

## Re: Application, dll and driver design

**Source:**

<http://www.tech-archive.net/Archive/Development/microsoft.public.win32.programmer.ui/2004-08/0212.html>

---

**From:** Raj (Raj\_at\_discussions.microsoft.com)

**Date:** 08/17/04

Date: Tue, 17 Aug 2004 05:43:03 -0700

Thank you very much for your detailed reply. I agree with you.

But there is one problem with replacing window messages with packet reading thread. I have 15 processes all talking to each other and Dll process. If I replace with packet reading thread, each of the process should have 'PRT semaphore' handle of each other. It would mean 15\*15 handles. When one of the process is exiting, all other processes should clear their handle "in their context". So cleaning up will be exhaustive race prone cases. If one of process is terminated, even bigger headache. I can implement this scheme, but should be very careful with design. Finally I came to some idea.

Each process will have handle of other process PRT handle only if they are talking. All may not talk to every other process. Even if there are 15\*15 handles, I am willing to do so if there is tangible performance benefit. Every handle also has a status variable attached to it. When a process is exiting, it will set this status state to invalid in the shared memory of these handles. It will then post a message to Dll and clears off. Dll will post a message to all other apps to clear handle in their context. So much work to replace standard interface.

Looks like there is performance difference. I will test it and let you know the results. Any comments appreciated.

"Le Chaud Lapin" wrote:

> Raj <Raj@discussions.microsoft.com> wrote in message  
news:<2F1BC142-0ED1-470F-884D-E5ACD4ECFC47@microsoft.com>...  
>> Need some advice on overall design issue.  
>>  
>> My Dll process can support upto 15 applications. These applications will  
>> receive data from remote PC on some transport like USB. My driver collects  
>> this data and using pending IRPs and IOCTL calls, dll will pull the data from  
>> driver and post 'DATA\_RECEIVED' windows message(with packet Identifier) to  
>> respective Application. Application will call dll API with packet ID and  
>> receive data. Shared memory is used IOCTL buffers and dll API copies data to  
>> App buffer from this shared buffer pool. I use 20 buffers each 10K size. One

> > 10K buffer is sent in each IOCTL call to pull as many packets as fit. All  
> > 200K belongs to shared memory at user mode memory mapped file.  
> >  
> > Every thing works fine. But I observe lack of speed when small packets are  
> > used and mega bytes of data sent. Speed difference is low as a factor of 20  
> > between 32K packet and 2K packets.  
>  
> If I understand you correctly, a device-interface-thread (DIT) in the  
> DLL is blocking on a call into the driver to get the data via IOCTL  
> whereby it copies received packets into one of the 10K buffers, then  
> posts a message to the appropriate window, after which window  
> procedure calls back into the DLL to copy the data \*again\* to the app  
> buffer.  
>  
> If so, try this:  
>  
> First, get rid of the window message passing. There is a lot of  
> machinery involved in passing message to windows, including multiples  
> calls against mutexes. Instead, put a packet-reader-thread (PRT) in  
> each app that blocks while trying to read from the shared buffer of  
> the DLL. Blocking will persist until the DIT of the DLL raises a  
> semaphore for the appropriate PRT. The PRT will then unblock and scan  
> for packets in the packet pool, process them, and clear a flag for the  
> 10K slot to say that that slot is free. However, it should be evident  
> that if you have this flag on a per-10K buffer basis, this method will  
> not work, as two different PRT's might have incoming packets residing  
> in the same 10K buffer. The solution is to make your buffers smaller,  
> much less than 10K of course, but up to the maximum size of a packet,  
> then when your DLL thread goes to read into the pool in the IOCTL,  
> specify the indexes accross the user/kernel transistion (array of  
> unsigned int might work) of the slots that are available for filling.  
> Of course, if the shared section is available at ring-0, a hard scan  
> in a tight for loop will work. The PRT blocking semaphores will need  
> names so that you can reference them accross process boundaries (if  
> you so choose). Also note that if your apps will be running as  
> services before log in, you should supply a NULL security descriptor  
> in call to CreateSemaphore. Finally, if packets cannot be processed  
> immediately after they arrive, but must be queued, you should still  
> try to avoid copying if possible, perhaps by maintaining a (protected  
> by mutex) queue of pointers to packet in each app, the pointers  
> pointing to packets that have been "acknowledge for later processing".  
> If you detect that this is not feasible (look up Little's Theorem on  
> Internet to determine whether it is or not based on expected receive  
> rate and expected time for processing), then you should increase the  
> size of your packet pool, adding a bit for expected deviation, to  
> avoid overflow. If this is not possible, then go ahead and do the copy  
> to a separate app buffer, but use memcpy or inline assembly code to  
> effect the copy - they are much faster than a run-of-the-mill for  
> loop.  
>  
> Doing these things will not eliminate the context switching overhead,

microsoft.public.win32.programmer.ui: Re: Application, dll and driver design

- > *but it will reduce the amount of the system spends working with thread*
- > *queues.*
- >
- > *–Chaud Lapin–*
- >