C++ in kernel mode

Roman Beleshev

About the author

solarwinds msp

- Solution architect at SolarWinds (former IASO)
- Online backup and recovery
- 17 years in production C++

About the project

solarwinds msp

- Backup is NOT just copying of files
- Customers don’t need a backup
- Minimize RPO (restore point objective)
- Minimize RTO (restore time objective)
- Driver development required
  - RPO - being released
  - RTO - secret project

User mode vs Kernel mode

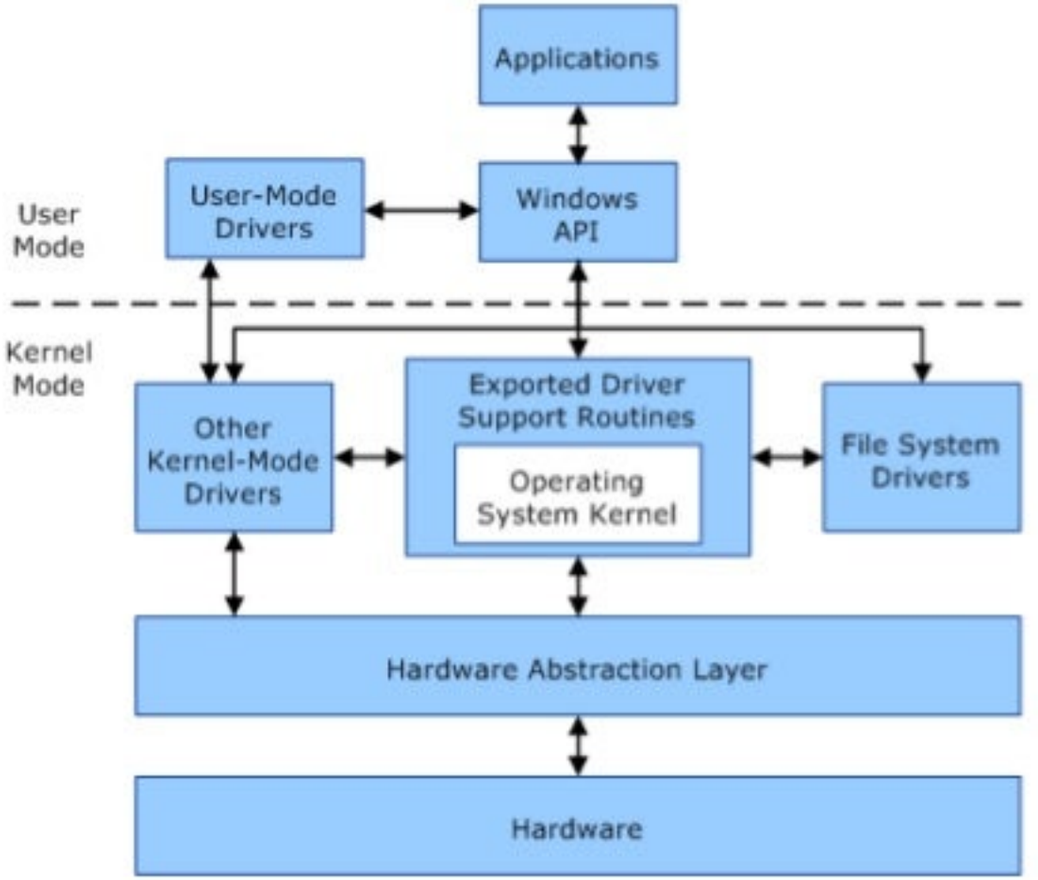
User mode vs kernel mode: background

solarwinds msp

"If builders built buildings the way programmers write programs, then the first woodpecker that came along would destroy civilization"

- Weinberg's Second Law

- Idea: reliability through restrictions
  - no direct hardware access
  - no direct memory access
- CPU enforced (protection rings)
- Similar for most OSes and CPUs



User mode vs kernel mode: relevant differences

solarwinds msp

- Code is mostly API calls
- API is different
  - less functional, more verbose
  - most of libraries are unavailable (including CRT)
- Drivers are callback-driven
- Performance critical
- Restrictions
  - BSOD if something goes wrong
  - IRQL
  - spinlock 25 ms example

From C to C++

Why C++?

solarwinds msp

- Zero-overhead principle
  - What you don’t use, you don’t pay for
  - What you do use, you couldn’t hand code any better
- Strong typing
- Clearer, smaller and better structured code
- Less error-prone code (e.g. RAII)
- Motivates developers

Why C++?

solarwinds msp

```
HANDLE res1 = ::AllocateResource();
if (res1 == INVALID_HANDLE)
{
    return FALSE;
}

HANDLE res2 = ::AllocateResource();
if (res2 == INVALID_HANDLE)
{
    ::FreeResource(res1);
    return FALSE;
}

HANDLE res3 = ::AllocateResource();
if (res3 == INVALID_HANDLE)
{
    ::FreeResource(res2);
    ::FreeResource(res1);

    return FALSE;
}
...

::FreeResource(res3);
::FreeResource(res2);
::FreeResource(res1);
return TRUE;
```

```
BOOL result = TRUE;
HANDLE res1 = ::AllocateResource();
if (res1 == INVALID_HANDLE)
{
    result = FALSE;
    goto end;
}

HANDLE res2 = ::AllocateResource();
if (res2 == INVALID_HANDLE)
{
    result = FALSE;
    goto free_res1;
}

HANDLE res3 = ::AllocateResource();
if (res2 == INVALID_HANDLE)
{
    result = FALSE;
    goto free_res2;
}
...

free_res3: ::FreeResource(res3);
free_res2: ::FreeResource(res2);
free_res1: ::FreeResource(res1);
end: return result;
```

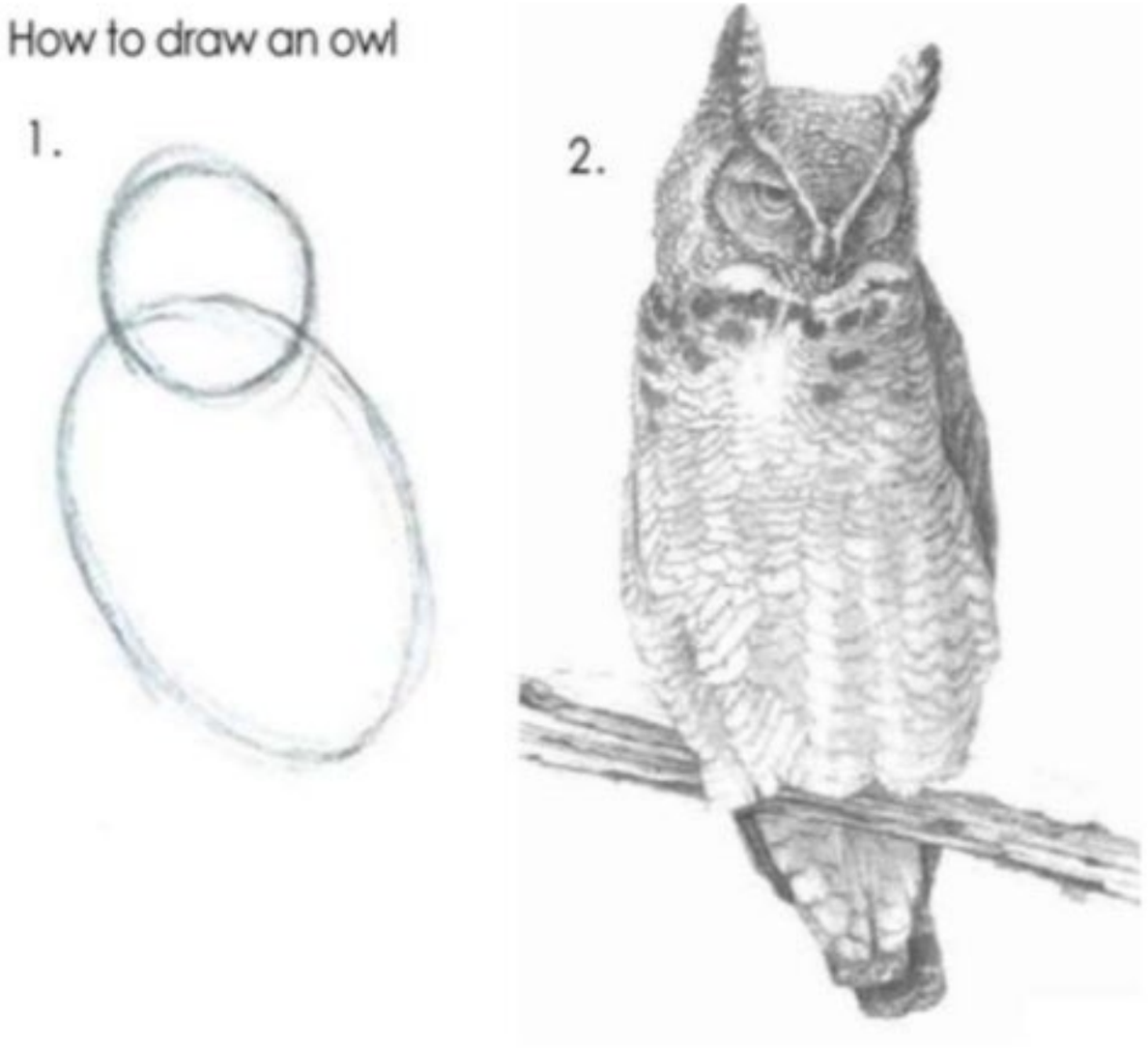
```
Handle res1(::AllocateResource());
Handle res2(::AllocateResource());
Handle res3(::AllocateResource());

...
```

Compile driver in C++? Easy!

solarwinds msp

- Generate Filter Driver project
- Rename *.c file to *.cpp
- Make some corrections :)
  - disable warnings 4510;4512;4610
  - #undef ALLOC_PRAGMA
  - extern “C” DriverEntry



What do we get for free

solarwinds msp

Pure language features and idioms

- automatic construction/destruction
- RAII
- templates
- three pillars of OOP
- strong typing
- lambdas
- constexpr
- many more (ask audience)

What is left

solarwinds msp

- Dynamic memory allocation
- Static variables initialization
- Exceptions
- Libraries
  - CRT
  - STL

Dynamic memory allocation

Dynamic memory allocation: solution

solarwinds msp

Overload new/delete

- globally
- for specific types
- do not forget new[] and delete[]

```
// Kernel-mode allocation routines

PVOID ExAllocatePoolWithTag(
  _In_ POOL_TYPE PoolType,
  _In_ SIZE_T    NumberOfBytes,
  _In_ ULONG     Tag
);

VOID ExFreePoolWithTag(
  _In_ PVOID P,
  _In_ ULONG Tag
);
```

Dynamic memory allocation: nuances

solarwinds msp

- Different pool types may be required
- Performance may be a concern
- Be careful with allocation block size
- Handle no memory case

Static objects initialization

Static variables initialization: problem

solarwinds msp

- Magic statics work
- Need to store global state
- Driver is callback-driven
- No CRT available

```
typedef void (*_PVFV)(void);
typedef int (*_PIFV)(void);

// C initializers
__declspec(allocate(".CRT$XIA")) _PIFV __xi_a[] = { 0 };
__declspec(allocate(".CRT$XIZ")) _PIFV __xi_z[] = { 0 };

// C++ initializers
__declspec(allocate(".CRT$XCA")) _PVFV __xc_a[] = { 0 };
__declspec(allocate(".CRT$XCZ")) _PVFV __xc_z[] = { 0 };

// C pre-terminators
__declspec(allocate(".CRT$XPA")) _PVFV __xp_a[] = { 0 };
__declspec(allocate(".CRT$XPZ")) _PVFV __xp_z[] = { 0 };

 // C terminators
__declspec(allocate(".CRT$XTA")) _PVFV __xt_a[] = { 0 };
__declspec(allocate(".CRT$XTZ")) _PVFV __xt_z[] = { 0 };
```

Static variables initialization: solutions

solarwinds msp

- Singleton(s) based on magic statics
  - how to uninitialize?
- Implement part of CRT
  - looks elegant and native
  - works if there are no initialization parameters
- Manually construct global state object(s)
  - dynamically allocated
  - in-place constructed

Exceptions

Exceptions: SEH

solarwinds msp

- Native Windows mechanism
- Compiler + API
- Performs stack unwinding

```
void WINAPI RaiseException(
  _In_       DWORD     dwExceptionCode,
  _In_       DWORD     dwExceptionFlags,
  _In_       DWORD     nNumberOfArguments,
  _In_ const ULONG_PTR *lpArguments
);

// Whole picture
__try
{
    ::RaiseException(ERROR_CODE, 0, 0, NULL);
}
__except(EXCEPTION_EXECUTE_HANDLER)
{
    PEXCEPTION_POINTERS e = ::GetExceptionInformation();
    HandleException(e->ExceptionRecord->ExceptionCode);
}
```

Exceptions: C++

solarwinds msp

- C++ exceptions are based on SEH
- Throw:
  - allocates memory and constructs exception object
  - wraps C++ exception into SEH exception
  - calls RaiseException
- Exception handler
  - calls destructors
  - filters exception through catch blocks
  - decides if to pass exception

Exceptions: solution (limited)

solarwinds msp

- SEH is unavoidable
- Calling destructors on stack unwind is sufficient
- Turn on SEH compiler option
- Implement `__CxxFrameHandler3`
- Throw using function call
- Catch using SEH syntax in driver callbacks
- Possibly, use <system_error>

Libraries

CRT

solarwinds msp

- Partially available
  - all headers are in place
  - unsafe functions issue linker errors
    - floating point
    - malloc/free, I/O
- Use kernel API (RtlXxx)
- Reimplement or borrow

STL

solarwinds msp

- Some parts depend on CRT (I/O)
- Prognosis: good (for the rest)
- Brute-force attempt failed
- Alternative STL implementation

What's next?

solarwinds msp

- Secret project is in progress
- Kernel-mode framework
- Kernel-mode coding guidelines
- Possibly open-source

Useful links

solarwinds msp

User-kernel modes

https://docs.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/user-mode-and-kernel-mode
https://en.wikibooks.org/wiki/Windows_Programming/User_Mode_vs_Kernel_Mode
https://blog.codinghorror.com/understanding-user-and-kernel-mode/

Global objects initialization

https://msdn.microsoft.com/en-us/library/bb918180.aspx
https://gist.github.com/mmozeiko/ae38aeb10add7cb66be4c00f24f8e688

Exceptions

https://www.codeproject.com/Articles/22801/Drivers-Exceptions-and-C

Questions?

roman.beleshev@solarwinds.com