Almost Unlimited Modern C++ in Kernel-Mode Applications

billy.baker@cox.net

September 25, 2017

Why Are You Here

- Death
- Destruction
- Mayhem
- Panic

Tale as old as time

- Linus Torvalds (2004)

In fact, in Linux we did try C++ once already, back in 1992.

It sucks. Trust me - writing kernel code in C++ is a BLOODY STUPID IDEA.

The fact is, C++ compilers are not trustworthy. They were even worse in 1992, but some fundamental facts haven't changed:

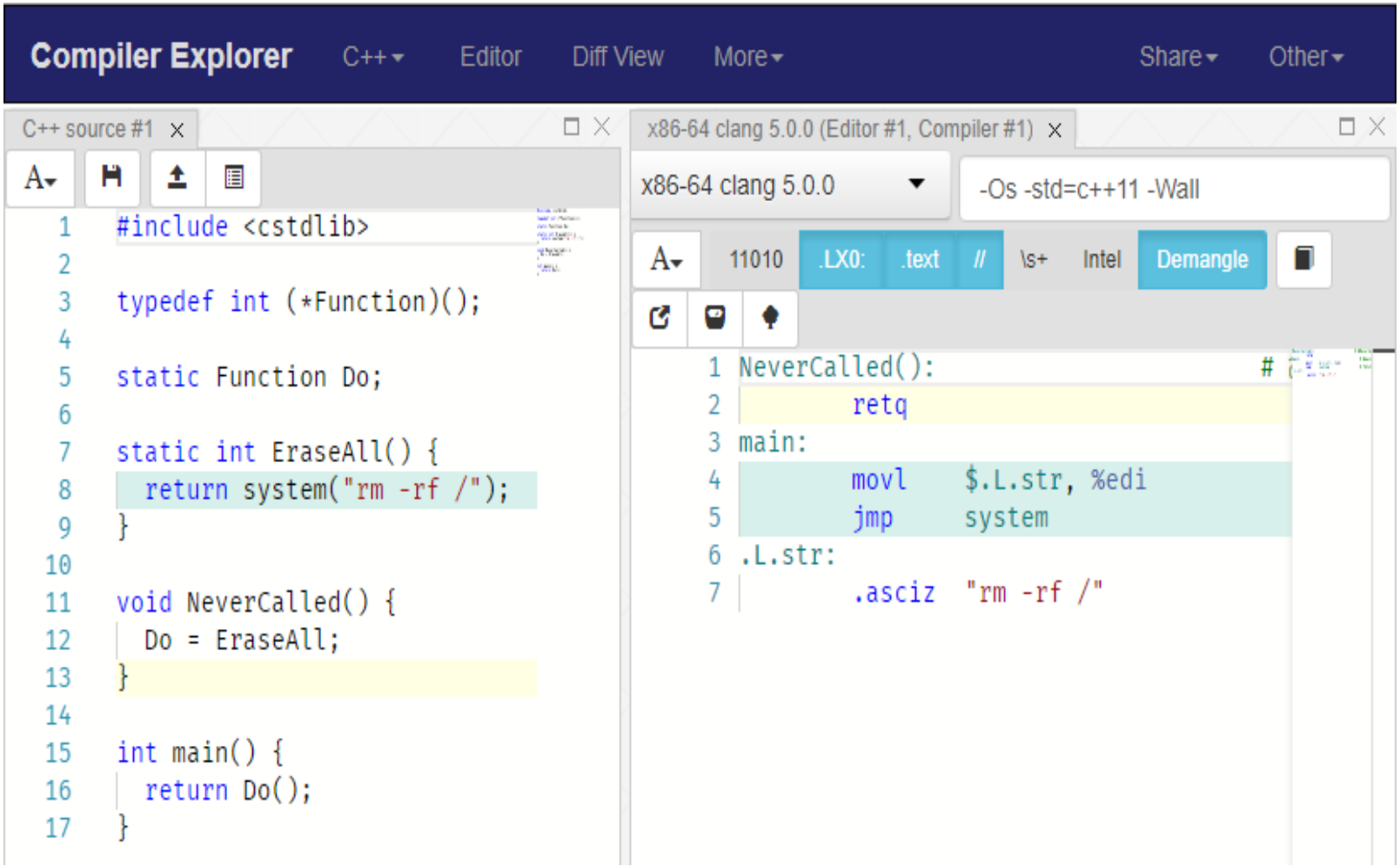
- the whole C++ exception handling thing is fundamentally broken. It's _especially_ broken for kernels.

- any compiler or language that likes to hide things like memory allocations behind your back just isn't a good choice for a kernel.

- you can write object-oriented code (useful for filesystems etc) in C, _without_ the crap that is C++.

---

http://harmful.cat-v.org/software/c++/linus



https://www.reddit.com/r/cpp/comments/6xeqr3/compiler_undefined_behavior_calls_nevercalled/
Wednesday: John Regehr - Undefined Behavior in 2017
Friday: Piotr Padlewski - Undefined Behvaior is awesome!

A Good Year for Thoughts on C++ and Kernel Programming

- Windows (2004)

It is "advanced" C++ features such as non-POD ("plain ol' data", as defined by the C++ standard) classes and inheritance, templates, and exceptions that present problems for kernel-mode code. **These problems are due more to the C++ implementation and the kernel environment than to the inherent properties of the C++ language.**

Anything involving class hierarchies or templates, exceptions, or any form of dynamic typing is likely to be unsafe. Using these constructs requires extremely careful analysis of the generated object code. Limiting use of classes to POD classes significantly reduces the risks.

---

http://bit.ly/1aO1G4r

- Windows (2012)

Visual C++ /kernel option

- Exceptions disabled - compilation errors for try/catch
- RTTI disabled - compilation errors for dynamic_cast and typeid
- Users must replace new and delete
- /arch:IA32 for 32bit
- /arch:AVX not supported for 64bit

A problem has been detected and Windows has been shut down to prevent damage
to your computer.

PFN_LIST_CORRUPT

If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

Technical information:
*** STOP: 0x0000004e (0x00000099, 0x00900009, 0x00000900, 0x00000900)

Beginning dump of physical memory
Physical memory dump complete.
Contact your system administrator or technical support group for further
assistance.

Choice of a new generation

Embedded and real-time options

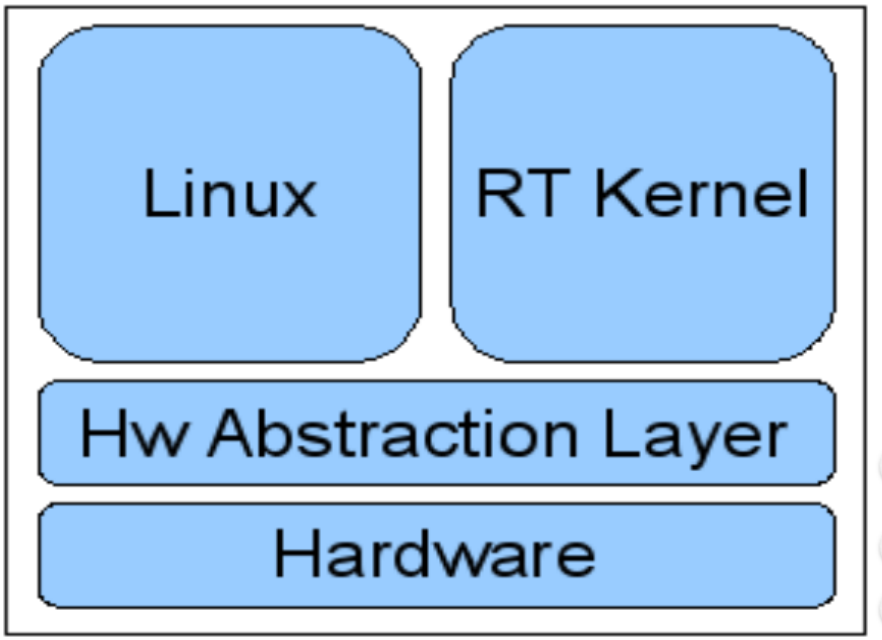
- WindRiver VxWorks - gcc
- Linux - gcc/clang
- GreenHills Integrity RTOS - gcc
- **On Time - Visual C++/Borland C++**
- **Windows/Tenasys InTime - Visual C++/Intel C++/clang**
- **Windows/IntervalZero RTX - Visual C++/Intel C++/clang**
- bare-metal - Borland C++
- AIX
- CX-UX
- Irix
- VMS
- pSOS
- ...

boost

As of 1.64

- **AIX**
- AmigaOS
- BeOS
- BSD
- cloud ABI
- Cray
- Cygwin
- Haiku
- HPUX
- **Irix**
- **Linux**
- MacOS
- QNX Neutrino
- Solaris
- Symbian
- **VMS**
- **VxWorks**
- **Win32**

Linux Real-Time



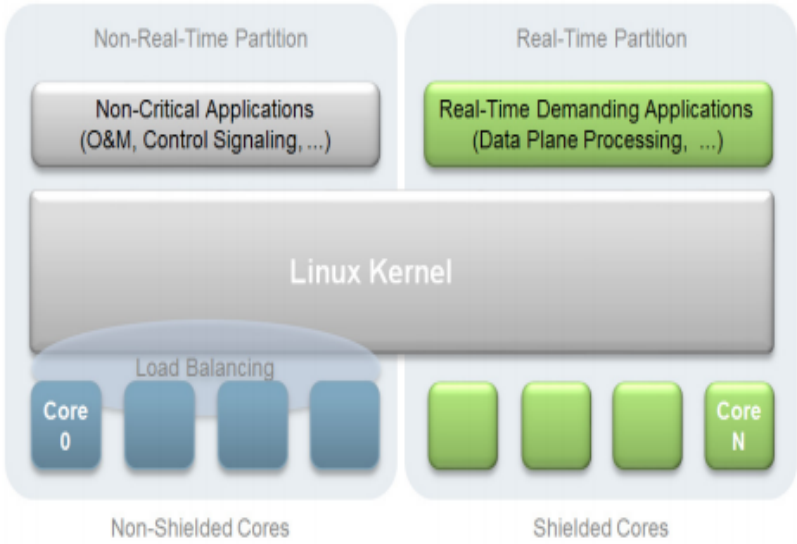


Figure 5. Vertical partitioning with CPU resource shielding

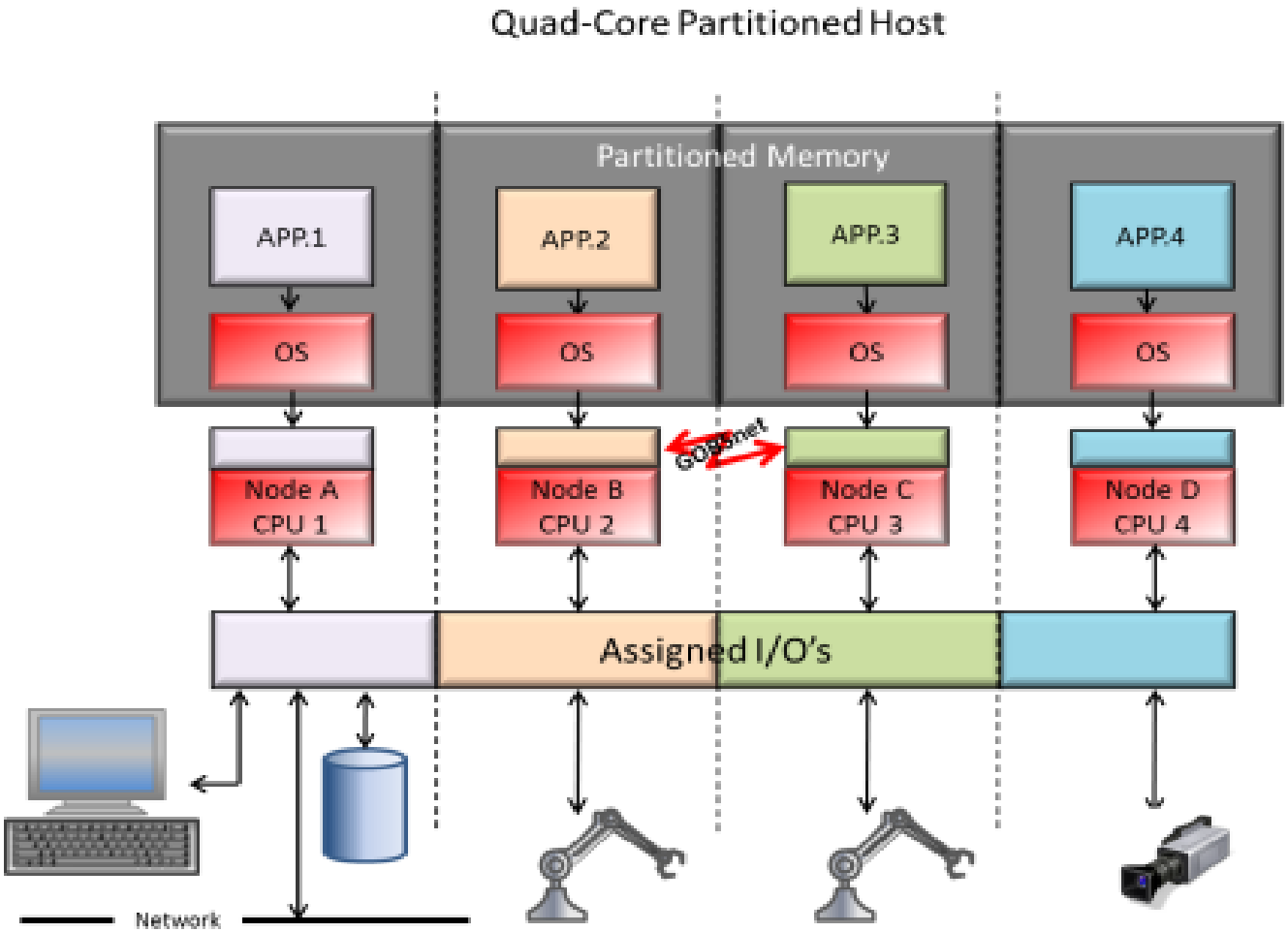
---

http://www.sprg.uniroma2.it/kernelhacking2008/lectures/lkhc08-05print.pdf
https://www.enea.com/globalassets/downloads/operating-systems/enea-linux/enea-enabling-linux-for-real-time-on-embedded-multicore-devices-whitepaper.pdf

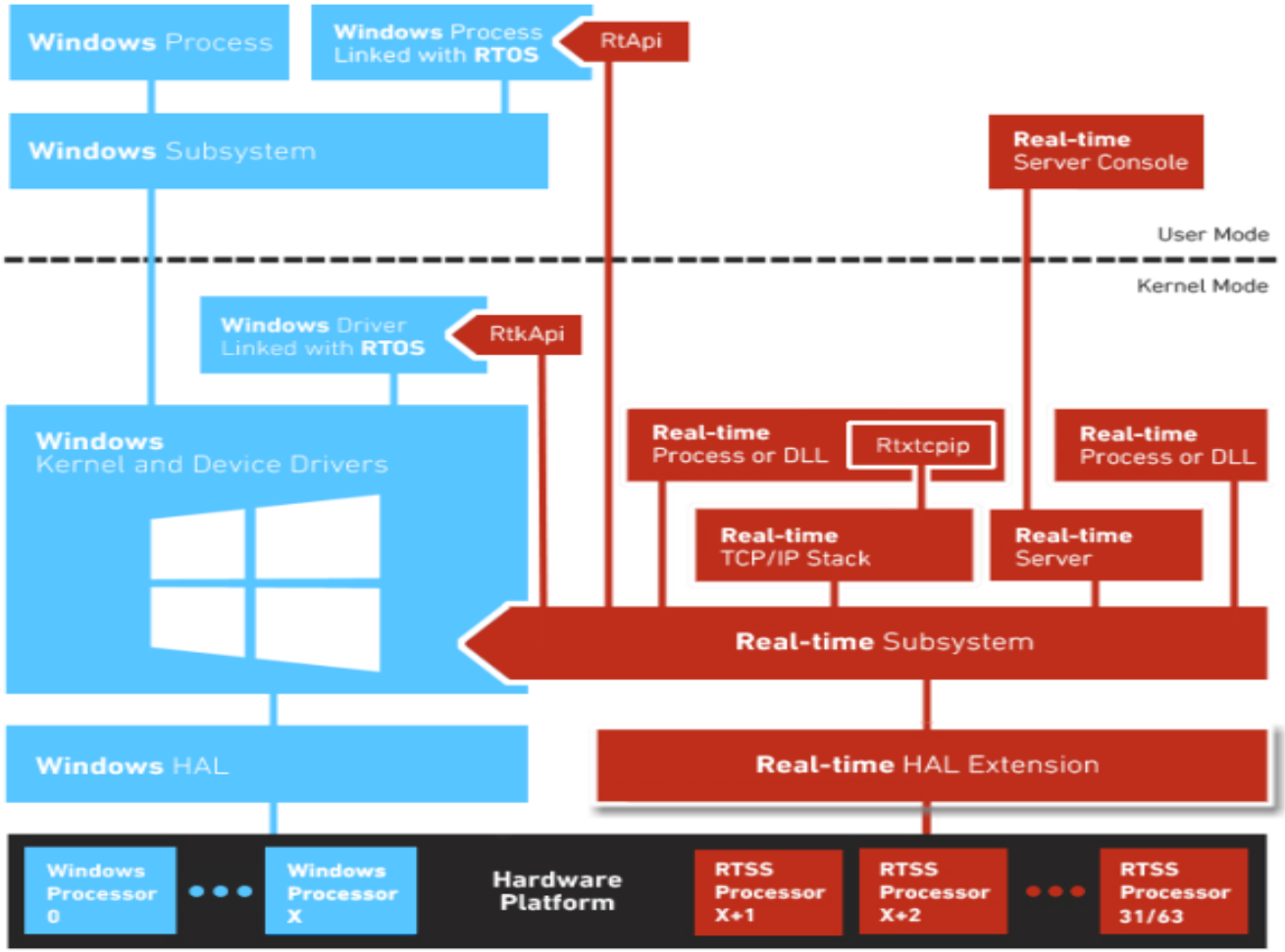
Windows and Real-Time - InTime and RTX

- A lot alike
  - Intel/AMD architecture
  - **CPU segregation/alternate scheduler**
  - no sharing of legacy IRQs
  - custom shared libraries
- With some differences
  - InTime: user-mode applications
  - **RTX: kernel-mode (Ring 0, priviledged) applications**

Windows and InTime



Windows and RTX



Would you like Windows with that?

Our experience is that the industry tendency of using Windows Operating Systems for mission-critical systems is not providing the reliability required. Negative experiences include:

- Host systems requiring to be rebooted more often.
- Intermittent misbehaviors that are very hard to trace and fix.
- Increased instability of the IOS and Lesson Plan systems. Frequent reboots necessary.
- Background work influencing/crashing simulation.
- Single board processors like SOUND or AUDIO running Windows Embedded frequently running out of memory and requiring frequent rebooting.

Whilst we can understand the TDMs wanting to use powerful development environments that Windows offers, cross-compilation allows easy deployment on dependable Linux hosts.

Flight Simulation Engineering and Maintenance Conference (FSEMC)

I Can Do Bad All By Myself

- Toyota brake case - spaghetti code
  - MISRA violations
  - incorrect Watchdog usage
  - concurrency issues
- Nest thermostat

---

https://users.ece.cmu.edu/~koopman/pubs/koopman14_toyota_ua_slides.pdf
http://www.ibtimes.co.uk/google-owned-nest-thermostat-plunges-customers-into-cold-after-software-glitch-1537979

Real-Time C++ Book

- Chris Kormanyos (2013)

  - 4 KB - 1 MB program size
  - 256 byte - 128 KB RAM
  - 8-bit - 32-bit CPU
  - 8 MHz - 200 MHz CPU frequency

1.21 Gigawatts

Think of the possibilities. . .

- up to 64 cores
- gigabytes of memory
- 32-bit and 64-bit CPUs
- gigahertz not megahertz

freestanding vs. hosted

- Headers
  - 87 as of N4659
- Freestanding
  - at least 16 headers
  - <atomic>, <exception>, <initializer_list>, <limits>, <new>, <type_traits>, <ciso646>, <cstddef>, <cfloat>, <climits>, <cstdint>, <cstdlib>, <cstdarg>, <cstdalign>, <cstdbool>
  - implementations can supply more

What Could Go Wrong?

random_device

[rand.device]/2

If implementation limitations prevent generating nondeterministic random numbers, the implementation may employ a random number engine.

```
1  #include <random>
2
3  int main() {
4      try {
5          std::random_device rd;
6          ...
7      } catch (const std::exception&) {
8          // implementation-defined exception
9      }
10 }
```

And the Implementation Says. . .

- Visual C++ 2012 C Runtime
  - rand_s dynamically looks for RtlGenRandom in cryptbase.dll/advapi32.dll
  - No fallback - rand_s returns ENOMEM

```
1 extern "C" BOOLEAN WINAPI SystemFunction036(
2     PVOID buffer,
3     ULONG buffer_count
4     );
```

- Newer Windows SDKs
  - rand_s dynamically looks for SystemFunction036 and expects that it will exist
  - No fallback - abort if not found

But Aren't Shared Objects Different?

- Kernel modules for Linux
- Windows export drivers are kernel DLLs
  - No user-mode code
- RTX
  - can't use Windows DLLs
  - entry points are different

Tuesday: James McNellis - Everything You Ever Wanted to Know about DLLs

Can We Fix It?

- Could try and make rand_s work
  - SystemFunction036 is available in RTX
  - It always appears to return failure
- Or, just make rand_s always return ENOMEM

You Say You Have a filesystem

```
1  #include <string>
2  #include <cstdint>
3
4  #include <sys/types.h>
5  #include <sys/stat.h>
6
7  std::uintmax_t file_size(
8      const std::string& file
9  ) {
10     struct stat s{};
11     stat(file.c_str(), &s);
12     return s.st_size;
13 }
```

- 8 unresolved externals for newer Windows SDK
  - Set/GetCurrentDirectory
  - GetFullPathName
  - PeekNamedPipe
  - FindClose

Where am I?

- No current working directory
  - no relative paths
  - no directory traversal

P0544 - User Injection of Filesystems

- fake filesystems for testing
- in-memory filesystems

---

http://wg21.link/P0544

path - Not Quite in My Vocabulary

- Visual C++ 2012/2013
  - non-TS filesystem implementation
  - path character type = char

- Visual C++ 2015/2017
  - TS filesystem implementation
  - path character type = wchar_t
    - extern function for wide to narrow conversion
    - .obj with conversion function requires lots of unavailable Win32 API functions

Concurrenency Facilities

```
1 #include <thread>
2
3 int main() {
4     std::thread t([] { });
5     t.join()
6 }
```

std::thread

- Visual C++ 2012 - 17 unresolved externals
  - DuplicateHandle
  - mutex and condition variable initialization
- Visual C++ 2017 - 14 unresolved externals

- STL on changes to Visual C++ 2015

We've reimplemented the STL's multithreading primitives to avoid using the Concurrency Runtime (ConcRT). Using ConcRT was a good idea at the time (2012), but it proved to be more trouble than it was worth.

boost::thread

- 3 unresolved externals
  - __imp_WaitForMultipleObjectsEx
    - WaitForMultipleObjectsEx is available
  - __imp_GetLogicalProcessorInformation
    - phyiscal_concurrency
  - __imp_LocalFree
    - from FormatMessage usage in error_core.cpp

system_error

- Why would LocalFree be needed for system_category::message(int)
  - boost allows FormatMessage to allocate
  - Visual C++ makes not one but two allocations with static runtime
    - 32767 char's and 32767 wchar_t's

Stacks

- 12KB stacks for 32bit, 24KB for 64bit - Windows kernel
  - 32KB for Itanium with 32KB backing store
- 8KB on RTX but can be set when creating a thread
- some static analysis can determine large stack usage in a single function

Did I mention implementing thread?

- No stack guard pages
  - explicit stack size
  - commit equal reserve for Windows thread creation
    - STACK_SIZE_PARAM_IS_A_RESERVATION
- No memory protection
  - Usage of AddressSanitizer would require Windows/Linux

---

http://wg21.link/P0320
http://wg21.link/P0484

RTX and C++ Concurrency Issues

- Good
  - atomics
  - lock guards
- Bad
  - async, future, promise, packaged_task
  - condition variable
  - mutex

RTX and C++ Concurrency Issues

- ~~sorry, Gor~~
  - ~~coroutines~~ (see additional slide for more information)
- ???
  - call_once

Thread Safe Initialization

```
1  #include <string>
2
3  struct bar {
4      std::string s;
5  };
6
7  void foo() {
8      static bar b;
9      ...
10 }
```

Is This Safe?

```
?foo@@YAXXZ (void __cdecl foo(void)):
  00000000: 55               push      ebp
  00000001: 8B EC            mov       ebp,esp
  00000003: A1 00 00 00 00   mov       eax,dword ptr [?$S1@?1??foo@@YAXXZ@4IA]
  00000008: 83 E0 01         and       eax,1
  0000000B: 75 26            jne       00000033
  0000000D: 8B 0D 00 00 00 00 mov      ecx,dword ptr [?$S1@?1??foo@@YAXXZ@4IA]
  00000013: 83 C9 01         or        ecx,1
  00000016: 89 0D 00 00 00 00 mov      dword ptr [?$S1@?1??foo@@YAXXZ@4IA],ecx
  0000001C: B9 00 00 00 00   mov       ecx,offset ?b@?1??foo@@YAXXZ@4Ubar@@A
  00000021: E8 00 00 00 00   call      ??0bar@@QAE@XZ
  00000026: 68 00 00 00 00   push      offset ??__Fb@?1??foo@@YAXXZ@YAXXZ
  0000002B: E8 00 00 00 00   call      _atexit
  00000030: 83 C4 04         add       esp,4
  00000033: 5D               pop       ebp
  00000034: C3               ret
```

How about Now?

```
?foo@@YAXXZ (void __cdecl foo(void)):
  00000000: 55                 push        ebp
  00000001: 8B EC              mov         ebp,esp
  00000003: A1 00 00 00 00     mov         eax,dword ptr [__tls_index]
  00000008: 64 8B 0D 00 00 00  mov         ecx,dword ptr fs:[__tls_array]
            00
  0000000F: 8B 14 81           mov         edx,dword ptr [ecx+eax*4]
  00000012: A1 00 00 00 00     mov         eax,dword ptr [?$TSS0@?1??foo@@YAXXZ@4HA]
  00000017: 3B 82 00 00 00 00  cmp         eax,dword ptr __Init_thread_epoch[edx]
  0000001D: 7E 3A              jle         00000059
  0000001F: 68 00 00 00 00     push        offset ?$TSS0@?1??foo@@YAXXZ@4HA
  00000024: E8 00 00 00 00     call        __Init_thread_header
  00000029: 83 C4 04           add         esp,4
  0000002C: 83 3D 00 00 00 00  cmp         dword ptr [?$TSS0@?1??foo@@YAXXZ@4HA],0FFFFFFFFh
            FF
  00000033: 75 24              jne         00000059
  00000035: B9 00 00 00 00     mov         ecx,offset ?b@?1??foo@@YAXXZ@4Ubar@@A
  0000003A: E8 00 00 00 00     call        ??0bar@@QAE@XZ
  0000003F: 68 00 00 00 00     push        offset ??__Fb@?1??foo@@YAXXZ@YAXXZ
  00000044: E8 00 00 00 00     call        _atexit
  00000049: 83 C4 04           add         esp,4
  0000004C: 68 00 00 00 00     push        offset ?$TSS0@?1??foo@@YAXXZ@4HA
  00000051: E8 00 00 00 00     call        __Init_thread_footer
  00000056: 83 C4 04           add         esp,4
  00000059: 5D                 pop         ebp
  0000005A: C3                 ret
```

- Visual C++ 2017 added /Zc:threadSafeInit
  - the runtime calls apparently crash an RTX application

chrono Clocks

```
#include <chrono>
#include <cstdint>
struct clock {
    using duration =
        std::chrono::duration<
            std::int64_t,
            std::ratio<1, 10'000'000>
        >;
    using rep = duration::rep;
    using period = duration::period;
    using time_point =
        std::chrono::time_point<clock>;
    static constexpr bool is_steady = false;

    static time_point now() noexcept;
};
```

TimeStamp Counters as Clock Source

- Fast to read
- Invariant TSC means constant increases
- Not synchronized
  - Large Δt if RTX starts on demand

High Precision Event Timer

- HPET introduced around 2004
- Single 32/64bit counter register
- 3+ comparators
- Not core dependent
    - more cycles needed to read

klaatu barada nikto

```
1  struct hpet {
2      hpet() {
3          // set timer enable = 1
4      }
5      ~hpet() {
6          // leave the system as we found it
7          // set timer enable = 0
8          // write zero to counter
9      }
10 }
```

klaatu barada nikto

```
1  struct hpet {
2      hpet() {
3          // if not enabled
4          //   take "ownership"
5          //   set timer enable = 1
6      }
7      ~hpet() {
8          // leave the system as we found it
9          // if owner
10         //     set timer enable = 0
11         //     write zero to counter
12     }
13 }
```

Exceptions

```
1  #incldue <stdexcept>
2
3  bool bar();
4  void foo() {
5      if (!bar()) {
6          throw std::out_or_range("bar is upset");
7      }
8  }
```

Exceptions With a Little More Determinism

```
1  #incldue <exception>
2
3  struct bar_upset_exception : std::exception {
4      const char* what() const noexcept override {
5          return "bar is upset";
6      }
7  };
8
9  bool bar();
10 void foo() {
11     if (!bar()) {
12         throw bar_upset_exception();
13     }
14 }
```

Friday: David Watson - C++ Exceptions and Stack Unwinding

Throw from a Structured Exception Handler

- /EHa and SetUnhandledExceptionFilter
- 64-bit capture stack backtrace
- see Raymond Chen's discussion on undefined behavior
  - https://blogs.msdn.microsoft.com/oldnewthing/20170728-00/?p=96706

Stacktraces for Exceptions would be Nice

- Antony Polukhin

  - boost 1.6.5 has a stacktrace library

Macro name or default	Effect
BOOST_STACKTRACE_USE_WINDBG	Uses COM to show debug info.
BOOST_STACKTRACE_USE_CACHED	Uses COM to show debug info and caches COM instances in TLS for better performance. Useful only for cases when traces are gathered very often.
BOOST_STACKTRACE_USE_NOOP	Use this if you wish to disable backtracing. stacktrace::size() with that macro always returns 0.

Where is My Code

- Must work with offsets rather than absolute addresses
  - Kernel-mode application loaded at different base address
  - Same issue with address space layout randomization

Modules

- Really should not cause a problem
- Non-technical issue that Visual C++ currently only supports DLL runtimes

Not Just C++

- Fortran
- D
- lua can be linked in RTX kernel applications
  - implemented in C
  - sorting might use non-deterministic functions

Friday: Andreas Weis - Howling at the Moon: Lua for C++ Programmers

Dissection of Unoptimized Fortran

- Math functions
  - Intel Fortran
    - C math function satisfy most symbols
    - various pow functions needed

Dissection of Unoptimized Fortran

- Strings
  - Intel Fortran
    - compare
    - concatenation
    - copying
  - Compaq Fortran
    - no runtime needed

Dissection of Unoptimized Fortran

- Miscellaneous
  - Intel Fortran
    - error reporting for bounds checking
  - Compaq Fortran
    - no option to generate bounds checks

Memory Protection Extensions (MPX)

- Intel SkyLake and above
  - New registers and instructions for array bounds checking
- Custom thread implementation allows easy creation of necessary resources and programming of registers

---

https://software.intel.com/sites/default/files/managed/9d/f6/Intel_MPX_EnablingGuide.pdf

MPX and Your Compiler

- Added in GCC 5
  - enabled in GCC 6
  - Was it really scheduled for deprecation in 7.0?
- Added in Visual C++ 2015 update 1
  - experimental compiler flag /d2MPX

Performance Monitoring

- No perf, valgrind, Intel PMU, vTune, Windows Performance Analyzer, . . . for kernel-mode application
- You can't measure what you don't measure - Alexandrescu
- You can always do it yourself
  - rdmsr/wrmsr instructions may be used since the application is running in kernel-mode
  - memory controllers performance counters can be accessed directly
  - cache management
  - QoS I/O accessible

Summary

- C++ programminig within the kernel, but not part of the OS, is easier given the right execution environment
- Unless the compiler and/or library explicitly supports the environment, plan for
  - possible boost modifications
  - implementing parts of the standard library or platform runtime
  - living without some functionality

Coroutines

- Visual C++ 2017
  - Sample from coroutines TS (http://wg21.link/N4680 [dcl.fct.def.coroutine]/8) runs fine in RTX
- Using std::future/std::promise would not work

Other Problem Areas with Standard C++

- assert
  - will try to display a message box
- networking TS
  - implementation usage of standard threading/concurrency features would be problematic
  - eventual Visual C++ version would expect a Winsock-compatible stack
  - timers may not be available