

Re: Questions regarding Synchronisation in Windows (Spinlocks)

Source:

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.kernel/2008-03/msg00154.htm>

- *From:* smawsk <sk.smawsk@xxxxxxxxxx>
 - *Date:* Wed, 12 Mar 2008 01:23:05 -0700 (PDT)
-

On Mar 10, 8:02 pm, "Alexander Grigoriev" <al...@xxxxxxxxxxxxxxxx> wrote:

1) spinlocks used to have different implementation in uniprocessor kernel. Vista/Longhorn now only ship multiprocessor kernel. Interrupts are handled in context of ISR. There is no thread switch, as they don't run in a particular thread context.

"smawsk" <sk.sma...@xxxxxxxxxx> wrote in message

news:b2a63809-b597-4a8c-a8f4-efc069b3a05b@xx

Hi,

I have some questions regarding the implementation of synchronisation techniques in Windows.

I would be grateful if some one can answer these.

1) How are Spinlocks implemented in Windows for a UNIPROCESSOR machine? Do they involve busy waiting?

Since Spinlocks execute at DPC/Dispatch level, there is no way for the thread to move to wait state.

Consider a scenario where a thread is trying to acquire a spin lock.

But an interrupt occurs which has an IRQL > DPC Dispatch level OR the IRQL of the thread that acquired the spin lock. For the new interrupt to be serviced, dispatcher has to be invoked to switch to the new thread servicing the interrupt.

Re: Questions regarding Synchronisation in Windows (Spinlocks)

What happens in this scenario?

2) What is the difference between an executive object and a kernel object? Can someone give an example for each?

I reckon Semaphore, Event, Mutex etc are executive objects. So what kernel objects are these executive objects built on?

Which category do "Dispatcher" objects belong to?

3) What is the difference between a Mutex and Fast Mutex/Guarded Mutex?

Thanks in advance.

Warm Regards.

Hi,

Thanks for your reply.

But when hardware generates an interrupt, whatever code that was being executed should be stopped. The executing context should be saved before the control is handed over to the ISR.

How different is this from a thread switch where in the context of the currently running thread is saved and code is executed in the context of a different thread?

I am trying to get my basics right.

Regarding other questions, I did some more research and found this. Could you please let me know if my understanding is correct?

The difference between a Normal Mutex and a fast mutex is that, a fast mutex does not maintain thread ownership as a normal mutex does.

For each executive object (dispatcher) there is a corresponding kernel object on which it is built.

For instance, Semaphore uses a data structure implemented by the kernel called KSEMAPHORE.

Re: Questions regarding Synchronisation in Windows (Spinlocks)

Re: Questions regarding Synchronisation in Windows (Spinlocks)

Thanks in advance.

Warm Regards.

.