(GetCurrentProcess(), h,
                        GetCurrentProcess(), phConverted,
                        0, bInheritHandle, DUPLICATE_SAME_ACCESS);
}
BOOL GetCurrentThreadHandle(BOOL bInheritHandle, HANDLE *phThread)
{
 return ConvertToRealHandle(GetCurrentThread(), bInheritHandle, phThread);
}
BOOL GetCurrentProcessHandle(BOOL bInheritHandle, HANDLE *phProcess)
{
 return ConvertToRealHandle(GetCurrentProcess(), bInheritHandle, phProcess);
}
```

Armed with your knowledge of pseudo-handles, criticize the following code:

```
class CSingleThreadedObject
{
public:
 CSingleThreadedObject() : _hThreadCreated(GetCurrentThread()) { }
 bool OnCorrectThread() { return GetCurrentThread() == _hThreadCreated; }
private:
 HANDLE _hThreadCreated;
};
class CFoo : protected CSingleThreadedObject
{
public:
 CFoo() { ... }
 // Every method of CFoo checks whether it is being called on the
 // same thread that it was created from.
 bool Snap()
 {
  if (!OnCorrectThread()) return false; // multithreading not supported
  ...
 }
};
```

Follow-up exercise: Criticize the following code that attempts to address the issues you raised in the previous exercise.

```
// ignore error handling in this class for the purpose of the exercise
class CSingleThreadedObject
{
public:
 CSingleThreadedObject() {
  ConvertToRealHandle(GetCurrentThread(), FALSE, &_hThreadCreated)
}
 bool OnCorrectThread() {
  HANDLE hThreadCurrent;
  ConvertToRealHandle(GetCurrentThread(), FALSE, &hThreadCurrent);
  bool onCorrectThread = hThreadCurrent == _hThreadCreated;
  CloseHandle(hThreadCurrent);
  return onCorrectThread;
 }
private:
 HANDLE _hThreadCreated;
};
```

Raymond Chen

Follow