My, those threads start up really fast nowadays

devblogs.microsoft.com/oldnewthing/20131025-00

October 25, 2013

Raymond Chen

Here's a little puzzle inspired by an actual bug:

```
// global variable
DWORD g_WorkerThreadId;
bool IsRunningOnWorkerThread()
{
  return GetCurrentThreadId() == g_WorkerThreadId;
}
bool LaunchWorkerThread()
{
 HANDLE hThread = CreateThread(nullptr, 0,
                               WorkerThread,
                               nullptr, 0,
                               &g_WorkerThreadId);
 if (hThread != nullptr) {
   CloseHandle(hThread);
   return true;
 }
 return false;
}
DWORD CALLBACK WorkerThread(void *Proc)
{
  // Can this assertion ever fire?
  assert(IsRunningOnWorkerThread());
  return 0;
}
```

Can the assertion at the start of `WorkerThread` ever fire?

Naturally, the answer is *Yes*, otherwise it wouldn't be a very interesting article.

The assertion can fire if the worker thread starts running *before the call the `CreateThread` returns*. In that case, the caller hasn't yet received the handle or ID of the newly-started thread. The new thread calls `IsRunningOnWorkerThread`, which returns `false` since `g_WorkerThreadId` hasn't been initialized yet.

The actual bug was something along the lines of this:

```
void DoSomething()
{
  if (IsRunningOnWorkerThread()) {
     .. do it one way ..
  } else {
     .. do it the other way ..
  }
}
void DoManyThings()
{
  DoSomething();
  DoSomethingElse();
  DoYetAnotherThing();
}
DWORD CALLBACK WorkerThread(void *Proc)
{
  ...
  DoManyThings();
  ...
  return 0;
}
```

If the new thread started up so quickly that the original thread doesn't get a chance to receive the new thread ID and put it into `g_WorkerThreadID`, then the `DoSomething` function called from the worker thread will accidentally do things the not-on-the-worker-thread way, and then things start go go awry.

One way to address is is to add suspenders to your belt:

```
DWORD CALLBACK WorkerThread(void *Proc)
{
  g_WorkerThreadId = GetCurrentThreadId();
  ...
```

By having both the original thread and the created thread set the `g_WorkerThreadId` variable, you cover both cases of the race. If the original thread runs faster, then the `CreateThread` function will set the `g_WorkerThreadId` variable to the ID of the worker thread, and the first line of `WorkerThread` will be redundant. On the other hand, if the worker thread runs faster, then the assignment at the beginning of `WorkerThread` sets the thread ID, and the assignment performed by the `CreateThread` function will be redundant.

Raymond Chen

Follow