

Re: Accessing Physical Memory & Other Process's Address Space

Source: <http://coding.derkeiler.com/Archive/Assembler/comp.lang.asm.x86/2008-02/msg00084.html>

- *From:* Gil Hamilton <spamtrap@xxxxxxxxxx>
 - *Date:* Tue, 12 Feb 2008 15:24:37 +0100 (CET)
-

Tim Roberts <spamtrap@xxxxxxxxxx> wrote in
news:t1dsq3djfaatlegbihks4vbg1me1pllu5b@xxxxxxxx:

Gil Hamilton <spamtrap@xxxxxxxxxx> wrote:

I believe Windows has an equivalent mechanism though I haven't done any Windows kernel programming in a very long time.

Windows does have kernel-mode threads, but in Windows a "thread" and a "process" are very different things. In Linux, the line between those two concepts is rather fuzzy.

I don't really buy that. Would you explain what differences you see? As I see it, while the mechanisms for creating processes and threads are significantly different in the two systems, they share almost every important conceptual attribute: The process represents those things that can be shared among multiple threads: process ID, executable code, virtual address space, open file descriptors, and so forth. The thread represents an execution context and hence has a stack in user-mode representing its context in user mode (and a kernel stack representing its context there).

Creating a process in linux with `fork(2)` simply creates a process with a single unnamed thread implicitly attached to it. I think it's slightly more explicit in Windows -- IIRC, at the lowest level `NtCreateThread` must be called to create the execution context before the process created by `NtCreateProcess` can do anything -- but in the end, it's in the same thing. In fact, ordinary programs just call `CreateProcess`, which creates both process and thread. It's pretty much exactly equivalent to `fork + exec`.

A kernel-mode thread in Windows actually belongs to the system process, which is just like every other process.

Re: Accessing Physical Memory & Other Process's Address Space

Isn't this also completely analogous to a linux kernel thread? Where is the user-mode context associated with the Windows system process? Isn't it just a place holder for threads that don't have a user-mode half?

GH

.