ndn-cxx - Feature #5133

Support compilation on Windows

11/09/2020 06:39 PM - Varun Patil

Status:	In Progress	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Varun Patil	**% Done:**	0%
Category:	Build	**Estimated time:**	0.00 hour
Target version:			

Description

Been trying to make this build on Windows for a couple of days. So far, I could get it to compile on MSYS2 with a fair amount of ifdef, and only ~50 odd unit tests fail (I suspect this is due to timer brokenness - working around this should be possible). MSVC may not be too difficult as a next step.

Is this something you would consider a patch for?

History

#1 - 11/09/2020 07:59 PM - Junxiao Shi

I attempted ndn-cxx and NFD on Windows in the past.

- First attempt (not working): https://ndncomm.github.io/NFD-Windows/
- Second attempt (cross-compile on a Linux machine using **MXE**, working): https://github.com/yoursunny/NFD-Windows

Then, **Windows Subsystem for Linux (WSL)** is released, so that I don't see a need to have an .exe build anymore.

- Experience on WSL1: https://yoursunny.com/t/2018/NFD-on-Windows-10-WSL/
- Experience on WSL2: most features work flawlessly, although I don't have a webpage on that

Any reason you can't use WSL?

#2 - 11/09/2020 09:43 PM - Varun Patil

WSL 2 is just Linux on Hyper-V, so that's not really "supporting" Windows IMHO.

Two reasons to do this that I can think of:

- (The obvious one) Native support for all platforms. I don't see any reason to not have this if it is fairly straightforward, also considering that (unfortunately) Windows remains a very popular platform.
- Might bring up inconsistencies that may be hard to identify otherwise. For example, the current build breaks when system_clock does not have nanosecond precision, but this is trivially easy to fix (and painfully hard to detect under MSVC's horrible error reporting). Unless POSIX provides such guarantees (not sure), such inconsistencies may arise on other platforms too.

#3 - 11/09/2020 11:09 PM - Varun Patil

Btw all units tests except one segfault pass now.

#4 - 11/10/2020 03:44 AM - Junxiao Shi

Varun Patil wrote in #note-3:

> Btw all units tests except one segfault pass now.

That's much better state than what I did. I basically disabled the whole test suite.

How many features do you have to delete? (Unix faces, pcap, etc)

#5 - 11/10/2020 09:47 AM - Varun Patil

Compiling only ndn-cxx right now, not NFD, so no pcap dependency (am I missing something here?)

Everything related to UnixTransport had to go due to lack of sockets, and one unrelated test (DestroyWithoutProcessEvents, #3248) segfaults. Everything done, I have 1162 passing unit tests vs 1176 on Linux.

There are some things I'd like to submit a patch for anyway; e.g. ndn-cxx/util/logging.cpp uses system variable environ as a variable name (https://man7.org/linux/man-pages/man7/environ.7.html). For some reason, gcc under Linux allows this, but both MSVC and gcc under mingw64

throw.

#6 - 11/10/2020 09:53 AM - Junxiao Shi

Varun Patil wrote in #note-5:

> Everything related to UnixTransport had to go due to lack of sockets

Windows has AF_UNIX now https://devblogs.microsoft.com/commandline/af_unix-comes-to-windows/ but it may not be useful if there's no NFD.

> Everything done, I have 1162 passing unit tests vs 1176 on Linux.

I think it's good enough overall.

Any ideas on how to have continuous integration builds on Windows? Are you willing to setup a Jenkins builder?
Not having CI means future changes could break Windows compilation.

#7 - 11/10/2020 10:57 AM - Varun Patil

Junxiao Shi wrote in #note-6:

> Windows has AF_UNIX now https://devblogs.microsoft.com/commandline/af_unix-comes-to-windows/ but it may not be useful if there's no NFD.

I'm aware, but this is something I'd want to look at only if we're going ahead with this in the first place.

> Any ideas on how to have continuous integration builds on Windows? Are you willing to setup a Jenkins builder?
> Not having CI means future changes could break Windows compilation.

I've zero experience for Windows CI, but I don't know how the licensing part would work out here. To the best of my knowledge, there is no easy way to run a self hosted slave at reasonable cost.

One way to work around this is to use third party CI, and provide Windows support only as best-effort (e.g. apply fixes only in retrospect if something fails on master). GitHub Actions offers free Windows containers, and I expect it would continue to do so for a long time as Microsoft owns GitHub.

There is one more roadblock. Both ndnsec and unit-tests don't exit properly. This is unlike anything I've ever seen before. When main returns, the program apparently exits, but does not return control to the shell. The interesting part is that while the process exists, as soon as I try to access any information about it (with sysinternals), it shuts down gracefully. Even a blank main.cpp linked with the other files in ndnsec exhibits this strange behavior, so I'm assuming this is a compiler/OS bug. MSVC might hopefully fix this.

#8 - 11/10/2020 11:10 AM - Junxiao Shi

Varun Patil wrote in #note-7:

> > Any ideas on how to have continuous integration builds on Windows? Are you willing to setup a Jenkins builder?
> > Not having CI means future changes could break Windows compilation.
>
> I've zero experience for Windows CI, but I don't know how the licensing part would work out here. To the best of my knowledge, there is no easy way to run a self hosted slave at reasonable cost.

University of Arizona has a site license for Windows Server 2019. It can be used on any university owned computers.
If you are willing to work with Jenkins admin to setup the builder, and more importantly come up with CI scripts, licensing is not an issue.

> There is one more roadblock. Both ndnsec and unit-tests don't exit properly. This is unlike anything I've ever seen before. When main returns, the program apparently exits, but does not return control to the shell. The interesting part is that while the process exists, as soon as I try to access any information about it (with sysinternals), it shuts down gracefully. Even a blank main.cpp linked with the other files in ndnsec exhibits this strange behavior, so I'm assuming this is a compiler/OS bug. MSVC might hopefully fix this.

This is something you have to fix. CI systems rely on exit code to determine whether an execution was successful.

#9 - 11/10/2020 11:24 AM - Varun Patil

> University of Arizona has a site license for Windows Server 2019.

That's great! I could write the CI scripts, but have no idea about how setting up the VM etc. works, so will need help on that part.

> This is something you have to fix. CI systems rely on exit code to determine whether an execution was successful.

Yes yes. I'll try to get the MSVC build going today and update. So far already spent ~10 hours trying to fix this, so giving up on mingw now :(

#10 - 11/10/2020 07:05 PM - Varun Patil

Fixed!

MSVC produces the exact same behavior, so this isn't a compiler bug. Turns out that boost::log::sinks::asynchronous_sink produces a very weird deadlock apparently with some OS process, so switching to the synchronous_sink counterpart fixes the issue. I'm guessing this is a bug in either Boost or Windows.

On the MSVC side, the library itself compiles flawlessly, but tests don't compile due to the this->template fun... syntax that it doesn't support. Is this standard C++?

#11 - 11/11/2020 11:19 PM - Davide Pesavento

- Category set to Build

- Start date deleted (11/09/2020)

#12 - 11/12/2020 12:16 PM - Varun Patil

Patch: https://gerrit.named-data.net/c/ndn-cxx/+/6260

#13 - 11/13/2020 10:46 AM - Davide Pesavento

Hi Varun and thanks for your contribution. A couple of high-level questions before we begin a detailed code review.

- I'm surprised there are compilation issues with msvc, in the past few years they've been very active in supporting the latest C++ standard features. My question is: what do Windows developer typically use nowadays? I haven't used Windows in ~15 years but I have the impression that msvc/VisualStudio is more popular than subsystems like MSYS2 and cygwin.
- I suppose we don't need to care about anything older than Windows 10? How's the 32-bit vs 64-bit situation on Windows today?
- How did you install dependencies such as openssl and boost?

#14 - 11/13/2020 11:05 AM - Davide Pesavento

Varun Patil wrote in #note-5:

> Everything related to UnixTransport had to go due to lack of sockets

Boost 1.75 (should be out next month) added support for UNIX domain sockets on Windows, so we could just depend on that version on Windows. Although it should be easy to #ifdef out the UnixTransport code.

> There are some things I'd like to submit a patch for anyway; e.g. ndn-cxx/util/logging.cpp uses system variable environ as a variable name (https://man7.org/linux/man-pages/man7/environ.7.html). For some reason, gcc under Linux allows this, but both MSVC and gcc under mingw64 throw.

This should not be a problem. The local variable declaration shadows the global-scope variable. What error are you seeing?

#15 - 11/13/2020 11:08 AM - Davide Pesavento

Varun Patil wrote in #note-10:

> On the MSVC side, the library itself compiles flawlessly, but tests don't compile due to the this->template fun... syntax that it doesn't support. Is this standard C++?

As far as I know, yes. It's needed (inside a class template) when the base class is dependent on the template parameter and the function being invoked is also a template.

#16 - 11/13/2020 12:54 PM - Varun Patil

Davide Pesavento wrote in #note-13:

> - I'm surprised there are compilation issues with msvc, in the past few years they've been very active in supporting the latest C++ standard features. My question is: what do Windows developer typically use nowadays? I haven't used Windows in ~15 years but I have the impression that msvc/VisualStudio is more popular than subsystems like MSYS2 and cygwin.

I'm not really a Windows developer (the only reason I use it is it offers twice the battery life on my laptop ¯_(□)_/¯) so I can't comment which one is more popular, but a couple of observations I made are that msvc is certainly way faster and produces leaner binaries than mingw (so I'd say it's "better" in some sense). A point to note that Visual Studio supports mingw, so Microsoft certainly acknowledges it as a good toolchain.

From what I see though, several popular cross platform packages seem to prefer mingw for the sake of consistency. Supporting MSVC is definitely a headache if you plan on using compiler-specific features and attributes, which have a lot of overlap between clang and gcc.

- I suppose we don't need to care about anything older than Windows 10? How's the 32-bit vs 64-bit situation on Windows today?

Yep, Windows 7 EOL was 1/14/20 and 8.1 is coming in 2023 (but who uses this anyway?). Don't see any point supporting legacy software. I do believe most people are running 64-bit at this point barring a few very low end devices (atom/celeron). Anyway, I think if it compiles on x64 it should work on x86, as we already take care of architecture on the Linux side. ARM might be more interesting.

- How did you install dependencies such as openssl and boost?

MSYS2 has binary packages for both. For MSVC, vcpkg compiles both from source with zero hiccups, which is amazing.

Davide Pesavento wrote in #note-14:

> Boost 1.75 (should be out next month) added support for UNIX domain sockets on Windows, so we could just depend on that version on Windows. Although it should be easy to #ifdef out the UnixTransport code.

Wow that is nice! As I said earlier, this is something I'd want to look at next, but only if we go ahead with this in the first place, for the sake of saving potentially wasted effort.

> This should not be a problem. The local variable declaration shadows the global-scope variable. What error are you seeing?

My bad (did I mention I really hate msvc's terrible error reporting btw). environ is a macro, apparently, from clang,

```
../ndn-cxx/util/logging.cpp:94:15: error: cannot initialize a variable of type 'const char *(*)' with an rvalu
e of type 'char *'
  const char* environ = std::getenv("NDN_LOG_NOFLUSH");
              ^         ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
C:\msys64\mingw64\x86_64-w64-mingw32\include\stdlib.h:707:17: note: expanded from macro 'environ'
#define environ _environ
                ^
C:\msys64\mingw64\x86_64-w64-mingw32\include\stdlib.h:323:21: note: expanded from macro '_environ'
#define _environ (* __MINGW_IMP_SYMBOL(_environ))
                    ^
C:\msys64\mingw64\x86_64-w64-mingw32\include\_mingw_mac.h:119:35: note: expanded from macro '__MINGW_IMP_SYMBO
L'
#  define __MINGW_IMP_SYMBOL(sym) __imp_##sym
                                   ^
<scratch space>:47:1: note: expanded from here
__imp__environ
^
1 error generated.
```

Windows also defines all sorts of other nonsense macros like ERROR, ERROR_TIMEOUT, ALTERNATE and what not, many of which I had to undef.

Davide Pesavento wrote in #note-15:

> As far as I know, yes. It's needed (inside a class template) when the base class is dependent on the template parameter and the function being invoked is also a template.

Interesting. That file actually seemed to compile when I removed the this->template (on msvc, that is), but then I hit roadblocks elsewhere. Not very sure what's going on.

#17 - 11/14/2020 12:25 PM - Davide Pesavento

- Tags set to needs-discussion

#18 - 11/25/2020 04:17 PM - Davide Pesavento

- Status changed from New to In Progress

- Assignee set to Varun Patil