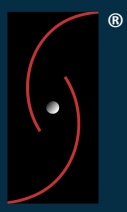
ISSUE VOLUME
03/21
NOV– DEC
2015

The NT Insider

A publication of OSR Open Systems Resources, Inc.

```
typedef struct _NT_INSIDER_CONTEXT {

    WDFDEVICE           WdfDevice;
    WDFINTERRUPT        WdfInterrupt;

    //
    // Wanted: Even Better Communication
    //
    PONTIFICATION       PeterPontificates;

    //
    // Lock used to protect articles
    //
    _Has_lock_kind_(_Lock_kind_spin_lock_)
    WDFSPINLOCK         ArticleSpinLock;

    //
    // You Just THINK it Works:
    // The 3 Most Common KMDF Errors in Shipping Drivers
    //
    _Guarded_by_(ArticleSpinLock)
    INSIDER_ARTICLE     ThreeMostCommonErrors;

    //
    // There's Gotta Be a Better Way:
    // Clarifying Critical Regions
    //
    _Guarded_by_(ArticleSpinLock)
    INSIDER_ARTICLE     CriticalRegions;

    //
    // An Embarrassment of Riches:
    // Visual Studio Customizations & Extensions for Driver Devs
    //
    _Guarded_by_(ArticleSpinLock)
    INSIDER_ARTICLE     VSCustomizations;

    //
    // Guest Article: Choosing The Right Synchronization Mechanism
    // Alex Ionescu on The State of Synchronization
    //
    GUEST_ARTICLE       StateOfSynchronization;

    //
    // General Info
    //
    ULONG               Issue = 21;   // Yes, this does compile
    ULONG               Volume = 3;   // No, we don't know what it does

} NT_INSIDER_CONTEXT, *PNT_INSIDER_CONTEXT;

WDF_DECLARE_CONTEXT_TYPE_WITH_NAME(NT_INSIDER_CONTEXT, NTIGetContextFromHandle)
```



And More:

- Training At OSR
- Threshold 2
- Seminars in Europe!!

Published by
OSR Open Systems Resources, Inc.
105 Route 101A, Suite 19
Amherst, New Hampshire USA 03031
(v) +1.603.595.6500
(f) +1.603.595.6503

http://www.osr.com

Consulting Partners
W. Anthony Mason
Peter G. Viscarola

Executive Editor
Daniel D. Root

Contributing Editors
Scott J. Noone
OSR Associate Staff

Send Stuff To Us:
NtInsider@osr.com

Single Issue Price: $15.00

The NT Insider is Copyright ©2015 All rights reserved. No part of this work may be reproduced or used in any form or by any means without the written permission of OSR Open Systems Resources, Inc.

We welcome both comments and unsolicited manuscripts from our readers. We reserve the right to edit anything submitted, and publish it at our exclusive option.

Stuff Our Lawyers Make Us Say
All trademarks mentioned in this publication are the property of their respective owners. "OSR", "OSR Online" and the OSR corporate logo are trademarks or registered trademarks of OSR Open Systems Resources, Inc.

We really try very hard to be sure that the information we publish in *The NT Insider* is accurate. Sometimes we may screw up. We'll appreciate it if you call this to our attention, if you do it gently.

OSR expressly disclaims any warranty for the material presented herein. This material is presented "as is" without warranty of any kind, either expressed or implied, including, without limitation, the implied warranties of merchantability or fitness for a particular purpose. The entire risk arising from the use of this material remains with you. OSR's entire liability and your exclusive remedy shall not exceed the price paid for this material. In no event shall OSR or its suppliers be liable for any damages whatsoever.

It is the official policy of OSR Open Systems Resources, Inc. to safeguard and protect as its own, the confidential and proprietary information of its clients, partners, and others. OSR will not knowingly divulge trade secret or proprietary information of any party without prior written permission. All information contained in *The NT Insider* has been learned or deduced from public sources...often using a lot of sweat and sometimes even a good deal of ingenuity.

OSR is fortunate to have customer and partner relations that include many of the world's leading high-tech organizations. As a result, OSR may have a material connection with organizations whose products or services are discussed, reviewed, or endorsed in *The NT Insider*.

Neither OSR nor *The NT Insider* is in any way endorsed by Microsoft Corporation. And we like it that way, thank you very much.

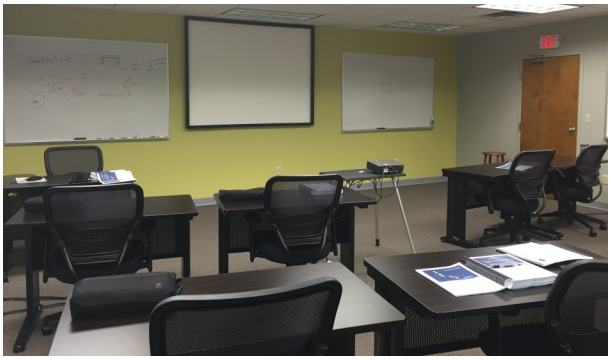
Training At OSR
Inaugural Presentation a Success!

OSR's Dedicated Training Facility

OSR is pleased to announce the success of the inaugural use of the OSR Training facility, co-located with OSR's New Hampshire headquarters.

In September, after months of preparation, OSR was finally able to open its doors and host a presentation—in this case, of its popular seminar: *Windows Internals & Software Drivers*.

"The beauty of this space is that it gives us the flexibility to control all aspects of the training venue", says OSR consulting partner and lead instructor, Peter Viscarola. "We know we can deliver definitive technical value in our seminars, but that's really only half the battle when it comes to providing a positive seminar experience for our customers."



A Comfortable & Professional Environment

Always Plenty of Food for Students

With comfortable seating for up to 12 students at individual workstations, and a separate café/kitchen area, the OSR staff is pleased with the response from the inaugural group of attendees. "Naturally, we were anxious to hear feedback—that's typically my job post-seminar, but our instructors couldn't help themselves from polling students all week, with all positive results", says OSR's Debra Stitt, who manages OSR's seminar business.

Beyond the flexibility gained in scheduling open-enrollment seminars on a more frequent basis, OSR plans to expand the use of the training facility. "The space allows us to consider hosting private training functions, "developer-in-residence" engagements, or even impromptu guest presentations on specialized topics", says Dan Root, OSR's general manager.

For more information on OSR's schedule of open enrollment offerings, opportunities for private seminars, or any OSR training-related enquiry, please contact Ms. Debra Stitt or a seminar representative at seminars@osr.com.

The OSR Café

Get Social with OSR
Real -Time Updates

Follow us!

Just in case you're not already following us on Twitter, Facebook, LinkedIn, or via our own "**osrhints**" distribution list, below are a few of the more recent contributions that are getting attention in the Windows driver development community:

Fun with WinDBG—NatVis Support
A quick intro to NatVis support with the Win10 WDK and VS.
http://www.osr.com/blog/2015/11/06/fun-windbg-natvis-support/

Some Things are Best Left to Experts. Really.
"Cowboying" a driver into existence just isn't a long-term solution for success.
http://www.osr.com/blog/2015/10/30/somethings-best-left-experts-really/

Right-Click Your DML!
Enhanced Debugger Markup Language (DML) support in the Win10 build of WinDbg.
http://www.osr.com/blog/2015/08/18/right-click-dml/

Become More Knowledgeable… Instantly!

We email our friends when we've got something interesting to say. Join the list! Send a blank email to join-osrhints@lists.osr.com and we'll add you to the list. We don't have THAT much to say. You'll probably get one or two emails a month.

What's New in Threshold 2?
As Always: We've Got You Covered

As we were writing this issue, Threshold 2, also known as The Windows 10 Fall Update or Windows 10 Version 1511 was pending release. How does TH2 affect driver devs? While we don't expect big changes, we'll update you on what's new just as soon as TH2 is released.

We'll post an update in our Developer's Blog describing the changes, check it on TH2 RTM. Or, better yet, join the OSR Hints list (see above for how) and we'll email you to let you know as soon as we've written something.

OSR Seminars Coming to Europe in 2016!
Tell us WHERE and WHAT You Want Us To Teach

In 2015, OSR brought our seminars to Asia, teaching developers in India and Singapore. By all accounts, they were a great success!

In 2016 our focus will be on bringing our seminars to Europe. BUT, we have a big favor to ask. We need your help to determine where and what we should be teaching. If you're interested in the possibility of attending an OSR seminar in Europe, please help us understand

Take our poll at
http://www.osr.com/seminars-europe/

 © OSR Open Systems Resources, Inc.

Peter Pontificates
Wanted: Even Better Communication

I have pontificated before, and at some length, about the fact that Microsoft has returned to its former position of actively **wanting** to engage with the 3rd party driver development community. This attitude is as commendable as it is welcomed.

What's most encouraging is that MSFT folks appear to be listening to the various feedback channels, and taking action on the feedback they receive. Take for example the issue with a recent Windows 7 update: the symbols for the update were published to the MSFT symbol server fully stripped and without the symbols for the data types added back in to them. This made debugging with those symbols (cough cough) "very difficult." A community member sent email to the debugger feedback alias (did you even know there was a debugger feedback alias? I didn't), but apparently didn't get a reply. So, when I heard about this problem, I made a nuisance of myself and asked a buddy of mine who happens to work in a relevant team at Microsoft if he knew what was up. I expected to have to explain the problem to him in detail, and ask for a favor to get the problem fixed. But much to my surprise, I discovered the relevant folks at MSFT already knew about the problem and were already working on a fix (that entailed a ton of manual work, unfortunately). And, guess what? A couple of weeks later, the fix was implemented and the updated symbols were on the symbol server. Yay!

Yes, that's just one small example, about a single item. But there are examples that tour concerns are being heard on much larger issues as well. For example, in talking with the WDK program managers I've learned that there's an ongoing demand for better and smoother integration for all facets of development with the WDK within Visual Studio. This includes better integration of the debugger and test system setup. I've also heard that community feedback regarding the difficulty in using some of these features lead the WDK team to make significant changes in the Windows 10 WDK in how test system setup is implemented. Now, you can't prove any of that by me, personally… because I **still** can't get test system setup or driver deployment to work from within VS. Not that I want to, to be honest. But now, apparently, some people actually **can** get this to work. And they can even get it to work with some of the IoT-based platforms, which is pretty cool and even potentially useful.

But before you start singing Kumbaya and propose a big group hug, there's much that needs improved. For example, **how does the community know** – as a whole – that our feedback is valued, is being heard, and is being considered for action (or even has been considered and rejected)? A quick look at my examples above demonstrates one of the current mechanisms. The only way the community knew that the fix for the stripped Windows 7 symbols was on the way was because I heard about the problem, asked a buddy of mine about it, and when he gave me the scoop I posted something to our WinDbg forum. And I happen to know about the whole issue of VS integration and deploy, because – well, to be honest – I've been complaining about this feature for years and I happen to know some of the right folks to complain to. And I've told you about it here.

As the complexity of the WDK increases, as the number of platforms supported by Windows continues to multiply (RPI 2 with Win10 IoT Core, anyone?), and as the integration between the WDK and Visual Studio gets tighter, the chances significantly increase that one of those complex, tightly integrated, things will go wrong on at least one of those platforms. That means that the community's feedback is even more crucial than ever. And it means that sharing that feedback, and MSFT's response to that feedback, broadly among the driver development community becomes vital. This prevents the world over from discovering various problems individually and wasting time struggling with them.

Now, to be fair, there **are** numerous feedback mechanisms that the community uses today to "get the message" to the right folks at MSFT. I mentioned the WinDbg feedback alias, above, as one example. Another similar method is the link at the bottom of every WDK documentation page that says "Send comments about this topic to Microsoft." That feedback link actually works. Or, at least, it certainly has in the past.

WDK Developer Support has historically been one of the most common, and effective, ways of providing feedback on the driver kit. Staffed by a group of folks that are, in my experience, both knowledgeable and helpful, when you have what seems to be a WDK problem contacting WDK Developer Support remains one of the best options for getting your issue handled. The only problem is, you contacting developer support helps **you**, but doesn't really do anything to let others in the community know about your experience.

(CONTINUED ON PAGE 5)

 © OSR Open Systems Resources, Inc.

Peter Pontificates... (Cont.)

(CONTINUED FROM PAGE 4)

One of the most effective mechanisms for providing feedback on the WDK **and** getting status back on that feedback is also probably also one of the least known. That mechanism is Microsoft Connect. You just have to know where to look!

Go to http://connect.microsoft.com/VisualStudio/Feedback and enter "WDK" in the search box. After a while (what seems to be an eternity with the "Please Wait" graphic) you'll see a list of outstanding WDK issues. See *Figure 1* for an example. There's no tags or standard keywords (why?) so, to be sure you've seen all the issues you have to search on "driver" as well.

Connect seems to be reasonably effective, as well, because people seem to get a (semi-intelligent) response within a week or ten days. And I don't just mean the "Thank you for this report" type feedback. I mean a reply from somebody who actually seems to have read the bug and understand the issue being reported.

(CONTINUED ON PAGE 27)

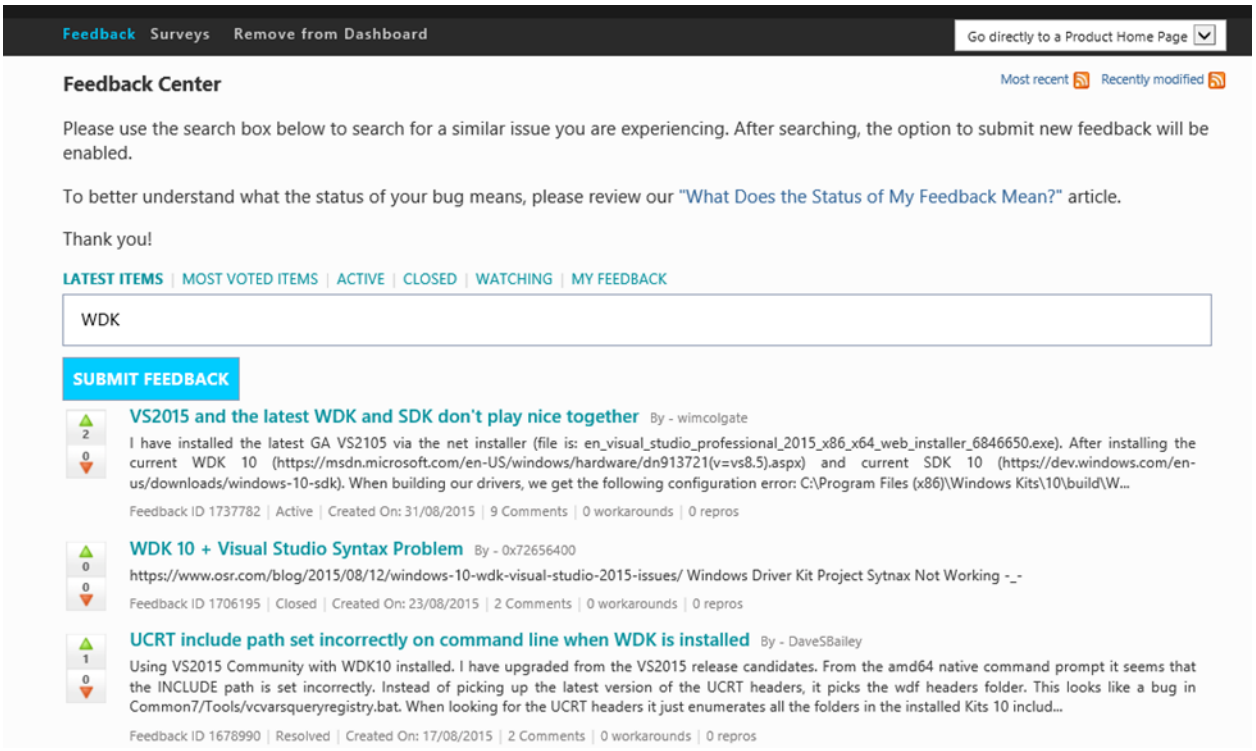


Figure 1

WINDOWS INTERNALS & SOFTWARE DRIVERS
For SW Engineers, Security Researchers, & Threat Analysts

Scott is extremely knowledgeable regarding Windows internals. He has the communications skills to provide an informative, in-depth seminar with just the right amount of entertainment value.

- Feedback from an attendee of THIS seminar

Next Presentation: Dulles/Sterling, VA
15-19 February

 © OSR Open Systems Resources, Inc.

You just THINK it Works
The Three Most Common KMDF Errors in Shipping Drivers

I wrote a blog post recently in which I tried to make the point (among others) that having a kernel-mode driver that is demonstrably working (and to the point where you never see it crashing or leaking) is not necessarily the same thing having a kernel-mode driver that is correctly written. Of course, anyone who's programmed for ten seconds knows that a program that looks like it "works" doesn't imply the program works **correctly**. But that's not what I'm trying to say here.

"Working" Does Not Imply "Correct"

I'm trying to call attention to the fact that when you write a kernel-mode driver, the environment in which that driver code runs -- the particulars of the system your code runs on and/or the implementation choices made in the current version of Windows or WDF – may allow your driver to run correctly. And this can make it **seem** that your driver was written correctly. However, unless your driver was written to carefully follow the architectural rules, when the environment in which your driver runs changes, you may get a big and very bad surprise. What had been "working right for months (or even years)" can all of a sudden stop working.

Windows kernel-mode development history is replete with these issues. For those fond of history, I'll just mention things like the Windows DMA API model, Port I/O Space resources appearing in Memory Space, and the necessity (or not) of calling **KeFlushIoBuffers** after completing a DMA transfer. Perhaps the best example of what I'm talking about is how, back in the 32-bit world, devs frequently used ULONG when they meant PVOID. They weren't then, and aren't now, the same thing. But storing your pointers in an ULONG worked. You could test your code all day long and it would **always** work. But that didn't make it **correct**. And when 64-bit Windows was introduced, a lot of people got that big, very bad, surprise and had to diagnose and fix their driver code. If they'd just done the right thing in the first place, they probably would have saved themselves some pain.

And so it is today. KMDF makes driver writing something that most any good C/C++ programmer can do, with very little experience and even less knowledge of Windows kernel-mode architecture. You Google around, you start with some sample code from the Internet, you ask a few questions on NTDEV, and bingo! Your driver practically writes itself.

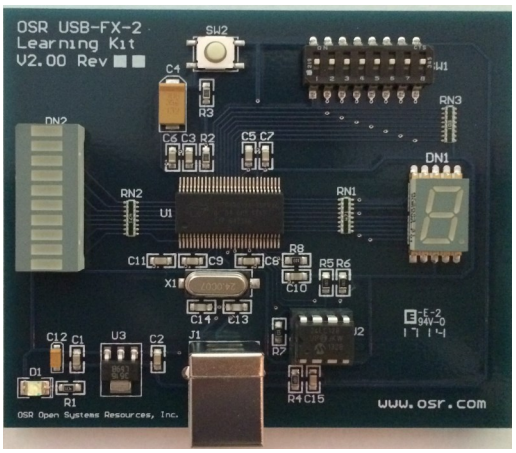
Or not. And before you think I'm singling out the more "casual" Windows driver devs, this problem is absolutely not restricted to them. Even experienced Windows driver specialists – folks who write Windows drivers every day for a living – sometimes forget, or just don't know, some of the more arcane architectural rules of the road. That's because, no matter how simple it seems, kernel-mode programming is hard and there are a lot of little issues about which you must be aware.

What Our Code Reviews Tell Us

One of the services we provide here at OSR are design and code reviews. We review a lot of drivers over the course of a year. And we see all types of drivers: Software-only kernel services that collect data and return it to user mode, file system filters, drivers for programmed I/O devices, and drivers that support busmaster DMA devices. We rarely see drivers that plainly don't work (although we do see some!). But we do see **a lot** of architectural errors.

So that you don't have to hire us to look at your code to find these errors for you, we'll describe for you the three most common errors that we see in KMDF drivers. And we'll also talk about how those errors can be fixed. We see these errors in almost equal numbers, so we'll present them in no particular order.

(CONTINUED ON PAGE 7)



OSR USB FX2 LEARNING KIT

Don't forget, the popular OSR USB FX2 Learning Kit is available in the Store at: http://store.osr.com.

The board design is based on the well-known Cypress Semiconductor USB FX2 chipset and is ideal for learning how to write Windows drivers in general (and USB specifically of course!). Even better, grab the sample WDF driver for this board, available in the Windows Driver Kit.

© OSR Open Systems Resources, Inc.

KMDF Errors... (Cont.)

(CONTINUED FROM PAGE 6)

Errors Involving Process Context Assumptions

The common architectural error that makes me the most crazy has to do with assuming the context in which an Event Processing Callback is executing. In KMDF, EvtIoXxx Event Processing Callbacks are called by the Framework, for a particular Queue, to provide the driver with a Request to process. So, for example, when a Queue has a read type Request to pass to the driver, the Framework will call the EvtIoRead Event Processing Callback specified by the driver for that Queue.

In many relatively sophisticated drivers, we see code similar to that shown in *Figure 1* (see the sidebar on p.26 about the example code), which might appear in the driver's **EvtIoDeviceControl** Event Processing Callback.

A quick look at this will show you this code is reasonably carefully written. It frames the call to **MmMapLockedPagesSpecifyCache** in a try block. It handles the exception appropriately. And, though you can't see it in Figure 1, I can tell you that this code even cleans-up correctly at the "done" label.

So what's not to like?

Note the specification of "UserMode" as the target address space (what the docs refer to as "access mode"… yuck!) into which to map the memory.

What's not to like is that EvtIoXxxx routines are, by architectural definition, **always** called in an **arbitrary** process context. Recall that the phrase "called in an arbitrary process context" just means that we can't know in advance the context the driver will be called in. In other words, we don't know what user-mode program will be the "current thread" when we're called.

It's not like this architectural constraint is some sort of secret. The WDK docs explicitly mention this under the topic of Request Handlers:

> The framework calls your driver's request handlers in an arbitrary thread context.

(CONTINUED ON PAGE 22)

```
// Map the device memory into the user's virtual address space.
//
// Note: Mapping to UserMode can raise an exception. We protect against
// that by doing the mapping within a try block.
//
__try {

    MappedUserAddress = MmMapLockedPagesSpecifyCache(
                                            MdlAddress,
                                            UserMode,
                                            MmNonCached,
                                            NULL,
                                            FALSE,
                                            NormalPagePriority);
}
__except (EXCEPTION_EXECUTE_HANDLER) {

    MappedUserAddress = NULL;
}
//
// Did that work?
//
if (MappedUserAddress == NULL) {

    //
    // No!  The only reason it could fail is lack of PTEs.
    //
    status = STATUS_INSUFFICIENT_RESOURCES;

    goto done;
}
```

Figure 1

 © OSR Open Systems Resources, Inc.

There's Gotta Be a Better Way
Critical Regions

Critical Regions are one of the more confusing and poorly documented concepts in Windows kernel mode development. Long considered something that only file system developers cared about, most developers just ignore the topic and assume that it doesn't affect them.

While most of us *are* safe being blissfully ignore of Critical Regions, there has been some discussion about them recently on the NTDEV discussion forum. Given that it's not exactly clear what a Critical Region does, let alone when you need them or when you don't, it's not surprising that there's confusion about their usage. The unfortunate part is sometimes you **must** use Critical Regions to avoid introducing a very subtle (but nasty) denial of service attack in your code.

Before we proceed, a word for those of you coming from a user mode background: do **not** confuse Critical Regions with Critical Sections! Entirely, completely, 100% unrelated. Forget about Critical Sections entirely before proceeding with this article (feel free to use a hammer if necessary).

The Basics

Your code enters a Critical Region by calling **KeEnterCriticalRegion** or **FsRtlEnterFileSystem** (which is just a wrapper for **KeEnterCriticalRegion**). Your code then leaves a Critical Region by calling **KeLeaveCriticalRegion** or **FsRtlExitFileSystem**.

The first important thing to note about Critical Regions is that they are a **thread specific construct**. Therefore, when you enter a Critical Region you are entering for the current thread **only**. Because of this, Critical Regions are most definitely not a synchronization mechanism.

What good are they then? The answer lies in an insightful comment provided with the **FsRtlEnterFileSystem** macro (*Figure 1*).

What this comment is trying to say is that entering a Critical Region prevents the current thread from being suspended. Thread suspension is performed by a Normal Kernel Asynchronous Procedure Call (KAPC), which is simply a kernel mode callback directed to a particular thread. By entering a Critical Region, your code prevents Normal KAPCs from executing on the current thread. Note that other things are done via Normal KAPCs as well (e.g. hard error popups), but for our discussion we just care that Critical Regions prevent thread suspension.

```
//++
//
//  VOID
//  FsRtlEnterFileSystem (
//      );
//
//  Routine Description:
//
//      This routine is used when entering a file system (e.g., through its
//      Fsd entry point).  It ensures that the file system cannot be suspended
//      while running and thus block other file I/O requests.  Upon exit
//      the file system must call FsRtlExitFileSystem.
//
```

Figure 1

Given this, we can make two statements:

- Threads in a Critical Region **cannot** be suspended
- Threads **not** in a Critical Region **can** be suspended (as long as the thread is running at IRQL PASSIVE_LEVEL)

(CONTINUED ON PAGE 9)

DESIGN AND CODE REVIEWS
You've Written Code — Did You Miss Anything??

Whether you're a new Windows driver dev or you've written dozens of drivers before, it's always hard to be sure you haven't missed something. Windows changes, WDF changes, security issues emerge. Best practices are a moving target.

Let OSR help! Our engineering team is 100% dedicated to Windows internals and driver development. Let us be the expert, second pair of eyes on your project… ensure it's done right!

 © OSR Open Systems Resources, Inc.

Critical Regions (Cont.)

(CONTINUED FROM PAGE 8)

At first this sounds pretty hideous. I mean, have you ever given serious consideration to what would happen if a thread running in your driver was suspended? Most driver developers don't think of this and, if they do, they assume that they're immune to suspension simply by running in kernel mode. No such luck. Any code in your driver that runs at PASSIVE_LEVEL and not in a Critical Region can be suspended.

We're Doomed!

What prevents this from being a complete mess is that in order to forcibly suspend a thread, a caller must have sufficient access to the thread in question. Thus, if a process has sufficient rights and wants to pause a running thread, they're allowed to do that. If this causes the process owning the thread to hang or otherwise malfunction, oh well! We can just document that under, "Don't Do That."

Why bother having Critical Regions at all then? Well, the problem comes when there's a knock on effect that causes problems in **other** threads or processes. What if suspending a thread in Process A causes threads in Process B to hang? Just because I have the authority to suspend threads in Process A doesn't mean I have that same authority with Process B, though I effectively achieved the same result. Even worse, imagine an unprivileged application that suspends own of its **own** threads, causing other privileged applications to hang or crash. This would be very bad indeed.

Thus, as you're writing your code you need to ask yourself: if the current thread was suspended, would that affect any other threads in the system? Most problem cases will occur when acquiring locks; What happens to other potential users of a lock if you acquire that lock and then your thread is suspended?

Take the file system for example. The file systems use a significant amount of file and volume level locking while processing I/O requests from user mode. What if a thread is currently holding one of these locks when it is suspended? This could potentially result in a system-wide deadlock as other threads come along and try to acquire the same lock to perform I/O operations of the file or volume.

We solve this problem in one of two ways. The first way is to acquire a lock that also implicitly enters a Critical Region. Examples of these types of locks are:

- Spinlocks
- Kernel Mutexes
- Fast Mutexes
- Guarded Mutexes

However, if you acquire a lock that does not implicitly enter a Critical Region then you must explicitly call **KeEnterCriticalRegion** before acquiring the lock and **KeLeaveCriticalRegion** after releasing it. Examples of locks that do not enter a Critical Region are:

- Executive Resources (ERESOURCE)
- Unsafe Fast Mutex
- KEVENT
- KSEMAPHORE

If you're using one of the above as a synchronization mechanism and therefore acquire it from multiple different thread contexts, you really need a Critical Region to keep yourself safe from a DoS.

Conclusion

In a nutshell, the Critical Region can be thought of simply as a way to prevent thread suspension. Threads executing in kernel mode are subject to being suspended, which is fine unless your driver creates cross thread dependencies (e.g. with a lock). In that case, it's up to you to properly enter and leave a Critical Region at the right times.

Follow us!

 © OSR Open Systems Resources, Inc.

An Embarrassment of Riches

Visual Studio Customizations and Extensions for Driver Devs

I used to be a Visual Slick Edit fan. But now I've fully bought-in to using Visual Studio (VS) as an editor for driver development.

Whenever I tried to use VS for driver development in the past, I was always frustrated with strange behavior ("Why does it insist on indenting my case statements incorrectly!"), lame and close to useless syntax highlighting, and pointless information panes.

Over the years, Visual Studio and I have come to an understanding. A happy agreement, even. I found that there are two primary keys to becoming happy with VS. These are:

- Taking the time to customize the environment
- Selecting the right set of Visual Studio Extensions

Interestingly, these two things work together.

Customization

If you're like me, you've got work to do. All the time. You just want to open a project and get busy writing your code. You don't want to spend time fooling with your environment. You've got better things to do.

The thing is, with Visual Studio, time spent getting your environment customized to your needs and preferences will be time well spent. You'll find the VS editor less annoying every time you use it. And, yes, I hesitate to say it, but you **will** be more productive. So, be good to yourself. Take a Friday afternoon and spend time on Visual Studio customization settings. I'll even give you a few ideas on where you can start.

In case you haven't noticed, there are at least 32 million different things you can set, change, or tune within Visual Studio. Just opening the dialog box from "Tools -> Options… ", presents you with around two dozen **primary** sets of options that you can play with. The exact number of options depends on what you've installed with VS. But in terms of customizing your driver coding environment, there are two or three primary option sections that you will definitely want to play with.

The first set of options are in the **Environment** section. Here, you can customize things ranging from the overall color scheme (you've probably already changed that) to how the "tabs" for each document that you have open in the editor look. I recommend you just take the time to walk through each of the various subsections and the options they contain some afternoon.

(CONTINUED ON PAGE 11)

ALREADY KNOW WDF? BOOST YOUR KNOWLEDGE

Read What Our Students Have to Say About
Writing WDF Drivers II: Advanced Implementation Techniques

It was great how the class focused on real problems and applications. I learned things that are applicable to my company's drivers that I never would have been able to piece together with just WDK documentation.

A very dense and invaluable way for getting introduced to advanced windows driver development. A must take class for anyone designing solutions!

Next Presentation:

Amherst, NH 8-11 December
Amherst, NH 15-18 March

 © OSR Open Systems Resources, Inc.

Visual Studio Customizations.. (Cont.)

(CONTINUED FROM PAGE 10)

Unless you're into boring, monochromatic, color-coding and green comments (I *hate* green comments) you'll want to take some time and update the **Fonts and Colors** subsection (shown in *Figure 1*). Here you can change how VS syntax highlighting colors various things. For example, I like comments to be in gray. And I like my local variable names to be bold. Preferences for exactly how and what you highlight vary widely among developers, so there's no single scheme that will work for everyone. Some people like a more muted palette of colors with most syntax elements being in similar colors (this is the VS default), and others like more aggressive color coding. It certainly true that you can easily go crazy here and wind-up with a mess, so make changes gradually. Note that changes do not take effect immediately, so you'll have to click "OK" to close the Options dialog before you see your results.

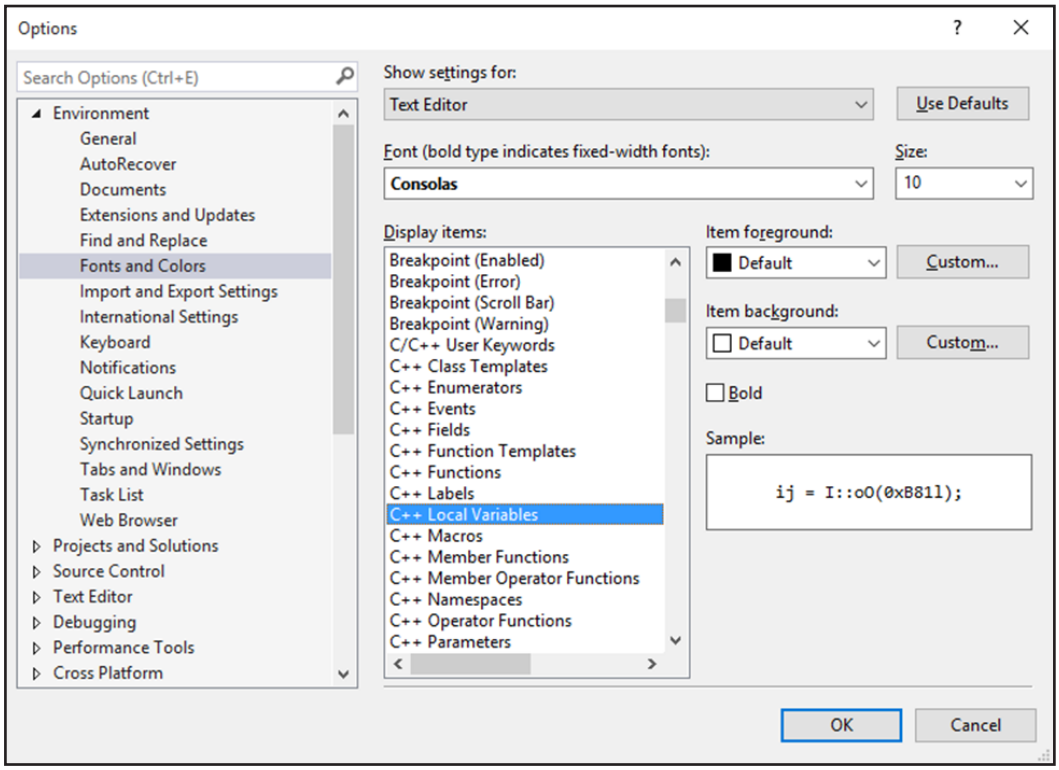


Figure 1

Another small item that you'll see in the Environment section is the **Task List** subsection. This section is often overlooked, I think. This section allows you to define a set of "tokens" that can appear in your comments. These comments, and the file and line on which they appear, will be picked-up by VS and listed in the VS "Task List" window (that you can show via View -> Task List). This allows you to leave notes to yourself about things that you want to return to in the code, and these notes are all gathered in one place within your solution. The default tokens are "TODO" and "HACK"... but you can of course add your own. I find this makes code cleanup easy when it's time to think about shipping a project.

The next major section in the Options dialog that you'll almost certainly want to customize is the **Text Editor** section (See *Figure 2, p. 18*). Here's where you can define, on a per-language basis, the formatting, indentation, and brace style that you prefer. Notably, this is where you set the option to use spaces instead of tabs in your code. You **do** use spaces instead of tabs, right? I mean, it's the only way code should be written.

(CONTINUED ON PAGE 18)

THE NT INSIDER - Hey...Get Your Own!

Just send a blank email to join-ntinsider@lists.osr.com — and you'll get an email whenever we release a new issue of The NT Insider.

 © OSR Open Systems Resources, Inc.

Guest Article
The *State* of Synchronization

By Alex Ionescu
Community Contributor

One of the signature features of the Windows kernel is the rich array of synchronization mechanisms available for use, most of which are also exposed to 3rd party driver developers. Unfortunately, this is a double-edged sword. On one hand, such a comprehensive set of tools allows developers to choose *the* best locking mechanism for a particular job. On the other hand, synchronization is a complex topic and the unique properties of each type of lock, combined with sometimes less-than-stellar documentation, leaves many well-intentioned developers confused about which lock is best for a given situation. This can lead to developers choosing a "one-size-fits-all" implementation strategy, where they choose a locking mechanism they understand over more specialized locks that would perform, or fit their needs, better. In the best cases, optimizations are ignored and efficiency is lost. In the worst of cases, incorrect lock usage causes deadlocks or hard-to-diagnose issues as subtle scheduler and IRQL requirements are misunderstood. The goal of this article, therefore, is to examine both changes in how some of the more well-known locking mechanisms work, as well as to describe the latest generation of synchronization primitives. Note that this article doesn't attempt to go into the deep implementation details of these locks, or cover non-3rd party-accessible locks as the book *Windows Internals* does. Also not covered are non-lock based synchronization primitives such as InterlockedXxx APIs or other lock-free semantics. You should expect to have to consult MSDN for additional information.

Asking the Right Questions

I often see driver developers asking "what lock should I use?" and expecting some sort of universal answer – instead of thinking of the different lock primitives offered as akin to tools on a belt. Much like you wouldn't want to sit through LA traffic in a Ferrari, or drive a Prius at Nuremberg, each lock offers its own pros and cons, and picking the right lock is often about understanding its characteristics and how those characteristics map to the lock's intended use. Once the lock's primary attributes are understood, it's often clear what the winning lock is *for this particular use*. Here's the first set of questions you should ask yourself each time you attempt to guard against a particular synchronization concern. Note that these are not in any particular order:

- Is it okay to use a *wait*-based lock, or do I need to use a *spin*-based lock? Many developers are tempted to only think of IRQL requirements to drive the answer. But performance concerns are also important. Is it really worth taking a potential page fault or context switch when computing a 32-bit hash on 256-byte bounded data?
- Do I need to support *recursion* in my synchronization logic? That is, should a thread be allowed to re-acquire an already acquired lock?
- Will the data sometimes be concurrently accessed only for read semantics, with only some uses requiring write semantics, or are all concurrent accesses likely to require modification of the data?
- Is heavy contention expected on the lock, or is it likely that only one thread or processor will ever need the lock?
- Does the lock require fairness? In other words, if the lock is contended, does the order in which the lock is granted to waiters matter?
- Are there CPU cache, NUMA latency, or false sharing considerations that will impact performance?
- Is the lock global, or is it highly instantiated/replicated throughout memory? Another way of putting this is: are there storage cost concerns?

(CONTINUED ON PAGE 13)

I TRIED !ANALYZE-V...NOW WHAT?

You've seen our articles where we delve into analyses of various crash dumps or system hangs to determine root cause. Want to learn the tools and techniques yourself? Consider attendance at OSR's Kernel Debugging & Crash Analysis seminar.

 © OSR Open Systems Resources, Inc.

The *State* of Synchronization (Cont.)

(CONTINUED FROM PAGE 12)

Try to build a table that has these questions, and answer each one with a simple "yes", or "no".

In some cases, you may need to look beyond just how your code interacts with the data or region of code that needs synchronization, and also analyze the type of machine your code is expected to run on. Also, if the lock is surrounding runtime dynamic data, a consideration might be the amount and structure of the data your code needs to ingest while holding the lock. For example, a Webcam driver is unlikely to be used in a server environment with > 64 processors and multiple NUMA nodes. Webcam I/O is also unlikely to come from multiple threads. Questions around false sharing and fairness are easy to answer in such scenarios.

Types of Locks

Once you've gained a clear view of your requirements as described above, your next task is to understand how those requirements map onto the different kernel synchronization primitives. This will guide you to the right lock type for your use. Let's begin by first breaking down the locks into *wait* (*thread*)-based vs. *spin*-based.

As of Windows 10, the following *wait*-based locks are available:

- Mutexes, Fast Mutexes, and Guarded Mutexes
- Synchronization Events, and Semaphores
- Executive Resources and Pushlocks

On the *spin* side, the kernel offers:

- Spinlocks and Interrupt Spinlocks
- Queued Spinlocks (In-Stack and Global)
- Reader/Writer Spinlocks

Spin-type Locks

Two things should lead a developer to look at spin-type locks: either there is an IRQL restriction, or the operation that will be performed while holding the lock has a very fast turnaround time: the cost of the potential page fault or context switch that could occur outweighs the cost of keeping the CPU busy during the operation. Things like linked list operations, hash lookups, and maybe even certain types of tree lookups are good candidates here.

If there is an IRQL restriction, is it interrupt based? If so, the exercise ends here: the Interrupt Spinlock is the *only* kernel-provided primitive that can be used to serialize a code segment with your ISR. Similarly, if interrupts are not involved, there's no use considering the Interrupt Spin lock as your chosen locking mechanism.

Recursion is our next question, and that is an easy one as well. Simply put, no spin-based lock mechanism offers recursion. In other words, if the requirement is IRQL-based, you must redesign your code to avoid recursion. If the requirement is perf-driven, you need to address what's more important: the need for recursion, or the need for latency reduction? If recursion is a key need, spin-type locks cannot be used.

Next up is the need to separate reader vs. writer semantics. If all contended accesses modify the data, there's no need to complicate things with a reader/writer lock. However, if there are performance benefits to be gained because multiple contentions are merely caused by read-only semantics, you may want to use a *multi-reader-single-writer spinlock.* While not at all well known, Reader/Writer Spin Locks do actually exist. In fact, they've been officially available since Windows Vista SP1. An article that describes these locks appeared in the March 2013 issue of *The NT Insider*.

The consideration that you are likely to encounter next is fairness. If you don't need read/write semantics, you still have two locks to choose from. On the other hand, if you do, then you need to sacrifice fairness, and Reader/Writer Spin Locks are inherently unfair with regard to writer contention. This makes sense, because if writer contention is a big deal for your use case, the optimization to allow concurrent readers likely won't help much.

(CONTINUED ON PAGE 14)

 © OSR Open Systems Resources, Inc.

The *State* of Synchronization (Cont.)

(CONTINUED FROM PAGE 13)

The need for fairness will determine your decision between In-Stack Queued Spinlocks or Standard Spinlocks. As you can probably guess, the former, by adding additional complexity around acquisition, allow an ordered waiter chain to be built, while the latter leave acquisitions to chance.

Note that one of our questions is around the amount of contention that is expected on the lock. It's important to realize that this question directly impacts the need for fairness: if the lock is almost never contended, fairness does not come into play. In other words, a non-contended use case should just directly utilize the Standard Spinlock, as additional questions basically assume the need to optimize contention.

If fairness is irrelevant, but contention is still an issue, cache line optimizations, NUMA awareness, and false sharing avoidance come into play next, and may offer the In-Stack Queued Spinlock another chance in your implementation. You see, since the Spinlock is a global bit in memory, multiple contenders will spin on the same bit, located in some DIMM on some NUMA node. Immediately, this causes initial memory accesses by other processors to require cross-node memory access, increasing latency. Because each processor is then likely to cache the bit, subsequent spinning on the bit will reduce the non-locality effects, but each modification (due to one processor acquiring the lock) will force a bus-wide cache snoop, adding additional latency. For a highly contended-lock on a NUMA system with a large number of processors, this memory traffic and resulting latency may not be desired.

Finally, if either fairness or locality steered you toward the In-Stack Queued Spinlock, the question of storage costs comes into play. Thankfully, due to the implementation of these locks, both an In-Stack Queued Spinlock and a Regular Spinlock each only consume a pointer-sized slot. At *runtime*, the In-Stack Queued Spinlock will then consume *n*-additional instances in memory, but since the lock is spin-based, you cannot have multiple threads on the same CPU in the middle of acquiring different spinlocks – and thus, there's no storage cost explosion to worry about. We'll see that this is not necessarily true in *wait*-based locks.

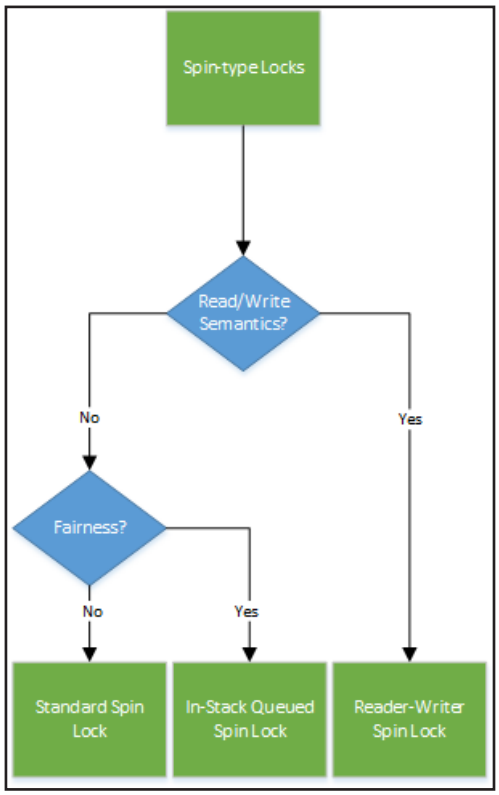


Figure 1 - Decision Tree for Spin-type Locks

As we've seen, due to the somewhat limited selection of available *spin*-based lock types, once this direction has been chosen, the choices are somewhat easier to make. Once a key characteristic is selected, there's usually a single lock that provides the required functionality. The exercise therefore, is deciding what's more important: latency, fairness, or throughput.

Wait-type Locks

The sheer number of locking mechanisms that utilize the thread scheduler to provide synchronization makes it harder to select a single wait-type lock. However, by understanding some of the implementation details of each lock type, the choice usually becomes just as clear as it is with spin-type locks. Let's once again follow the previously discussed set of questions to help us pick the correct wait-type primitive for mutual exclusion.

If recursion is a requirement, the choice is immediately limited to two locking primitives: the Regular Mutex and the Executive Resource. If recursion isn't a requirement, you should probably eliminate these two from the list. Although you can use these lock mechanisms in non-recursive scenarios, they incur an overhead by supporting it.

Next up is the need for single-writer/multi-reader semantics. If you need reader/writer semantics and your locking mechanism also needs to support recursion then your choice is the Executive Resource. On the other hand, if recursion is not needed, then the Pushlock now becomes a candidate as well. Further questions will help you decide between the Executive Resource and Pushlock. Be careful, however! While Executive Resources do support recursion, they support it only within compatible semantic usage levels. For example, an exclusive and shared owner can both attempt another shared acquire – but a shared owned can never

(CONTINUED ON PAGE 15)

 © OSR Open Systems Resources, Inc.

The *State* of Synchronization (Cont.)

(CONTINUED FROM PAGE 14)

attempt an exclusive acquire (it will deadlock on itself). A final remark here is that officially, Pushlock support for 3rd party driver developers is only available starting in Windows 8 and later. Previously, accessing this lock required linking with the File System Mini-Filter Manager and using the FltXxx routines that wrap the Pushlock API, which might look odd for say, a network card driver.

Note that in the wait-based world, there is another type of multiple-acquire primitive that can be used (although one that does not differentiate writers vs. readers): the semaphore. Typically used to protect a *group* of data resources that a set of threads can use, it functions like a non-recursive Mutex which allows multiple acquisitions from multiple threads – up to a certain defined limit, at which point future acquires will block. If this specific need is part of your requirements, the semaphore is the only primitive which can provide multiple-writer acquisition.

If neither recursion (eliminating the need for a Standard Mutex) nor multi-reader (eliminating the need for Executive Resources or Pushlocks) or multi-writer (eliminating Semaphores) semantics are needed, this leaves the Synchronization Event, the Fast Mutex, and the Guarded Mutex in play. At this point, the choice between the Fast and Guarded Mutex types on Windows 8 and later is *an absolute no-op.* The two implementations are 100% identical code in modern versions of Windows, contradicting much of the guidance offered in previous books and whitepapers. What gives? Originally, there was indeed a difference. But changes in the hardware of modern processors has over the years allowed Windows to converge the implementation of these two lock types. But we haven't answered all of our questions yet. Does fairness matter? If it does, it's important to note that Synchronization Events and Fast/Guarded Mutexes are all fair as they utilize the dispatcher, which implements wait queues. Additionally, Pushlocks are also fair, and even implement wait queue optimization by re-ordering the wait queues. So this doesn't help us narrow things down.

Let's look at the contention risks next. In a both highly contended and no-contention scenarios, Synchronization Events will always require a round-trip through the dispatcher, including acquiring the dispatcher lock (with variable costs depending on OS version). On the other hand, Fast/Guarded Mutexes have an interesting optimization: their acquisition state is first stored as a bit outside of the dispatcher's purview, and only once contention around the bit is detected, does the dispatcher become involved – because Fast/Guarded Mutexes actually integrate a Synchronization Event! This implies that in a non-contended state, the acquisition and release cost of a Fast/Guarded Mutex is as cheap as that of a spinlock, as only a single bit is involved. And so, if your use case calls for a lock that is mostly non-contended, wrapping the Synchronization Event behind the pseudo-spinlock optimization that a Fast/Guarded Mutex offers will prove beneficial and reduce latency.

Let's look at caching and sharing next. Out of all the locks offered in the wait-based bucket, only Pushlocks offer any sort of NUMA and cache-aware functionality. In fact, a *special* type of Pushlock, called the cache-aware Pushlock, exists to offer this additional performance gain if needed. Indeed, this does mean that even if multi-reader semantics aren't needed because all your lock contention is under exclusive semantics, a Pushlock may still be the best solution due to its ability to scale across multiple processor cache lines and NUMA nodes. Unlike a Queued Spinlock, however, this efficiency is implemented at initialization time,

(CONTINUED ON PAGE 16)

WE KNOW WHAT WE KNOW

We are not experts in everything. We're not even experts in everything to do with Windows. But we think there are a few things that we do pretty darn well. We understand how the Windows OS works. We understand devices, drivers, and file systems on Windows. We're pretty proud of what we know about the Windows storage subsystem.

What makes us unique is that we can explain these things to your team, provide you new insight, and if you're undertaking a Windows system software project, help you understand the full range of your options. AND we also write kick-ass kernel-mode Windows code. Really. We do.

Why not fire-off an email and find out how we can help. If we can't help you, we'll tell you that, too.

Contact: sales@osr.com

 © OSR Open Systems Resources, Inc.

The *State* of Synchronization (Cont.)

(CONTINUED FROM PAGE 15)

and is not a runtime acquisition cost. In other words, using a cache-aware Pushlock in a 640 processor, multi-node machine, can result in multiple memory allocations of multi-KB lock structures.

And this is where the last question comes into play: is this a global lock, or will it be instantiated and replicated across multiple structures? If the lock is going to be replicated multiple times, size must be considered: Pushlocks consume a mere pointer. Executive Resources, on the other hand, consume more than 100 bytes on x64 systems. If recursion is not needed, it's clear that this cost is not worth it, especially if replicated across multiple structures. Likewise, all Mutex types also utilize 56 bytes, and even Synchronization Events use 24 bytes. If storage costs are important, and recursion is not a need, Pushlocks are great, even if they offer multi-reader functionality you may never need.

If size is an issue and tentatively selected for cache-aware Pushlocks as a result of our discussion above, you now have an unintended side-effect. In order to provide cache-awareness, these types of locks are both padded to cover an entire cache line (64 bytes these days), **and** potentially replicated across each NUMA node. With a global lock, there is an upper limit on the memory costs of such a cache-aware Pushlock (n CPUs per node * cache line size * nodes), but if this lock is local to a frequently instantiated data structure (let's say, a per-process state structure your driver maintains), you must now multiply this cost by potentially a few thousand active processes.

(CONTINUED ON PAGE 17)

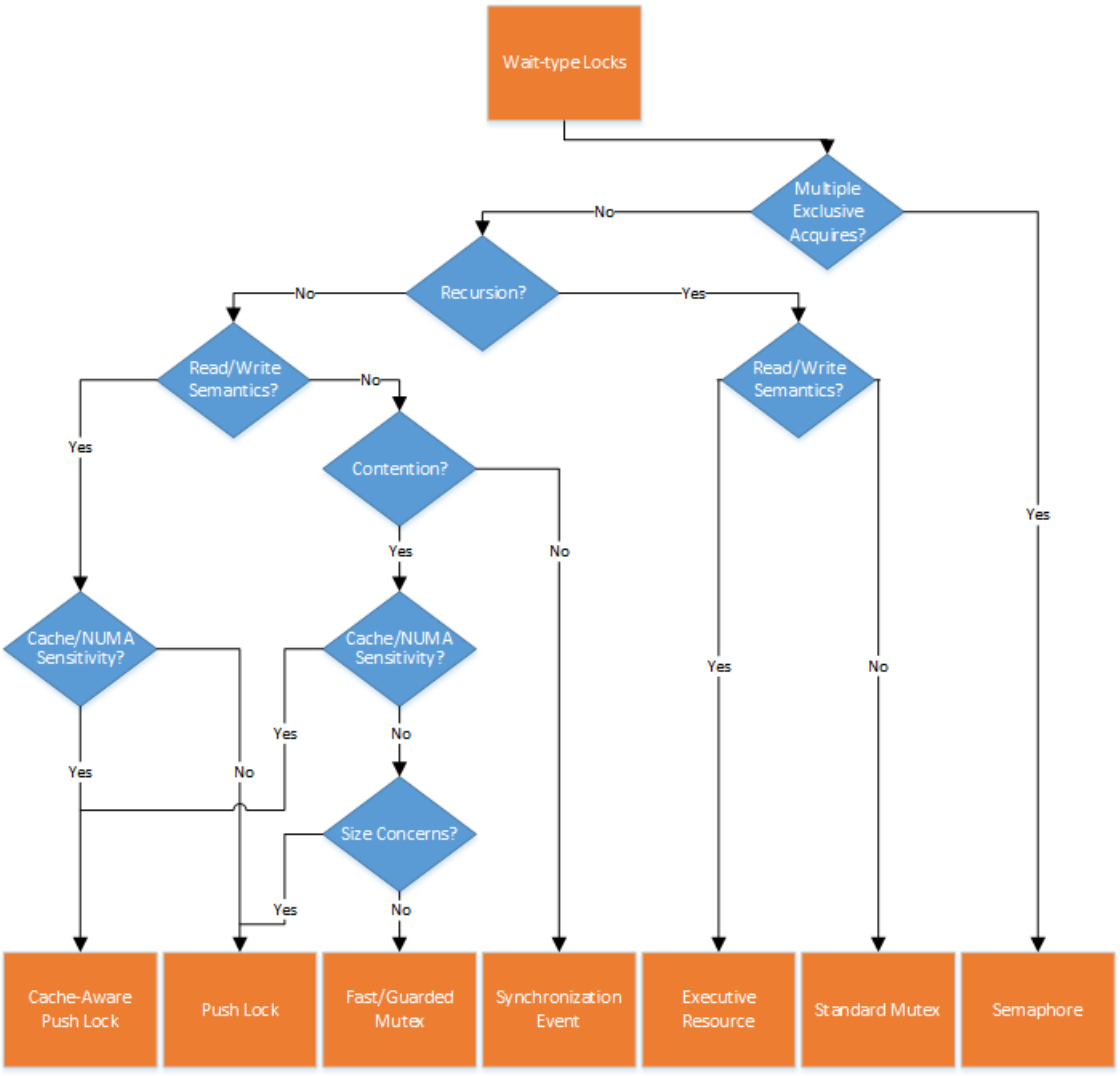


Figure 2 - Decision Tree for Wait-type Locks

 © OSR Open Systems Resources, Inc.

The *State* of Synchronization (Cont.)

(CONTINUED FROM PAGE 16)

With this in mind, Windows 10 includes what is probably the most complex lock yet: the Auto-Expand Cache-Aware Pushlock. As the name suggests, these Pushlocks begin by being cache-unaware: unless there's significant shared usage, cache and NUMA awareness doesn't help much, as exclusive acquirers still need to touch the cross-processor local structures, and the storage costs are being wasted. On the other hand, if the usage is agreeable to it, the storage costs may be worth enabling multiple remote processors to hyper efficiently perform read-only operations on the locked data. The Auto-Expand Cache-Aware Pushlock provides the best of both worlds, at the expense of additional internal tracking and telemetry, and a one-time expansion cost.

Ownership

There are a few, minor, one-off considerations that may impact your ability to use a particular kind of lock over another. For example, you may strongly care about *ownership*. For Executive Resources and all types of Mutexes, the kernel tracks this data for you, and even provides certain capabilities built on top of this semantic: Regular (but not Fast/Guarded) Mutexes can be "abandoned" and provide protection against unexpected owner termination (this should normally only matter for user-mode), while Executive Resources provide lock handoff boosting and I/O boosting to avoid starvation and priority inversion scenarios.

Ownership can also be invaluable for debugging locking bugs in code and deadlock situations, and for locks which don't provide it (such as Pushlocks or Spinlocks), this can make troubleshooting much harder. You should note, however, that if ownership is important to you merely for troubleshooting, you can easily encapsulate any of the kernel-provided locks in your own data structure, which stores the owner by using KeGetCurrentThread during acquisition, and clears it during release. This also helps you provide assertions against double-releases or recursive acquisitions, which the kernel may only have on checked builds.

APC and I/O Considerations

While the framework described in this article is likely to be sufficient for the selection of the "best lock" for most use cases, it *does* omit one important detail specific to wait-based locking implementations: the interactions with APC delivery, and ultimately your ability to perform file-system-based I/O, or even certain types of registry access. In some cases, this may prove to be the overriding factor behind the lock selection. We all know that spin-based locks all operate at DISPATCH_LEVEL or above, but wait-type locks are bit more complex.

Unfortunately, space doesn't permit me to address all the issues regarding APC delivery and wait-type locks in this printed article, but you should head over online to OSR's Website for the full version of the article describing these intricacies.

Conclusion

The wide breadth of locking primitives in the Windows kernel can sometimes overwhelm developers. Changes to the internal behaviors of certain locks, or the introduction of new locks, can exacerbate this problem, as can hidden assumptions and operations performed by different interfaces to the same underlying primitive. Hopefully, this article provides a comprehensive framework for selecting the right type of lock based on a standardized set of questions, as well as describing deeper considerations that can lead to easy-to-miss design flaws if they are not understood.

Alex Ionescu is a world-class software architect and consultant expert in low-level system software, kernel development, security training, and reverse engineering. He teaches Windows Internals courses to varied organizations around the world and is the coauthor of the last two editions of the Windows Internals series, along with Mark Russinovich and David Solomon. He is currently the Chief Architect at CrowdStrike, Inc and can most easily be reached at @aionescu or his blog at www.alex-ionescu.com.

Follow us!

 © OSR Open Systems Resources, Inc.

Visual Studio Customizations.. (Cont.)

(CONTINUED FROM PAGE 11)

You'll want to play around with various settings in this section. Take the time to work with them, until you've got what you like. If you're like me, you'll be amazed at how configurable VS is, how much less annoying you can make its default editing behaviors, and how much it will automatically do for you once you just take the time to tell it how you like things.

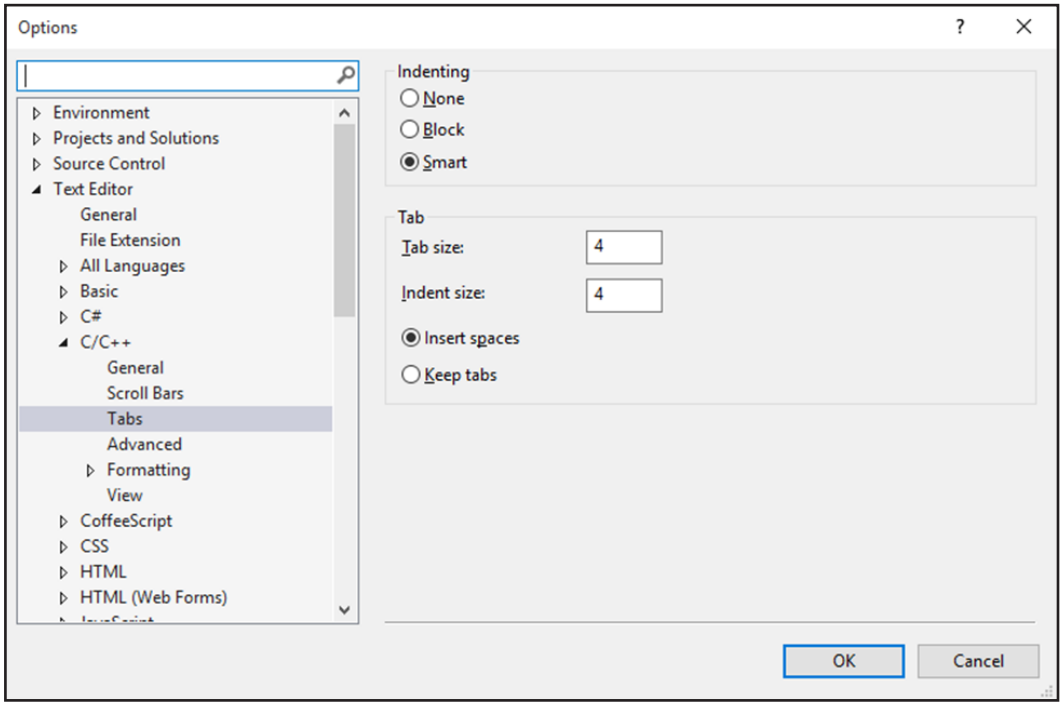


Figure 2

The third major section of the Options dialog that you might want to customize is the **Source Control** section. I've found that integrating VS into our source control system uber-convenient. Gone are the annoying prompts about read-only files. And you can see at a glance which files you've checked-out or modified. We use Perforce as our source control system, and the P4VS plug-in works just great.

Extensions

Even with all the customization that are available, Visual Studio can still be made better. A **lot** better, in fact. And one way that it can be improved is through the use of Extensions.

A wide range of VS Extensions are available on the web, and they range from the free and simple to the extensive and expensive. I'll highlight three Extension packages that I've used and have come to love.

The first collection of Extensions, one that I really think is a must have, is **Productivity Power Tools** (See *Figure 3, p. 19*). It's free, it's provided by Microsoft, and it works great. You can individually enable or disable the various features it provides. This allows you to eliminate those features you find horrible (I'm thinking the Structure Visualizer here), and only use those that you find helpful.

(CONTINUED ON PAGE 19)

DID YOU KNOW?

You can receive an additional $100 off OSR public seminar registration fees when you purchase an OSR USB FX2 Learning Kit

I TRIED !ANALYZE-V...NOW WHAT?

You've seen our articles where we delve into analyses of various crash dumps or system hangs to determine root cause. Want to learn the tools and techniques yourself? Consider attendance at OSR's Kernel Debugging & Crash Analysis seminar.

 © OSR Open Systems Resources, Inc.

Visual Studio Customizations.. (Cont.)

(CONTINUED FROM PAGE 18)

What features does Productivity Power Tools provide that are really cool? One subtle, but very helpful feature is "Syntactic Line Compression." What this does is reduce the height of lines that don't contain any text by 25%. This helps fit more useful code onto your screen, while still preserving your white space. Another feature I love is having more control over the "tabs" that represent each open document. Productivity Power Tools allows you to customize this little aspect of your VS use in many helpful ways, including the ability to color-code the tabs for files that belong to different projects within one solution.

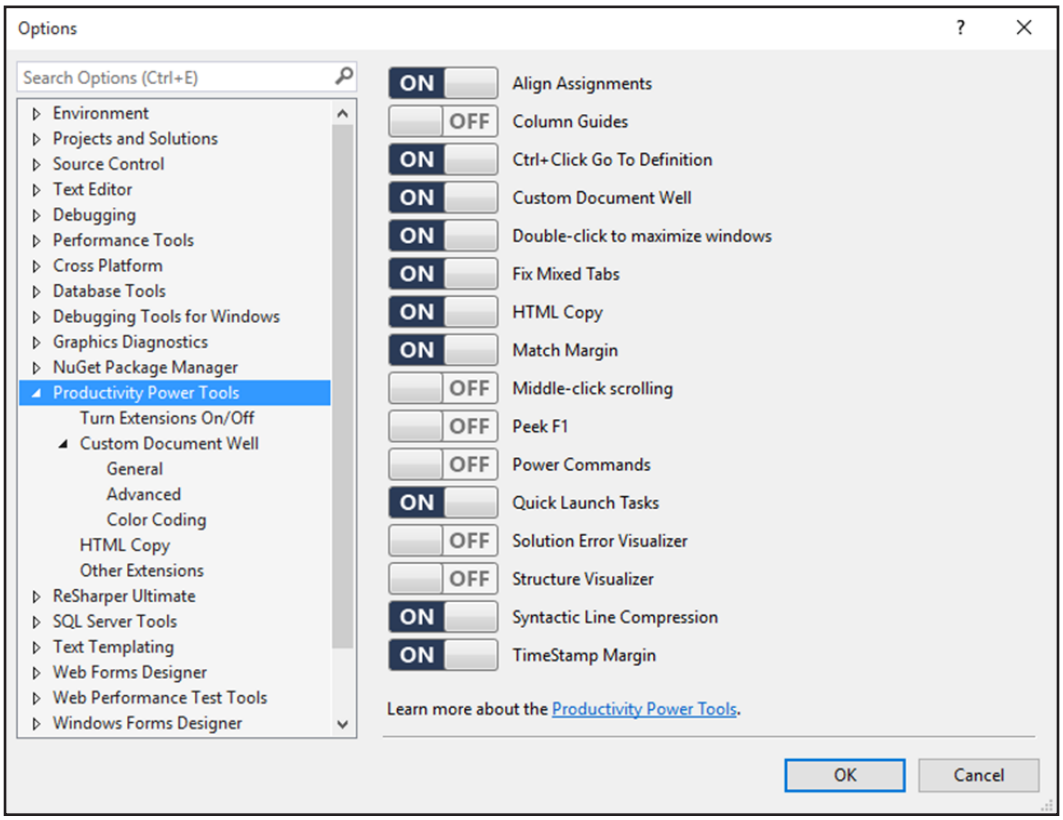


Figure 3

Another thing that Productivity Power Tools will do is automatically detect mixed spaces and tabs in a source code file, and offer to convert it to consistently use one or the other with just one click. How cool is that?

Those are just a few small items. I strongly urge you to download the latest version of Productivity Power Tools and play with it. It's free, and if you hate it you can uninstall it. What have you got to lose?

Moving beyond free tools, there are two major paid Extensions that you may want to consider. The first is **ReSharper C++** (by JetBrains, details here). I became acquainted with ReSharper during a period of time when I was writing lots of C# code. It is no exaggeration for me to say that I love this Extension so much that if I had to write C# code on a daily basis I'd be willing to pay for the damn thing out of my own pocket. ReSharper elevated the quality of my C# code from "beginner" to "cool" with very little effort on my part.

ReSharper C++ is a relatively new addition to the ReSharper family, and as such is still suffering some growing pains. For example, in the most recent release (which added some clever handling of asserts) it does not properly understand WDK style ASSERT statements. But, whatever.

It's difficult to describe all the things that ReSharper can do, but I'll try to hit some of the things I personally like best. For one thing, it supports an intelligent version of "rename" that actually works. This lets you rename variables (local or global), structure fields, class members, and functions. I'm insane about names, I obsess over what variables should be called, so for me this is a really helpful feature.

ReSharper's strength has always been its ability to analyze your code. In C#, it's uncanny the way it suggests parameters. In C++, ReSharper also has a few clever tricks. It improves on VS Intellisense, both in suggesting function names and also in terms of suggesting variables to be passed as arguments. It will also do handy little things like substitute "->" for "." automatically if you unintentionally type the wrong one. Or, if you select the "if" in an "if" statement, you can ask ReSharper to invert the sense of the

(CONTINUED ON PAGE 20)

 © OSR Open Systems Resources, Inc.

Visual Studio Customizations.. (Cont.)

(CONTINUED FROM PAGE 19)
"if" (automatically converting the text, adding an "else" clause if one was not there already, and putting the code that was in the "if" clause into the "else" clause).

And, of course, ReSharper C++ adds a "smarter" type of syntax highlighting and color coding.

Just like VS itself, ReSharper C++ has its own set of options that allow you to tune, customize, and configure it to your liking. ReSharper C++ also lets you define default naming styles. So, for example, you can tell it that you use Camel Case for variable names, and start those names with upper case if they're formal parameters to the function and lower case if they're local variables. And, if you don't follow these rules in the code you write, ReSharper will gently remind you.

Another nice feature that ReSharper has is that it'll show you (in the VS scroll bar) the location of any issues it detects in your code.

The age-old competitor to ReSharper is Visual Assist (by Whole Tomato Software, details here). Unlike ReSharper, Visual Assist started life as a C++ package. Thus its support for C++ is significantly more mature than that in ReSharper.

Most of the tricks that ReSharper can perform, Visual Assist can perform as well. And, like ReSharper, Visual Assist can be fully customized and tuned so that it does what you want, and avoids the sorts of things that you consider annoyances. It provides enhanced syntax coloring including a deeper understanding and coding of various data types than Visual Studio itself provides. It enhances Intellisense, and it knows to change your errant "." to "->" and vice versa. It provides a properly working rename capability that is a pleasure to use.

One small feature that Visual Assist has that I really love is how it highlights references to a given symbol. With Visual Assist, when you click on a symbol it will highlight all uses of that symbol within its scope. What makes it superb is that fact that it highlights uses of that symbol on the left side of an equals sign in pink, and uses of the symbol on the right side of an equals sign in blue (see *Figure 4*). This really speeds up my understanding when I'm working through code.

```
    for (ULONG i=0; i < WdfCmResourceListGetCount(ResourcesTranslated); i++) {

        //
        // Get the i'th partial resource descriptor from the list
        //
        resourceTrans = WdfCmResourceListGetDescriptor(ResourcesTranslated, i);

        if(!resourceTrans) {
#if DBG
            DbgPrint("NULL resource returned??\n");
#endif
            return(STATUS_DEVICE_CONFIGURATION_ERROR);
        } else {
        }

        //
        // Let's examine and store the resources, based on their type
        //
        // We expect to get a single register block that we'll use to talk
        // to the device and a single MSI interrupt.
        //
        switch (resourceTrans->Type) {

        //
        // The memory resource we're expecting is a 128 byte long register block
        //
        case CmResourceTypeMemory:

#if DBG
            DbgPrint("Resource %lu: Register 0x%0x:%0x Length %lu.\n",
                        i,
                        resourceTrans->u.Memory.Start.HighPart,
                        resourceTrans->u.Memory.Start.LowPart,
                        resourceTrans->u.Memory.Length);
#endif
            memoryResourcesFound++;
```

Figure 4

Another thing I love about Visual Assist is that it spell checks my comments as I type them. This makes me look smart. I need the help.

ReSharper C++ and Visual Assist each have their own unique features, and there is absolutely no way I can properly describe the features of either of these very powerful packages in this short article. Heck, you'll barely be able to get a handle on the extent of their features by spending time on the vendor's web sites! There are many little, subtle, features in each package that combine to really smooth some of Visual Studio's rough edges. Fortunately, both packages are available for free trial, so you simply have to download each and try them yourself.

If you do download the trial version of these Extensions, you'll want to enable only one of them at a time. Each can be individually disabled. And you'll want to spend the time to do at least some basic customization and tuning of the package. Don't waste your time downloading one of these packages, using it in its default mode, and making your decision based on that. Spend a little time

(CONTINUED ON PAGE 21)

 © OSR Open Systems Resources, Inc.

Visual Studio Customizations.. (Cont.)

(CONTINUED FROM PAGE 20)
customizing, and learning about its available features. Trust me on this. It'll be time well spent. And when you make your decision, you'll be happy with the outcome.

Because I've been a Microsoft MVP practically forever, over the past several years the makers of ReSharper and Visual Assist have been nice enough to provide me with free copies of their tools. As a result, I've had the pleasure of using both tools extensively over the course of several years. Because of its long-term availability, its stability in C++, its spell checking feature, and that cute little pink and blue highlighting trick I mentioned, I currently use Visual Assist X for my driver development projects. But I use ReSharper for the C# applications that I write. And I have both products installed, and I do periodically try the latest and greatest features of each.

In Summary

So where does that leave us? If you didn't know this before, you now know that if there are things about Visual Studio that annoy you, or seem to work less than optimally, there are numerous customizations available that may be able to help. I've found that it's totally worth spending a couple of hours playing with some of the major VS settings to "improve" my every day experience of editing.

In addition, you don't have to be satisfied with the tricks that Visual Studio knows out of the box. There are several Extensions that make good sense for driver developers. One must-have Extension is Productivity Power Tools. It's free and it's undeniably helpful. The "big names" in VS Extensions, namely Resharper C++ and Visual Assist also work well in the world of drivers. They are both ridiculously rich in terms of features, and your choice will likely come down to personal preference. You'll need to download **and spend time customizing** and using each one before you'll know if it's something you'll want to add to your everyday work environment.

But, regardless of which Extensions you use, keep in mind that you owe it to yourself to spend the time needed to get your development environment setup to your liking. Once you do, every day you spend coding will have fewer little annoyances. And that's **got** to be a win.

Follow us!

WINDOWS FILE SYSTEM TRAINING

File system development for Windows is complex and it isn't getting any easier. Filtering file systems is more common, but is frankly MORE complex - especially if you don't understand the not so subtle nuances in the Windows file system "interface" (ask us what that means!). Instructed by OSR partner and internationally-recognized file system expert, Tony Mason—this is your chance to learn from his many years of practical experience!

I needed to learn as much as I could, and this was the right choice. I have a stack of books, but a classroom experience was much more direct and an efficient way to learn the material. I have already felt the value of this seminar in my day-to-day work.

- Feedback from an attendee of THIS seminar

Next Presentation: Palo Alto, CA 5-8 April

 © OSR Open Systems Resources, Inc.

KMDF Errors... (Cont.)

(CONTINUED FROM PAGE 7)

And of course, an arbitrary thread context is not acceptable when you call **MmMapLockedPagesSpecifyCache**, given that the entire point of this call (when the *AccessMode* parameter is set to **UserMode**) is to map the buffer into the user-mode address space of a given process. So, in order to be able to call **MmMapLockedPagesSpecifyCache** effectively, the caller (the driver's EvtIoXxxs routine in this case) must be running in the context of the process into which the memory is to be mapped. It's not like this is a secret either, as the WDK docs for the function clearly show in the Remarks section:

> If *AccessMode* is **UserMode**, be aware of the following details:
> …
> The routine returns a user address that is valid in the context of the process in which the driver is running. For example, if a 64-bit driver is running in the context of a 32-bit application, the buffer is mapped to an address in the 32-bit address range of the application.
> …

To further emphasize this fact, there's a specific EvtIo Event Processing Callback this is guaranteed to be called in the context of the requesting process. That routine is **EvtIoInCallerContext**. So, there is a way to get the behavior that the dev of the code in Figure 1 is seeking, it's just that that way is not from within the **EvtIoRead** Event Processing Callback.

So, you may ask, if **EvtIoRead** runs in an arbitrary process context, and **MmMapLockedPagesSpecifyCache** relies on running in a specific process context to be effective, how does the code shown in Figure 1 **ever** work?

The answer to that is: It only works because of the way KMDF happens to currently be currently written. **If** your driver is called directly from a user-mode application, and **if** several other conditions apply, KMDF can wind-up calling your driver in the context of the process that sent the I/O Request. You can see the exact sequence of events, and the conditions under which KMDF will call your driver in the context of the requesting process, in the WDF source code on GitHub. If you're curious about the details, walk through the code for the method FxIoQueue::DispatchEvents and you'll see exactly what I'm talking about.

You can test your driver all day, and it'll work. But if there's a change in how your driver code is executed – like, for example, you device hasn't been powered-up yet when a Request arrives – or if you make what seems to be a small change in your driver's code – let's say you add a SynchronizationScope to your Queue, change your Queues Dispatch Type, or you decide to allow your device to automatically idle in low power state – all of a sudden, your driver no longer works.

That's not the extent of the problem, either. Your driver's behavior could become broken very easily by some small change to KMDF by the development team.

And because the code you've written is fragile, cutting and pasting it into some other driver, where the implementation constraints that happen to be present in your original driver that allow it to work may not necessarily be true, will propagate the problem and potentially yield a non-working solution.

So the moral of the story is: EvtIoXxx routines (except EvtIoInCallerContext, mentioned above) are called in **arbitrary** process and thread context. The fact that your driver happens to be called sometimes in the specific thread context that you **want does not change this fact**. You have to follow the architectural rules. Proper operation doesn't mean correct implementation.

Errors Involving IRQL Assumptions

My second-favorite bug, and – again – one that we see all the time, is making invalid assumptions about the IRQL at which various KMDF callbacks will run. We most frequently see this in EvtIoXxx Event Processing Callbacks and in Request Completion Routines. By default, both EvtIoXxx Event Processing Callbacks and Request Completion Routines are architecturally defined to run by default at any IRQL less than or equal to IRQL DISPATCH_LEVEL. Despite this, the drivers that we review often assume that these routines will run at IRQL PASSIVE_LEVEL.

These assumptions take many forms. Sometimes the code for these routines is defined to be pageable (arrgh… a particular hot-button of mine). Sometimes, you see this code serialize access to shared data structures with a synchronization mechanism, such

(CONTINUED ON PAGE 23)

 © OSR Open Systems Resources, Inc.

KMDF Errors... (Cont.)

(CONTINUED FROM PAGE 22)

as a mutex, that assumes the code is running at an IRQL less than IRQL DISPATCH_LEVEL. Or, sometimes, we see something in the **EvtIoRead** Event Processing Callback like the code in *Figure 2*:

```
// Send the request synchronously with an arbitrary timeout of 60 seconds
//
WDF_REQUEST_SEND_OPTIONS_INIT(&SendOptions,
                              WDF_REQUEST_SEND_OPTION_SYNCHRONOUS);

WDF_REQUEST_SEND_OPTIONS_SET_TIMEOUT(&SendOptions,
                                     WDF_REL_TIMEOUT_IN_SEC(60));

Status = WdfRequestAllocateTimer(IoctlRequest);

if (!NT_SUCCESS(Status)) {
    goto TestReadWriteEnd;
}

if (!WdfRequestSend(IoctlRequest, IoTarget, &SendOptions)) {
    Status = WdfRequestGetStatus(IoctlRequest);
}

if (NT_SUCCESS(Status)) {
    *IoTargetOut = IoTarget;
}
```

Figure 2

In Figure 2, you see the driver calling **WdfRequestSend** having previously specified that the Send operation be performed synchronously. Two minutes of thought will bring to your attention the fact that, in order for the Send operation to be synchronous, the caller will have to wait. That wait will be on a Windows Dispatcher Object, and waiting on a Dispatcher Object requires that the caller be running below IRQL DISPATCH_LEVEL. Ooops.

Requirements

Target platform	Universal
Minimum KMDF version	1.0
Minimum UMDF version	2.0
Header	Wdfio.h (include Wdf.h)
IRQL	<= DISPATCH_LEVEL (see Remarks section)

Figure 3

Again, there's no reason that dev who owns this code should reasonably assume that their driver's **EvtIoRead** Event Processing Callback function will be called at IRQL PASSIVE_LEVEL. This is very clear in the WDK documentation. It's even in the cute little table they provide on the page for that DDI (See *Figure 3*).

And, to be clear, there's a method that a driver dev can use that will guarantee that their EvtIoXxx Event Processing Callbacks will execute at IRQL PASSIVE_LEVEL, if that's what they want. That method is to establish an **ExecutionLevel** Constraint

(CONTINUED ON PAGE 24)

OSR'S CORPORATE, ON-SITE TRAINING
Save Money, Travel Hassles; Gain Customized Expert Instruction
We can:

- Prepare and present a one-off, private, on-site seminar for your team to address a specific area of deficiency or to prepare them for an upcoming project.
- Design and deliver a series of offerings with a roadmap catered to a new group of recent hires or within an existing group.
- Work with your internal training organization/HR department to offer monthly or quarterly seminars to your division or engineering departments company-wide.

 © OSR Open Systems Resources, Inc.

KMDF Errors... (Cont.)

(CONTINUED FROM PAGE 23)
of **WdfExecutionLevelPassive** prior to creating the WDFDEVICE or WDFQUEUE. The WDK docs even mention this explicitly.

However, in most cases, you'll be able to test and run your driver for days (or longer!) and you'll never see EvtIoRead called at anything other than IRQL PASSIVE_LEVEL. Why? Because that happens to be the way KMDF is currently written. But… make a small change in your code, and you can wind-up with a blue screen. For example, in most cases setting Synchronization Scope will (by default) cause your driver's EvtIoXxx functions to be called at IRQL DISPATCH_LEVEL. Surprise! Ooops.

So, what's the ultimate lesson here? EvtIoXxxx Event Processing Callbacks can run at any IRQL less than or equal to IRQL DISPATCH_LEVEL. And this is true regardless of how many times you try it, and your callbacks happen to execute at IRQL PASSIVE_LEVEL. If you want your Event Processing Callbacks to run at IRQL PASSIVE_LEVEL, all you need to do is establish an ExecutionLevel Constraint of IRQL PASSIVE_LEVEL, and you're architecturally in the clear.

Errors Involving I/O Targets

I could, and I think at some point I have, written entire articles on the many ways that WDF I/O Targets can be unintentionally abused and misused. But, by far, the most common error we see is that shown in *Figure 4*.

The code in Figure 4 is either 100% correct or 90% incorrect. And on what does this depend? It depends on whether the I/O Target that's being used is a Local I/O Target or a Remote I/O Target.

The architectural rules for **WdfRequestFormatRequestUsingCurrentType** require the I/O Target that's being used to be the driver's Local I/O Target **only**. It is not valid to call **WdfRequestFormatRequestUsingCurrentType** on a Request that's being sent to a Remote I/O Target. Again, the WDK docs for this function make this abundantly clear:

> The **WdfRequestFormatRequestUsingCurrentType** method formats a specified I/O request so that the driver can forward it, unmodified, to the driver's local I/O target.

When you want to forward a Request to a Remote I/O Target, you always need to format it explicitly for a particular I/O Target. This means you'll need to call a WDF I/O Target function, one that starts with WdfIoTargetFormatRequestForXxxx. You're not allowed to take the **WdfReuqestFormatRequestUsingCurrentType** shortcut, if the I/O Target is a Remote I/O Target.

But there's more. In many cases, you're not allowed to specify _SEND_AND_FORGET when you're sending a Request to a Remote I/O Target. In this particular case, the WDK docs for this option really aren't so very clear on this point. But the correct documentation is indeed provided:

> Your driver *cannot* set the WDF_REQUEST_SEND_OPTION_SEND_AND_FORGET flag in the following situations:

(CONTINUED ON PAGE 25)

```
WDF_REQUEST_SEND_OPTIONS_INIT(
     &RequestSendOptions,
     WDF_REQUEST_SEND_OPTION_SEND_AND_FORGET);

 WdfRequestFormatRequestUsingCurrentType(Request);

 Result = WdfRequestSend(
     Request,
     devContext->MyIoTarget,
     &RequestSendOptions);

 if (Result == FALSE) {

     Status = WdfRequestGetStatus(Request);

     WdfRequestComplete(Request, Status);
 }
```

Figure 4

 © OSR Open Systems Resources, Inc.

KMDF Errors... (Cont.)

(CONTINUED FROM PAGE 24)

…

- The driver is sending the I/O request to a remote I/O target and the driver specified the **WdfIoTargetOpenByName** flag when it called **WdfIoTargetOpen**.
- The driver is sending the I/O request to a remote I/O target, and the driver specified both the **WdfIoTargetOpenUseExistingDevice** flag and a **TargetFileObject** pointer when it called **WdfIoTargetOpen**.

…

The big one to notice here is the first restriction, above. Many times a Remote I/O Target is opened, it's opened by name.

But, as with our previously described cases, when this code works for a given situation it will almost always stay working indefinitely. The reason is because of an optimization in KMDF. But if the environment changes even just slightly, let's say your Remote I/O Target gets a filter driver loaded on top of it, you'll get a blue screen. Not a good thing for throughput, that.

The lesson from this third class of error is to be very careful, and very particular, about how you use WDF I/O Targets. Read the docs carefully. Be sure that the WDK documentation explicitly allows the combination of DDIs and operations you're attempting to perform. And whenever you're using Remote I/O Targets, you need to specifically format the Request for that specific I/O Target using a call of the form WdfIoTargetFormatRequestForXxxx.

Catching the Problems Before They Ship

The problem with the errors I've described is that, as we discussed, no amount of testing will typically reveal them. However, in some cases, there are methods (short of a code review by an experienced Windows driver developer) that **might** help you to find them.

Your best chance to find these errors is to follow best practices for driver validation. For static validation, these practices are:

- Always run Code Analysis on your driver code. Set Code Analysis to run on your driver at the "Microsoft All Rules" level every time you do a build, in both Debug and Release configurations.
- Run Static Driver Verifier (SDV) with the "All" Rule Set enabled, as soon as your driver is reasonably working, and frequently during your development process. Yes, SDV **did** at one point suck. Yes, a few years back, it **was** a time wasting exercise. However, SDV has been worth its weight in gold for at least the past few years.

(CONTINUED ON PAGE 26)

OSR CUSTOM SOFTWARE DEVELOPMENT

I Dunno...These Other Guys are Cheaper...Why Don't We Use Them?

Why? We'll tell you why. Because you can't afford to hire an inexperienced consultant or contract programming house, that's why. The money you think you'll save in hiring inexpensive help by-the-hour will disappear once you realize this trial and error method of development has turned your time and materials project into a lengthy "mopping up" exercise...long after your "inexpensive" programming team is gone. Seriously, just a short time ago, we heard from a Turkish entity looking for help to implement a solution that a team it previously hired in Asia spent *two years* on trying to get right. Who knows how much money they spent—losing two years in market opportunity and still ending up without a solution is just plain lousy.

You deserve (and should demand) definitive expertise. You shouldn't pay for inexperienced devs to attempt to develop your solution. What you need is fixed-price solutions with guaranteed results. Contact the OSR Sales team at sales@osr.com to discuss your next project.

 © OSR Open Systems Resources, Inc.

KMDF Errors... (Cont.)

(CONTINUED FROM PAGE 25)

If you are not using both of these static analysis tools in detail, not only are you not following best practices, you're not following the minimum standard for doing acceptable kernel-mode engineering. And when you **do** run these tools, take the time to carefully review their results. Neither of these tools is like Lint. Their goal is not to throw-up 65 million false positives so that you have to wade-through a ton of crap. If you take the time to really examine the results from these tools, you'll find that more often than not they're smarter than you are. We have a famous case here at OSR where one of our devs spent **40 minutes** trying to find the bug that SDV was complaining about. And when he did find it, it was a real, actual, bug – and damn subtle as well.

In terms of dynamic testing, you absolutely must test your code with WDF Verifier enabled. Just turn WDF Verifier on for your driver on your target test machine, and leave it on during your development process. Enable Verbose tracing, and trace at the "Warnings and Errors" level. Here at OSR, we set the WDF Verifier options in the driver's INF file, so that whenever our test team is exercising our driver code, WDF Verifier is also running.

Together, these tools won't necessarily catch all the architectural errors I've described in article. But the tools are constantly improving. Your best chance of identifying problems that you're not aware of is with these tools. And if the tools report some weird thing that you don't understand, post a question to NTDEV, and the good folks there will try to help you out.

Conclusion

The nature of kernel-mode development is that it is complex and that it assume a lot of prerequisite knowledge. This makes it hard when you don't know what you don't know. The best way to counter these problems is to do whatever you can to make yourself knowledgeable about Windows driver development (may I recommend you take a class?). Read the WDK docs carefully. Do your research. Use all the tools that are available to you.

And, most of all, never assume that just because your driver works when you test it – even if you think you test it "really, really, well" – that your driver is correct. Because, well… maybe it's not.

About Those Examples

Just so we're all clear: The example code containing the errors that we show in the *Most Common KMDF Errors* article are **definitely not** taken from actual customer drivers that we've reviewed. Rather, these are examples that we've taken from the standard set of WDK samples (where the code in question is used correctly) or that we've created specifically for this article to illustrate the common errors that we see. It probably doesn't matter to you, unless you're driver is one of the ones we've reviewed recently. We're pretty sure you don't want your code published in *The NT Insider*, given that we signed an NDA.

Follow us!

NEED TO KNOW WDF?

Tip: You can read all the articles ever published in *The NT Insider* and still not come close to what you will learn in one week in our WDF seminar. So why not join us!

Seminar Outline and Information here: http://www.osr.com/seminars/wdf-drivers/

Upcoming presentations:

Amherst, NH 11-15 January
Amherst, NH 7-11 March

 © OSR Open Systems Resources, Inc.

Peter Pontificates... (Cont.)

(CONTINUED FROM PAGE 5)

Aside from that fact that people got timely feedback, probably the thing that struck me most about Connect is the lack of community-generated comments on the WDK. This either means that everyone is pretty happy with the WDK and that there aren't many problems found, or that people don't know about Connect and are providing their feedback through other channels.

Given that Microsoft has demonstrated their interest in our feedback, I think there's some considerable room for improvement here on all sides. Here are the things that I'd like to see happen:

- There should be a central place where members of the community can report WDK bugs or issues. If that's Connect, fine. But (Connect is so horribly slow and) it really has to be easier to separate the WDK-related problems from the vast collection of VS problems. Maybe that means there should be a separate (public) feedback program for the WDK. Maybe all that's needed is a distinguished keyword or tag in Visual Studio's feedback program that's added by the MSFT reviewer. But there needs to be something.

- There similarly needs to be a place where community members can see the list of problems or issues that are outstanding in the WDK. It would be most helpful if that list was not confined to just the issues submitted via Connect, but rather included issues found in internal test, issues reported via WDK Developer Support, or other methods. This is key to helping folks make the best use of the kit, and have confidence that issues are being heard and handled.

- The community needs to step-up and report bugs and issues when they find them. They need to ensure that the bugs and issues that are reported are reported clearly, in a way that they can be understood and replicated by whoever reads the bug. It never ceases to amaze me how very terrible, how unclear, and how non-specific some of the bug reports I see are. These are bug reports filed by actual devs. If you're a dev, and you're filing a bug report, please think carefully about the data that you would need in order to fix the problem you're reporting. As the Good Book says "Report bugs unto others as you would have them report bugs unto you."

While it's non-traditional, what I'm asking for shouldn't be impossible. These days, there are numerous companies that have their real, live, honest-to-goodness internal bug tracking system online and accessible by the public. I'm not saying these companies don't mark some bugs as "Internal Only" – of course they do. But having this level of openness is a real win in terms of establishing a great relationship between a product team and the community they serve.
Hey… A boy can dream, right?

Peter Pontificates is a regular column by OSR Consulting Partner, Peter Viscarola. Peter doesn't care if you agree or disagree with him, but there's always the chance that your comments or rebuttal could find its way into a future issue. Send your own comments, rants or distortions of fact to: PeterPont@osr.com.

Follow us!

FILE ENCRYPTION SOLUTION FRAMEWORK
Develop Per-File Encryption Solutions Almost Exclusively in User Mode!

The OSR File Encryption Solution Framework (FESF) allows Clients to incorporate transparent, per-file encryption into their products—with the added benefit of development in user mode.

Beta releases of FESF are now available via a limited-access Early Adopter Program (EAP).

Learn more about FESF here, or contact the OSR Sales team at sales@osr.com.

 © OSR Open Systems Resources, Inc.

OSR Seminar Schedule

Seminar	Dates	Location
WDF Drivers II: Advanced	8-11 December	At OSR! Amherst, NH
WDF Drivers I: Core Concepts	11-15 January	At OSR! Amherst, NH
Kernel Debugging & Crash Analysis	25-29 January	Seattle, WA
Internals & Software Drivers	15-19 February	Dulles/Sterling, VA
WDF Drivers I: Core Concepts	2-6 November	At OSR! Amherst, NH
Developing File Systems	5-8 April	Palo Alto, CA

More Dates/Locations Available—See website for details

OSR Seminars

We "Practice What We Teach" For a Reason

When we say "we practice what we teach", this mantra directly translates into the value we bring to our seminars. But don't take our word for it...below are some results from recent surveys of attendees of OSR seminars:

- I've learned everything that I wanted to learn and quite a bit that I did not know I needed.
- I think Scott nicely catered to a diverse audience, some of whom had only taken intro to OS and some who already had experience developing drivers.
- Scott was fantastic. He was very helpful at trying to meet each student's needs. **I will highly recommend him and OSR to my associates.**
- "Peter's **style of teaching is excellent** with the kind of humor and use of objects to give you a visual representation of how things work that was simply amazing.

THE NT INSIDER - You Can Subscribe!

Just send a blank email to join-ntinsider@lists.osr.com — and you'll get an email whenever we release a new issue of The NT Insider.

Join OSRHINTS

Private Training

A private, on-site seminar format allows you to:

- **Get project specific questions answered.** OSR instructors have the expertise to help your group solve your toughest roadblocks.
- **Customize your seminar.** We know Windows drivers and file systems; take advantage of it. Customize your seminar to fit your group's specific needs.
- **Focus on specific topics.** Spend extra time on topics you really need and less time on topics you already know.
- **Provide an ideal experience.** For groups working on a project or looking to increase their knowledge of a particular topic, OSR's customized on-site seminars are ideal.
- **Save money.** The quote you receive from OSR includes everything you need. There are never additional charges for materials, shipping, or instructor travel.
- **Save more money.** Bringing OSR on-site to teach a seminar costs much less then sending several people to a public class. And you're not paying for your valuable developers to travel.
- **Save time.** Less time out of the office for developers is a good thing.
- **Save hassles.** If you don't have space or lab equipment available, no worries. An OSR seminar consultant can help make arrangements for you.