THIS IS A PUBLIC DOCUMENT

Monthly meetings for Windows/COFF-related developments

We (Ubisoft & Google & other contributors) would like to organize a monthly call to discuss Windows/COFF developments in LLVM in general. The meeting is open to everyone.

Current time slot is on Thursday at **9:00 am Pacific Time / 12:00 pm Eastern Time / 18h00 Paris Time**. We decided on a meeting every two months, the first one being on April 1st, 2021.

We're using Microsoft Teams as a video-conferencing platform. The meeting can be recorded, if no participants oppose. The minutes will be updated below after each meeting.

Please click on the highlighted link below to join:

Microsoft Teams Meeting

Please click here to join the meeting

(upcoming) Meeting #4 - October 7th, 2021 (tentative)

Meeting #3 - September 2th, 2021

Agenda:

- Removal of global state in LLD?
- Unclean paths in warnings on Windows
- Distributed ThinLTO -> full/relative paths
- UB discovery while migrating from Clang 11->12
- Alignment issue in sanitizer -> https://bugs.llvm.org/show_bug.cgi?id=51693

Attendees:

Alexandre Ganea (Ubisoft), Lambert Clara (Ubisoft), Rudy Pons (Ubisoft), Hans Wennborg (Google), Fangrui Song (Google), Philip D'Antonio (Ubisoft), Reid Kleckner (Google), Markus Böck, Saleem Abdulrasool (Google)

Summary:

- Saleem suggested that the DirectoryWatcher is ready, needs users and usages. Shortly discussing about Clang-as-a-daemon.
- Saleem brings the subject of librairications of LLVM (as DLLs).
    - How do we annotate the functions as dll imports/exports
    - How do we compile as a DLL (cmake option)
    - Objective is smaller toolchain binaries.
    - Swift uses LLVMSupport, could be a client
    - Joachim mentions LLVMSupport can be used as a standalone lib
    - DLL feature needs to be opt-in
    - Alex asks what would be good granularity of DLLs
    - Saleem mentions 3 major users: compiler, linker, archiver - DLLs need to be designed in consequence
    - Mentions about Clang-as-a-DLL

- Lambert asks what is compiler-a static lib.
    - UBSan cannot be compiled as dynamic lib.
    - Mentions this comment in the code:

```
# Replace the /M[DT][d] flags with /MT, and strip any
definitions of _DEBUG,
# which cause definition mismatches at link time.
# FIXME: In fact, sanitizers should support both /MT and /MD,
see PR20214.
```

    - Mentions that manually compiling /MD and removing a runtime assert works.

- Lambert ask about issue he has regarding alignment issue when using UBSan: https://bugs.llvm.org/show_bug.cgi?id=51693

- Alex asks if anyone know why the Type sanitizer hasn't ever landed.https://llvm.org/devmtg/2017-10/slides/Finkel-The%20Type%20Sanitizer.pdf
    - Mentions that some projects at Ubisoft have disabled -fstrict-aliasing

- Alex mentions some users of Clang on Windows having "unclean" paths in warnings.
    - This issue occurs when local preprocessed files, but compiled remotely
    - Solved it by cleaning the path in Fastbuild

- Also mentions that we found the ThinLTO issue which was causing empty .native.o files. This was caused by a mix of absolute/relative paths when calling the ThinLTO indexing & the later opt/codegen step.
    - No fix yet.

- Reid discusses the potential optimization we could do in LLD
    - Optimize perhaps reading of memory mapped files (prefetching)

- Multi-thread input files process and comdat merging
- Symbol internalization, like debug types
- Alex mentions MOLD & its upcoming COFF driver
- Fangrui discusses the lack of parallel structures, the inability to integrate IntelTBB and possible use of pstl instead.

Meeting #2 - June 10th, 2021

Agenda:

- Debug infos in lambdas
- Performance of sys::fs::status() on Windows
- In-process compilation of multiple TUs
- On Windows, StringMap with a path as key, separator and case insensitive
- LLVM Dlls on Windows https://lists.llvm.org/pipermail/llvm-dev/2021-May/150804.html

Attendees:

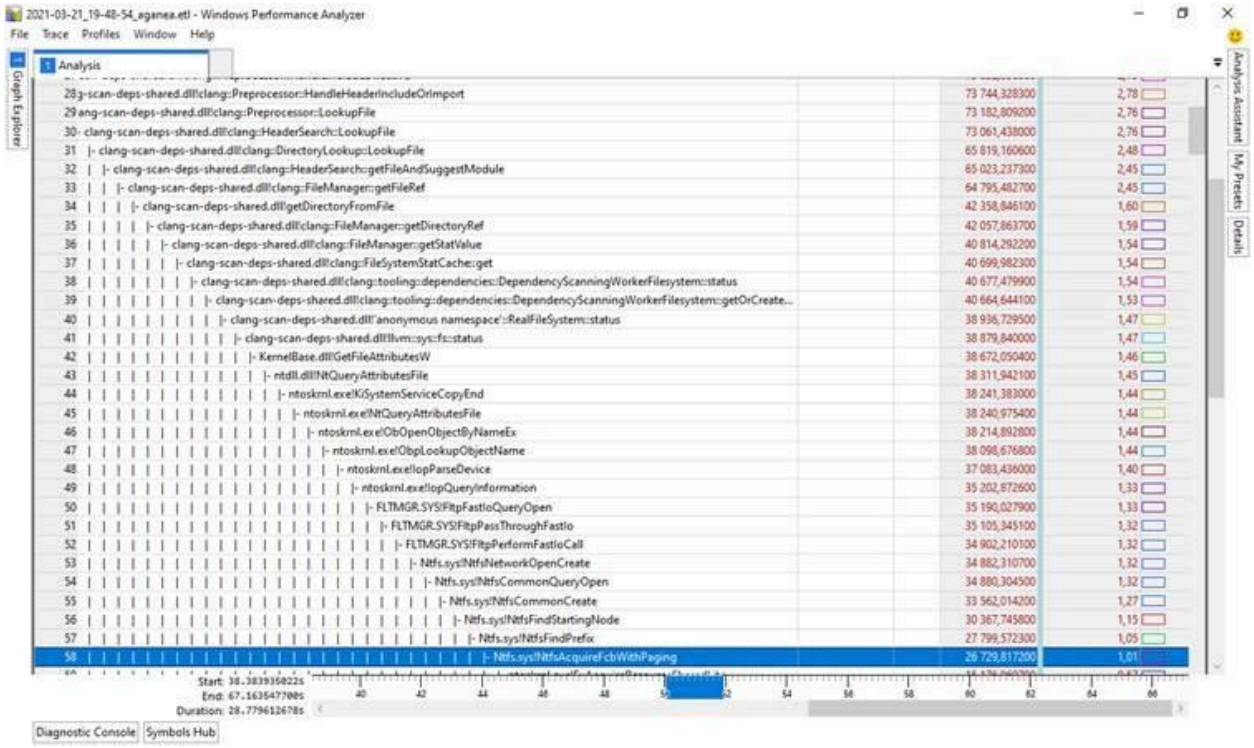
Alexandre Ganea (Ubisoft), Lambert Clara (Ubisoft), Eric Astor (Google), Sylvain Audi (Ubisoft), Rudy Pons (Ubisoft), Martin Storsjö, Hans Wennborg (Google), Fangrui Song (Google), Philip D'Antonio (Ubisoft), Reid Kleckner (Google), Helena Ton-That (Ubisoft), Nicolas Fleury (Ubisoft), Markus Böck, Matheus Izvekov, Florence Cloutier (Ubisoft), Joachim Meyer, Saleem Abdulrasool (Google), Neil Henning (Unity), David D.Racine (Ubisoft), Adrian McCarthy (Google)

Summary:

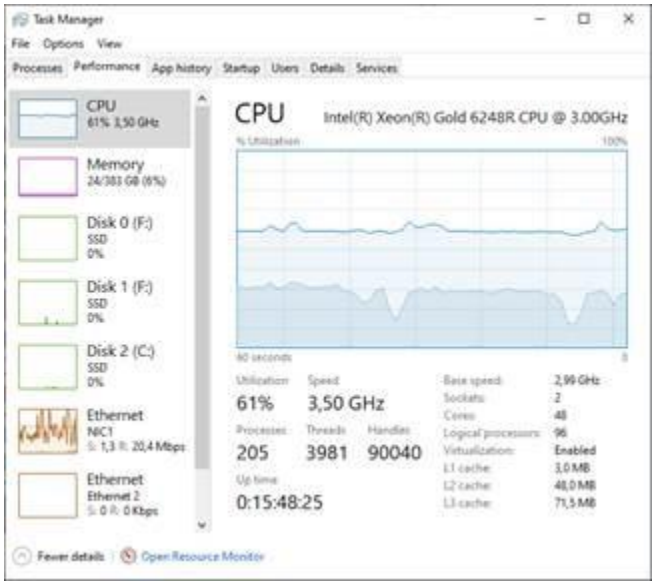
- Neil working on the Burst C# compiler suggests that he has a pipeline where many TUs (thousands) are compiled & linked in-process. Could use a JIT but for now using the regular pipeline.
    - Would like to use LLD in-process, as a library. Currently launching many instances of lld-link.exe is slow because of startup app. time on Windows.
    - Martin suggests that LLD was explicitly designed to be fast, using globals and early exit. Should be kept in mind if converting LLD to be usable as a library
    - Fangrui points out this comment: https://reviews.llvm.org/D85278#2434383
        - Also, suggests that fatal() requires stack unwind.
    - Reid discusses how errors should bubble up in a side state, and only be inspected/used after the process/LLD-as-lib call completes.
    - Follow-up RFC: https://lists.llvm.org/pipermail/llvm-dev/2021-June/151031.html

- Alexandre discusses the “globals” situation in llvm-buildozer where globals were converted to thread_local temporarily for the demonstration purposes. Saleem suggests using LLVM Context or a LLD Context to move out global state into stack-local.

- Alexandre asks about anyone having issues with debug infos in lambdas. Markus suggests there is an issue with external state (captured by the lambda) not being in the debug infos.
- Saleem discusses the mandatory tail call optimizations needed for concurrency support in Swift.
    - Suggests also that the File Watcher patch was reverted -> https://reviews.llvm.org/D88666 - Adrian suggests that we should revisit the patch.
- Alexandre raises up the performance of llvm::fs::stats which is slow on Windows, because it hits NT kernel locks. See the following callstack as an example:



This prevents 100% CPU usage because of the contention in the kernel:



    - Alexandre suggests that there's an existing cache interface, but it is not used. Ubisoft has a implementation for clang-scan-deps (not posted on Phabricator yet)
- Sylvain discusses the storage of file paths in Clang. There are issues related to casing which causes duplicate string in StringMaps.
    - Adrian suggests that we should canonicalize paths internally. This is what was suggested by Reid a while ago. Might be a big endeavour.
    - Eric points out that we could simply convert paths to the platform format when calling the platform APIs.
- Nicolas discusses integration of fmtlib and issues he has with disabled runtime exceptions. Throwing is not possible for constexpr code in that case.
- Matheus discusses a trivially copyable difference between what Clang generates vs. MSVC. This causes an ABI breakage. A PR exists: https://bugs.llvm.org/show_bug.cgi?id=47346
    - Reid suggests that we should run the Microsoft test suite: https://github.com/microsoft/compiler-tests

- Alexandre raises an issue that Ubisoft has with distributed ThinLTO. Symbols are missing in the final link stage, usually globals.
    - Reid suggests that Chromium has a implementation but it is disabled: https://source.chromium.org/chromium/chromium/src/+/main:tools/clang/scripts/goma_link.py;l=695
    - Reid points out an address significance table bug fixed recently: https://reviews.llvm.org/D101512
- Neil says that they've enabled the new pass manager. Discusses issues with supporting different LLVM versions.
    - Reid points out LLVM ABI check macro
    - https://reviews.llvm.org/D104216

- Eric asks if anyone has legacy MASM code, and would be willing to test LLVM-ML with it. Last outstanding patch : https://reviews.llvm.org/D92507
- Talks around code sharing between LLD drivers. Martin suggests that duplication is by design.
    - Matheus discusses linking with GCC object files.
    - Martin discusses linker scripts that will never be supported in LLD.

- Joachim discusses compiling LLVM as DLLs on Windows. Point to recent thread: https://lists.llvm.org/pipermail/llvm-dev/2021-May/150804.html
    - Saleem suggests that LLVM toolchains are big (2 GB+) and that there's lots of binary duplication.
    - Discussions about supporting exporting symbols
    - Reid suggests that we should have a header for explicitly exporting symbols.

Meeting #1 - April 1st, 2021

Attendees:

Alexandre Ganea (Ubisoft), Lambert Clara (Ubisoft), Eric Astor (Google), Sylvain Audi (Ubisoft), Rudy Pons (Ubisoft), Martin Storsjö, Hans Wennborg (Google), Janusz Nykiel (Ubisoft), Fangrui Song (Google), Philip D'Antonio (Ubisoft), Nico Weber (thakis) (Google), Reid Kleckner (Google), Pan Tao (Intel), Helena Ton-That (Ubisoft), Kamal Essoufi (Ubisoft), Nicolas Fleury (Ubisoft)

[Reid] I took some notes by hand, this is the summary:

- Cross-compilation is important, many of the attendees use it.
    - Auxiliary tools (llvm-ml and rc) help make cross-compilation easy
    - Need to teach llvm-rc (and ml? forgot) shell out to clang to pre-process, preferred to more busy-boxing
    - (not discussed) Path issues with the host FS are a major cross-compilation issue for us, maybe others too
- Which driver should we use: clang-cl or clang[++]?
    - Microsoft is using clang-cl in their integration
    - Most attendees use clang-cl, but a uniform syntax across platforms would simplify cross-platform builds

    - CRT selection (/M[TD][d]) is a major missing feature in the GCC-style driver, worth adding
    - Static RTTI (I forget what this is…) is another
- Does upstream care about feature parity? Can I submit a feature for just PE/COFF LLD or the MS C++ ABI in clang?
    - Generally, yes: this happens all the time the other way around (ELF-only features). It's not ideal, but it's a cost of doing business in a cross-platform project.
- Subtarget CPU feature auto-detection and usage:
    - Lambert Clara (Ubisoft) wants to use AVX, SSE2, SSE4.1, etc and have just one EXE
    - ELF has a fancy function multi-versioning feature to dynamically dispatch and auto-select the best implementation for the current CPU
    - On Windows, there is no runtime dispatch, but you can use __attribute__((target("asdf"))) to selectively enable features like AVX etc, and then do the CPUID dispatch yourself. Chrome does this.
    - GNU multiversioning relies on loader support, but it may be feasible to add runtime support for this to compiler-rt to port the feature to Windows.
- Whole program vtables lead to a big bottleneck in LTO
    - Discussed data dependency between analyzing all vtables and every compilation action
    - Perhaps the feature is "fast" for batch links, but unsuitable for incremental development
    - Profiling may shed more light
- Discussion of LTO visibility, and how it's difficult with vtables in third party code that cannot be recompiled
    - Maybe the LTO visibility flags and attributes can be extended, maybe we can add a flag that makes everything LTO public by default, and then attributes mark things as LTO private so the defaults are conservatively correct.
- LF_BUILDINFO vs ghash
    - The LF_BUILDINFO records a command which contains paths, which is used by Live++ to replay compilation actions for individual TUs.
    - GHashing makes it hard/impossible to modify type records in place: changing the record changes the hash, and types cannot be inserted into the middle of the type stream
    - Could skip all LF_BUILDINFO records, make new ones, and add them later after parallel steps
    - Conclusion: maybe it's best to update Live++ and encode the compiler CWD some other way

Later things discussed after the meeting:

- S_OBJNAME: https://reviews.llvm.org/D43002, need to look at it, moving the object file name around is annoyingly difficult with -save-temps
- CI /MP (multi-process) feature
    - Alexandre would like to assign someone to work on this in May
    - Reid would like to add this and use it to implement multi-process ThinLTO
        - /MP for ThinLTO is about debuggability and distributability
        - A crashing single-threaded process is easy to re-run on the same inputs and debug, is more deterministic
        - Adding a prefix (gomacc, icecc, fastbuild, etc) is a way to distribute LTO compile actions
    - However, maybe it's best to use the approach of replacing the linker with a script, as is done here:
        - https://source.chromium.org/chromium/chromium/src/+/master:tools/clang/scripts/goma_link.py
    - Major downside is bad job scheduling: N links * N compiles -> N^2 actions, thrashing, badness.
        - Solution A: hoist actions into the build system, build system becomes ThinLTO-aware
        - Solution B: Some kind of system-global semaphore, but it won't interact with build system compile actions. =/