If a process crashes while holding a mutex, why is its ownership magically transferred to another process?

devblogs.microsoft.com/oldnewthing/20140925-00

September 25, 2014

Raymond Chen

A customer was observing strange mutex ownership behavior. They had two processes that used a mutex to coordinate access to some shared resource. When the first process crashed while owning the mutex, they found that the second process somehow magically gained ownership of that mutex. Specifically, when the first process crashed, the second process could take the mutex, but when it released the mutex, the mutex was still not released. They discovered that in order to release the mutex, the second process had to call `ReleaseMutex` *twice*. It's as if the claim on the mutex from the crashed process was secretly transferred to the second process.

My psychic powers told me that that's not what was happening. I guessed that their code went something like this:

```
// code in italics is wrong
bool TryToTakeTheMutex()
{
 return WaitForSingleObject(TheMutex, TimeOut) == WAIT_OBJECT_0;
}
```

The code failed to understand the consequences of `WAIT_ABANDONED`.

In the case where the mutex was held by the first process when it crashed, the second process will attempt to claim the mutex, and it will succeed, and the return code from `WaitForSingleObject` will be `WAIT_ABANDONED`. Their code treated that value as a failure code rather than a modified success code.

The second program therefore claimed the mutex *without realizing it*. That is what led the customer to believe that ownership was being magically transferred to the second program. It wasn't magic. The second program misinterpreted the return code.

The second program saw that `TryToTakeTheMutex` "failed", and it went off and did something else for a while. Then the next time it called `TryToTakeTheMutex`, the function succeeded: It was a successful recursive acquisition, but the program thought it was the

initial acquisition.

The customer didn't reply back, so we never found out whether that was the actual problem, but I suspect it was.

Raymond Chen

Follow