You just have to accept that the file system can change

devblogs.microsoft.com/oldnewthing/20071109-00

November 9, 2007

Raymond Chen

A customer who is writing some sort of code library wants to know how they should implement a function that determines whether a file exists. The usual way of doing this is by calling `GetFileAttributes`, but what they've found is that sometimes `GetFileAttributes` will report that a file exists, but when they get around to accessing the file, they get the error `ERROR_DELETE_PENDING`.

The lesser question is what `ERROR_DELETE_PENDING` means. It means that somebody opened the file with `FILE_SHARE_DELETE` sharing, meaning that they don't mind if somebody deletes the file while they have it open. If the file is indeed deleted, then it goes into "delete pending" mode, at which point the file deletion physically occurs when the last handle is closed. But while it's in the "delete pending" state, you can't do much with it. The file is in limbo.

You just have to be prepared for this sort of thing to happen. In a pre-emptively multi-tasking operating system, the file system can change at any time. If you want to prevent something from changing in the file system, you have to open a handle that denies whatever operation you want to prevent from happening. (For example, you can prevent a file from being deleted by opening it and not specifying `FILE_SHARE_DELETE` in your sharing mode.)

The customer wanted to know how their "Does the file exist?" library function should behave. Should it try to open the file to see if it is in delete-pending state? If so, what should the function return? Should it say that the file exists? That it doesn't exist? Should they have their function return one of three values (Exists, Doesn't Exist, and Is In Funky Delete State) instead of a boolean?

The answer is that any work you do to try to protect users from this weird state is not going to solve the problem because the file system can change at any time. If a program calls "Does the file exist?" and the file does exist, you will return `true`, and then during the execution of your `return` statement, your thread gets pre-empted and somebody else comes in and puts the file into the delete-pending state. Now what? Your library didn't protect the program from anything. It can still get the delete-pending error.

Trying to do something to avoid the delete-pending state doesn't accomplish anything since the file can get into that state after you returned to the caller saying "It's all clear." In one of my messages, I wrote that it's like fixing a race condition by writing

```
// check several times to try to avoid race condition where
// g_fReady is set before g_Value is set
if (g_fReady && g_fReady && g_fReady && g_fReady && g_fReady &&
    g_fReady && g_fReady && g_fReady && g_fReady && g_fReady &&
    g_fReady && g_fReady && g_fReady) { return g_Value; }
```

The compiler folks saw this message and got a good chuckle out of it. One of them facetiously suggested that they add code to the compiler to detect this coding style and not optimize it away.

Raymond Chen

Follow