Why can't the system hibernate just one process?

devblogs.microsoft.com/oldnewthing/20040420-00

April 20, 2004

Raymond Chen

Windows lets you hibernate the entire machine, but why can't it hibernate just one process? Record the state of the process and then resume it later.

Because there is state in the system that is not part of the process.

For example, suppose your program has taken a mutex, and then it gets process-hibernated. Oops, now that mutex is abandoned and is now up for grabs. If that mutex was protecting some state, then when the process is resumed from hibernation, it thinks it still owns the mutex and the state should therefore be safe from tampering, only to find that it **doesn't own the mutex any more** and its state is corrupted.

Imagine all the code that does something like this:

```
// assume hmtx is a mutex handle that
// protects some shared object G
WaitForSingleObject(hmtx, INFINITE);
// do stuff with G
...
// do more stuff with G on the assumption that
// G hasn't changed.
ReleaseMutex(hmtx);
```

Nobody expects that the mutex could secretly get released during the "..." (which is what would happen if the process got hibernated). That goes against everything mutexes stand for!

Consider, as another example, the case where you have a file that was opened for exclusive access. The program will happily run on the assumption that nobody can modify the file except that program. But if you process-hibernate it, then some other process can now open the file (the exclusive owner is no longer around), tamper with it, then resume the original program. The original program on resumption will see a tampered-with file and may crash or (worse) be tricked into a security vulnerability.

One alternative would be to keep all objects that belong to a process-hibernated program still open. Then you would have the problem of a file that can't be deleted because it is being held open by a program that isn't even running! (And indeed, for the resumption to be successful

across a reboot, the file would have to be re-opened upon reboot. So now you have a file that can't be deleted even after a reboot because it's being held open by a program that isn't running. Think of the amazing denial-of-service you could launch against somebody: Create and hold open a 20GB file, then hibernate the process and then delete the hibernation file. Ha-ha, you just created a permanently undeletable 20GB file.)

Now what if the hibernated program had created windows. Should the window handles still be valid while the program is hibernated? What happens if you send it a message? If the window handles should not remain valid, then what happens to broadcast messages? Are they "saved somewhere" to be replayed when the program is resumed? (And what if the broadcast message was something like "I am about to remove this USB hard drive, here is your last chance to flush your data"? The hibernated program wouldn't get a chance to flush its data. Result: Corrupted USB hard drive.)

And imagine the havoc if you could take the hibernated process and copy it to another machine, and then attempt to restore it there.

If you want some sort of "checkpoint / fast restore" functionality in your program, you'll have to write it yourself. Then you will have to deal explicitly with issues like the above. ("I want to open this file, but somebody deleted it in the meantime. What should I do?" Or "Okay, I'm about to create a checkpoint, I'd better purge all my buffers and mark all my cached data as invalid because the thing I'm caching might change while I'm in suspended animation.")

Raymond Chen

Follow