When you think you found a problem with a function, make sure you're actually calling the function

devblogs.microsoft.com/oldnewthing/20150617-00

June 17, 2015

Raymond Chen

On an internal mailing list, a tester asked if there were any known problems with the `FindFirstFileEx` function preventing a directory from being deleted and recreated.

> Our code creates a test folder, then calls `FindFirstFileEx` to look inside the test folder. When we're done, we call `FindClose`, then delete the directory. When we try running the test twice, the second time fails to create the test folder; we get `ERROR_ACCESS_DENIED`. But if we switch to `FindFirstFile` instead of `FindFirstFileEx`, then everything works as expected.
>
> Here's our code, simplified.

```
// Assume all functions succeed except where indicated.

CreateDirectory(L"C:\\Test", NULL);

// This version works:
//
// WIN32_FIND_DATA data;
// HANDLE hFindFile = FindFirstFile(L"C:\\Test\\*", &data);

// This version doesn't:
//
WIN32_FIND_DATA data;
HANDLE hFindFile = FindFirstFileEx(L"C:\\Test\\*",
                                   FileExInfoBasic,
                                   &data,
                                   FindExSearchNameMatch,
                                   NULL,
                                   0);
FindClose(hFindFile);

RemoveDirectory(L"C:\\Test");

// If we used FindFirstFile, then this CreateDirectory succeeds.
// If we used FindFirstFileEx, then this CreateDirectory fails.
CreateDirectory(L"C:\\Test", NULL);
```

I suggested that they try running their test with anti-malware software disabled. Anti-malware software will frequently intrude on file operations, and it could be that the virus scanner is still checking the old `C:\Test` directory when you get around to creating the new one. Content indexers are another case where this can happen, but content indexers tend to wait until the machine is quiet rather than introducing on actions as they occur. (Now, well-written virus scanners and content indexers know to do things like abandon a file scan when a delete request is made, or use opportunistic locks to get out of the way when an application wants to do something with a file being scanned. But not all virus scanners and content indexers as as well-written as we might like.)

We later heard back that they figured out the problem, and it wasn't because of a virus scanner or content indexing service.

The problem was that their code was running inside a test harness, and that test harness had mocked the `FindFirstFile` and `FindClose` functions, but it did not mock the `FindFirstFileEx` function. When the mock `FindClose` function was given a handle created by the *real* `FindFirstFileEx` function, it got confused and ended up leaking the directory handle. The `RemoveDirectory` function succeeded, but the directory was not fully removed due to the outstanding handle, and the attempt to recreate the directory therefore failed.

The tester also confirmed that the problem did not exist when they ran the code outside the test environment.

When you think you found a problem with a function, make sure you're actually calling the function. In this case, the code was running under nonstandard conditions: The test harness had redirected a bunch of OS functions. As a result, when the code called `FindClose`, it wasn't actually calling `FindClose` but rather a mock function provided by the test harness.

To be fair, the tester was new to the team and was likely not even aware that the test harness was mocking file I/O functions in the first place.

If you are having trouble with a function, one thing to check is that you're actually calling the function.

Raymond Chen

Follow