

Re: User vs. Worker threads

Source:

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.kernel/2007-01/msg00386.htm>

- *From:* "anton bassov" <soviet_bloke@xxxxxxxxxxx>
 - *Date:* 25 Jan 2007 21:36:53 -0800
-

Kevin,

Basically, you got it right. Congratulations!!!! However, a couple of amendments are still needed.

As far as kernel concerned, there are 3 types of threads:

1. Idle threads, i.e. the ones that CPU dispatches when it has no other threads to dispatch. Every CPU has its own idle thread. Collection of idle threads is known as a System Idle Process (this is how it is known to the Task Manager – its internal name is just System, and its PID is zero). Idle threads cannot get terminated, for understandable reasons

2. Worker threads. These threads run tasks that are not directly related to user requests, and, hence, have no user-mode representation. "Lazy writer" thread and Balance Set Manager are typical examples of worker threads. Collection of worker threads is known as System Process (its PID is non-zero). Worker threads cannot be terminated by external means, although they can terminate themselves by calling `PsTerminateSystemThread()`

3. All threads that are neither worker nor idle ones are referred to as user threads. These threads have their user-mode representation, and can get terminated both externally and upon their own initiative. In former case the target thread may get terminated only if it is able to receive user-mode APCs (i.e. the thread is currently in the user mode or in the waiting state with 'UserMode' having been specified as 'WaitMode' parameter to `KeWait...` call). Otherwise, the thread will be terminated only after it makes a return to the user mode

If driver creates a thread in the kernel mode with `PsCreateSystemThread()`, this thread will be a worker one – it will have no user-mode representation, and run in context of a system process, no matter in which context `PsCreateSystemThread()` has been made. However,

Re: User vs. Worker threads

as any other thread, it just cannot "execute separate from the IRQ-driven code" – when interrupt occurs, it gets processed in context of arbitrary thread, i.e. the one that happens to run on the given CPU at the time of interrupt, no matter if this thread is worker or user or idle one....

Anton Bassov

On Jan 26, 3:25 am, "Kevin" <kevi...@xxxxxxxxxxxx> wrote:

In case it is a non-MFC question, then:

A "user thread" is a thread that happens to be executing some arbitrary thread from some arbitrary process, while in the kernel.

A "worker thread" is a thread created by device driver code in the kernel, to execute separate from the IRQ-driven code.

"William DePalo [MVP VC++]" <willd.no.s...@xxxxxxxx> wrote in message news:uF64Lu9PHHA.1248@xxxxxxxxxxxxxxxxxxxxxxxxxxxx

"e_yossi" <eyo...@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote in message news:33E2CE27-947E-431B-9476-DE87A75A94C0@xxxxxxxxxxxxxxxxxxxx

What are the differences between worker threads and user threads?

To the operating system, the threads in an `_application_` are all the same.

MFC's framework makes a distinction between threads which own windows and have message maps and which therefore spin message loops (UI threads) and threads that don't (worker threads). If you need further information on MFC you should repost in one of the MFC groups.

Re: User vs. Worker threads

Regards,
Will

www.ivrforbeginners.com– Hide quoted text — Show quoted text –