Microsoft Windows Driver Developer Conference 2008

Driver Annotations in Depth
Part II

Donn Terry
Senior SDE
Static Analysis for Drivers
sdvpfdex@microsoft.com

Driver Annotations

- The “basic” annotations are a single identifier usually with “in”-ness or “out”-ness as part of the name
- Driver annotations are too rich for that to scale
- Use __drv_in(<annotation>) (etc.) instead

Problem

'Kinds' of Code

- Not all driver code is kernel mode
- Not all kernel code is driver code
- Choose the proper mode of analysis
  - **__kernel_driver;** For kernel-mode driver code.
    This is the default for PRE*f*ast for Drivers (PFD).
  - **__kernel_code;** For non-driver kernel-mode code
  - **__user_driver;** For user-mode driver code
  - **__user_code;** For non-driver user-mode code
- Place anywhere as a declaration after driverspecs.h (or wdm.h) is included

Design

Develop

Test

Install

Maintain

Problem

Typos

- PFD can check for many simple but common errors
  - Passing an incorrect enum value
    - __drv_strictType, __drv_strictTypeMatch
  - Passing an incorrect pointer to a PVOID
    - __drv_isObjectPointer
  - Constants where variables are needed
  - Variables where constants are needed
    - __drv_constant, __drv_nonconstant

Example

Enums

```
NTSTATUS KeWaitForMultipleObjects(
     __in ULONG  Count,
     __in PVOID  Object[],
     __in
       __drv_strictTypeMatch(__drv_typeConst)
     WAIT_TYPE  WaitType,
     __in
       __drv_strictTypeMatch(__drv_typeConst)
     KWAIT_REASON  WaitReason,
     __in
       __drv_strictType(KPROCESSOR_MODE/enum _MODE,
            __drv_typeCond)
     KPROCESSOR_MODE  WaitMode,
     __in BOOLEAN  Alertable,
     __in_opt PLARGE_INTEGER  Timeout,
     __in_opt PKWAIT_BLOCK  WaitBlockArray);
```

Never confuse WaitType, WaitReason, and WaitMode again

Example

Pointers

```
NTSTATUS
  KeWaitForSingleObject(
    __in __drv_isObjectPointer PVOID Object,
    __in
       __drv_strictTypeMatch(__drv_typeConst)
    KWAIT_REASON  WaitReason,
    __in
      __drv_strictType(KPROCESSOR_MODE/enum _MODE,
      __drv_typeCond)
    KPROCESSOR_MODE  WaitMode,
    __in BOOLEAN  Alertable,
    __in_opt PLARGE_INTEGER Timeout
    );
```

Never pass &p when you meant p again

Examples

Constants

```
UCHAR
  READ_PORT_UCHAR(
    __in __drv_nonConstant PUCHAR Port
    );
```

```
LONG
  KeSetEvent(
    __in PRKEVENT Event,
    __in KPRIORITY Increment,
    __in __drv_constant BOOLEAN Wait
    );
```

Avoid unjustified assumptions

Working smarter

- PFD can check for known errors
- __drv_reportError
  - Some combination of parameters and state isn’t a good idea.
- __drv_preferredFunction
  - There’s a better way.

Example

Checking for errors

```
__checkReturn
__drv_when((PoolType&0x1f)==2 || (PoolType&0x1f)==6,
         __drv_reportError("Must succeed pool allocations are"
         "forbidden. Allocation failures cause a system crash"))
PVOID
  ExAllocatePoolWithTag(
    __in POOL_TYPE  PoolType,
    __in SIZE_T  NumberOfBytes,
    __in ULONG  Tag
    );
```

Avoid illegal parameters and combinations

Design
Develop
Test
Install
Maintain

Example

Preferred Function

```
DECLSPEC_DEPRECATED_DDK         // Use native __int64 math
__drv_preferredFunction("compiler support for 64 bit", "Obsolete")
__inline
LARGE_INTEGER
NTAPI_INLINE
RtlLargeIntegerAdd (
    __in LARGE_INTEGER Addend1,
    __in LARGE_INTEGER Addend2
    );
```

Encourage good coding practice

Problem

Floating point

- If your driver uses floating point you must be very careful to protect the hardware.
- It’s easy to forget that you used it.
- Very hard to find during testing, typically not repeatable, and blue-screen is the usual symptom.
- Can span multiple functions
- __drv_floatUsed

Example

Floating point

```
long
intSqrt(long i)
{
    return (long) sqrt((double)i);
}
```

Example

Floating point

```
long
intSqrt(long i)
{
    return (long) sqrt((double)i);
}


…
if (KeSaveFloatingPointState(b))
{
    … intSqrt(…) …
    KeRestoreFloatingPointState(b);
}
else // deal with error



…
    intSqrt(…) …
…
```

Example

Floating point

```
__drv_floatUsed
long
intSqrt(long i)
{
    return (long) sqrt((double)i);
}

…
if (KeSaveFloatingPointState(b))
{
    … intSqrt(…) …
    KeRestoreFloatingPointState(b);
}
else // deal with error


…
    intSqrt(…) …
…
```

Tip
Transitivity

- Check both sides of contract.
- __drv_floatUsed relies on it to work.
- Used for utility functions with side effects.
  - PFD's single function scope seems a problem.
  - But PFD checks both sides of the contract.
  - Correctly stated contracts solve the problem.
- Use on wrapper functions.

Problem
Memory leaks

- PFD has always checked, but sometimes was noisy
- Checks for using freed memory as well

Memory Leaks

Acquire/Release

- __drv_allocatesMem(): the function (optionally via out parameter) allocates memory
- __drv_freesMem(): the memory is freed (and is no longer accessible)
- __drv_aliasesMem: the memory won’t leak and remains accessible

Memory Leaks

Requirements

- Allocated memory must be:
  - Freed (reach a __drv_freesMem)
  - Aliased by exiting the function (via global, out parameter, or function result).
  - Aliased by reaching __drv_aliasesMem.

- Complex data structures.
  - PFD keeps a “contained by” relationship
  - If allocated memory has not been freed at the end of the function, PFD follows the “contained by” links until it finds a container that exits the function via a global or function result. (Up to 5 levels, which is a lot statically.)
  - If it fails, it’s reported as a leak.

Memory Leaks

“Possibly Leaking” messages

- The “Possibly Leaking” messages indicate that the value reached a call that if annotated with __drv_aliasesMem would not have reported a warning.
- Does the called function really “keep” the value?
  - Yes: fix with annotation (likely will fix a lot).
  - No: you’ve found a leak.

Example

Memory allocation

```
NTKERNELAPI
NTSTATUS
IoCreateDevice(
    __in  PDRIVER_OBJECT DriverObject,
    __in  ULONG DeviceExtensionSize,
    __in_opt PUNICODE_STRING DeviceName,
    __in  DEVICE_TYPE DeviceType,
    __in  ULONG DeviceCharacteristics,
    __in  BOOLEAN Exclusive,
    __out
          __drv_out(__allocatesMem(Memory)) // see the book (deref implied)
           PDEVICE_OBJECT *DeviceObject
    );
```

Detect many leaks

Example

Aliasing memory

```
PDEVICE_OBJECT
__checkReturn
IoAttachDeviceToDeviceStack(
    __in PDEVICE_OBJECT SourceDevice,
    __in
        __drv_in(__drv_mustHold(Memory)
        __drv_when(return!=0, __drv_aliasesMem))
        PDEVICE_OBJECT TargetDevice
    );
```

Reduce false positives

Design
Develop
Test
Install
Maintain

Example

Freeing memory

```
NTKERNELAPI
VOID
IoDeleteDevice(
    __in __drv_freesMem(Memory)
        PDEVICE_OBJECT DeviceObject
    );
```

Don’t access freed memory

Design

Develop

Test

Install

Maintain

Problem
Leaked locks (or other resources)

- “Things” you acquire and release are resources.
- They can “leak” like memory, but the memory annotations don’t quite work for Lock type objects (I tried).
- Resources are also “richer”:
    - Must or never hold. (And no double take/free.)
    - Can be put into/taken out of other objects.
    - Some can be “named”.

Resources

Acquire/Release

- __drv_acquiresResource(kind)
- __drv_releasesResource(kind)
- __drv_acquiresResourceGlobal(kind,param)
- __drv_releasesResourceGlobal(kind,param)

- ‘kind’ is just a name (an arbitrary string)
- ‘param’ is “named by” (when there are many)

Resources
Holding

- __drv_mustHold(kind)
- __drv_neverHold(kind)
- __drv_mustHoldGlobal(kind,param)
- __drv_neverHoldGlobal(kind,param)

- Implements:
  - Exclusivity/Non-recursion
  - Unsafe situations (e.g. IoCompleteRequest)

Resources

Specializations

Exclusive (shorthand)

- __drv_acquiresExclusiveResource(kind)
- __drv_releasesExclusiveResource(kind)
- __drv_acquiresExclusiveResourceGlobal(kind, param)
- __drv_releasesExclusiveResourceGlobal(kind, param)

The cancel spin lock

- __drv_acquiresCancelSpinLock
- __drv_releasesCancelSpinLock
- __drv_mustHoldCancelSpinLock
- __drv_neverHoldCancelSpinLock

The critical region

- __drv_acquiresCriticalRegion
- __drv_releasesCriticalRegion
- __drv_mustHoldCriticalRegion
- __drv_neverHoldCriticalRegion

Example

Acquire/Release

```
__drv_maxIRQL(DISPATCH_LEVEL)
__drv_savesIRQL
__drv_setsIRQL(DISPATCH_LEVEL)
_DECL_HAL_KE_IMPORT
KIRQL
FASTCALL
KfAcquireSpinLock (
    __inout __deref __drv_acquiresExclusiveResource(SpinLock)
    PKSPIN_LOCK SpinLock);
```

```
__drv_maxIRQL(DISPATCH_LEVEL)
__drv_minIRQL(DISPATCH_LEVEL)
_DECL_HAL_KE_IMPORT
VOID
FASTCALL
KfReleaseSpinLock (
    __inout __deref __drv_releasesExclusiveResource(SpinLock)
    PKSPIN_LOCK SpinLock,
    __in __drv_restoresIRQL KIRQL NewIrql
    );
```

Design

Develop

Test

Install

Maintain

Example

Must/Never Hold

```
__drv_maxIRQL(APC_LEVEL)
__drv_mustHoldCriticalRegion
__drv_valueIs(==1)
__drv_when(Wait==0, __drv_valueIs(==0;==1) __checkReturn)
NTKERNELAPI
BOOLEAN
ExAcquireResourceSharedLite (
    __inout __deref __drv_neverHold(ResourceLite)
    __deref __drv_when(return!=0, __drv_acquiresResource(ResourceLite))
    PERESOURCE Resource,
    __in BOOLEAN Wait);
```

Example

Spin lock wrapper

```
VOID
  GetMySpinLock(
    __inout
       __drv_deref(__drv_acquiresResource(SpinLock))
    PKSPIN_LOCK SpinLock
)
{
    (void)KeAcquireSpinLock(SpinLock);
}
```

(Ignoring old IRQL value for clarity.)

Transitive annotations empower checks

Design

Develop

Test

Install

Maintain

Problem
Wrong IRQL

- Some functions can only be called at raised IRQL. Some must never be.
- Some functions can change the IRQL. Some can never do so.
- Some functions can temporarily change the IRQL, some shouldn’t.
- How high is safe?
- Tracking the combinations can be hard.

IRQLs

- Many things can be done wrong.
  - Some are simply losing track of the context.
  - Some are due to incomplete analysis in code changes.
  - Some are not understanding what IRQLs do.
- Static analysis can find many of these, and the better the annotation, the more it can find.

Function changes the IRQL

- __drv_sameIRQL: modifies the IRQL but promises to put it back where it was.
- __drv_raisesIRQL: raises it.
- __drv_setsIRQL: changes it (use rarely).
- __drv_restoresIRQL, __drv_restoresIRQLGlobal: undoes a raise/set.

Required IRQLs

- __drv_maxIRQL: maximum you can call it at.
- __drv_minIRQL: minimum you can call it at.
- __drv_requiresIRQL: just exactly one.
- __drv_functionMaxIRQL: function never exceeds.
- __drv_functionMinIRQL: function never goes below.

Saving

- __drv_savesIRQL, __drv_savesIRQLGlobal
- The “Global” annotations save/restore from a PFD-created location invisible to the program, matching the semantics of some functions.

Example

```
__drv_maxIRQL(DISPATCH_LEVEL)
__drv_minIRQL(APC_LEVEL)
F13();

__drv_requiresIRQL(PASSIVE_LEVEL)
F0();

void F()
{
     …
     F13();
     F0();
     …
}
```

PFD will report an error at the call to F0, although it could be the F13 call that’s wrong.

- If the call to F13 is successful, then the call to F0 can’t be.
- If the call to F0 is required, then F13 must be protected (or not used).
- Usually F13 and F0 are far apart. PFD tries to find the “other one”.

Example

```
__drv_raisesIRQL(APC_LEVEL)
__drv_savesIRQL
int raise();

void lower (__drv_restoresIRQL int i);



__drv_sameIRQL
void F()
{
    …
    old = raise();
    F13();
    lower(old)
    F0();
    …
}
```

No error reported here: this is safe.

Design Develop Test Install Maintain

Example

```
__drv_raisesIRQL(APC_LEVEL)
__drv_savesIRQL
int raise();

void lower (__drv_restoresIRQL int i);

__drv_functionMaxIRQL(PASSIVE_LEVEL)
__drv_sameIRQL
void F()
{
    …
    old = raise();
    F13();
    lower(old)
    F0();
    …
}
```

Error reported: raise() raises to APC_LEVEL, but this function says that’s not OK.

Example

```
__drv_raisesIRQL(APC_LEVEL)
__drv_savesIRQL
int raise();

void lower (__drv_restoresIRQL int i);

__drv_minIRQL(DISPATCH_LEVEL)
__drv_sameIRQL
void F()
{
    …
    old = raise();
    F13();
    lower(old)

    …
}
```

Error reported: we know we’re at DISPATCH (or higher). We can’t *raise* to a lower level.

Example

```
__drv_functionMaxIRQL(APC_LEVEL)
__drv_sameIRQL
void F()
{
    …
    old = raise();
    F13();
    lower(old)
    F0();
    …
}

__drv_functionMaxIRQL(PASSIVE_LEVEL)
void G()
{
        F();
}
```

Error reported in G(): G() should never raise above passive level.

Example

Callbacks

```
typedef
__drv_sameIRQL
__drv_clearDoInit(yes)
__drv_functionClass(DRIVER_ADD_DEVICE)
NTSTATUS
DRIVER_ADD_DEVICE (
    __in struct _DRIVER_OBJECT *DriverObject,
    __in struct _DEVICE_OBJECT *PhysicalDeviceObject
    );

typedef DRIVER_ADD_DEVICE *PDRIVER_ADD_DEVICE;
```

Check for correct implementation

Design

Develop

Test

Install

Maintain

__drv_inTry
Required Exception Handler

- __drv_inTry
  - Code must be inside the body of a structured exception handler (SEH): try/except, try/finally.
- __drv_notInTry
  - Code cannot be inside a SEH body.
- Transitive just like __drv_floatUsed

```
__drv_inTry
__drv_maxIRQL(APC_LEVEL)
NTKERNELAPI
VOID
NTAPI
ProbeForRead (
...

__drv_inTry
void ProbeDWORD(void *p)
{
    ProbeForRead(p, 4, 4);
}
```

Problem
Paged functions

- PAGED_CODE must be used with #pragma alloc_text
- Frequently one or the other is missed
- Not quite an annotation, but a lot like one
- Predates PFD
- Sets __drv_maxFunctionIRQL
- PAGED_CODE_LOCKED needed in some special cases
- Works with (dynamic) Driver Verifier

Tip
Things to remember

- There's more:
  - Read the documents
  - Read the documentation on warning messages as you get them.
- Always use the macros – that's what will be supported
- Stick to the predefined macros
- Annotate for the success case
- Annotate to the design, not the implementation
- Look at issues by line number, not warning number.
- Start early

Design Develop Test Install Maintain

Running PFD

- Preferred: Microsoft Automated Code Review (OACR)
  - Runs automatically in the background
- If needed: stand-alone PREfast command
  - Works from many environments besides the normal build environment.
- Identical versions, but OACR has a richer filter capability, so some warnings may differ.

OACR Customization

Add/modify %INIT%\oacruser.ini (pick one or make your own)

```
[defaults]
;All PFD rules
ErrorNumbers=<level0>;<level1>;<level2>;<level3_PFD_samples>;<level_4_PFD>
```

```
[defaults]
;All rules
ErrorNumbers=<all>
```

```
[defaults]
;Specific ones
ErrorNumbers=<level0>;<level1>;<level2>;281xx;281yy;…
```

- Don’t Forget:
  - oacr set all
- OACR chalk talk later.

Design
Develop
Test
Install
Maintain

On Correctness

- Annotations help with assuring correctness.
- In a recent IEEE Computer article, the writer asserts that for “critical” code, the code should be “obviously correct”.
  - Modulo Hoare Expression proofs or equivalent.
- Windows is critical for business infrastructure.

On Correctness

Code which is not obviously correct…

--- is obviously not correct.

Additional Resources

- Web resources
  - WHDC Web site
    - PRE*f*ast step-by-step
      http://www.microsoft.com/whdc/DevTools/tools/PREfast_steps.mspx
    - PRE*f*ast annotations
      http://www.microsoft.com/whdc/DevTools/tools/annotations.mspx
    - How to Use Function typedefs in C++ Driver Code to Improve PRE*f*ast Results
      http://go.microsoft.com/fwlink/?LinkId=87238
  - Blog: http://blogs.msdn.com/staticdrivertools/default.aspx
  - WDK documentation on MSDN
    - PRE*f*ast for Drivers
      http://msdn.microsoft.com/en-us/library/aa468782.aspx
- Chapter 23 in *Developing Drivers with the Windows Driver Foundation*
  - http://www.microsoft.com/MSPress/books/10512.aspx
- E-mail sdvpfdex @ microsoft.com

Related Sessions

Session	Day / Time
Using Static Analysis Tools When Developing Drivers	Mon. 8:30-9:30
Driver Annotations in Depth: Part 1	Mon. 1:30-2:30
Lab: PRE*fast* for Drivers	Mon. 11-12 and Wed. 8:30-9:30
Lab: Static Driver Verifier for WDM, KMDF, and NDIS	Mon. 5:15-6:15 and Wed. 11-12
Integrating PRE*fast* into Your Build by Using Microsoft Auto Code Review	Tues. 4-5
Using Static Driver Verifier to Analyze KMDF Drivers	Mon. 4-5
Using Static Driver Verifier to Analyze NDIS Drivers	Tues. 9:45-10:45
Using Static Driver Verifier to Analyze Windows Driver Model Drivers	Wed. 9:45-10:45

Questions?