Funner LLVM development

Nico Weber, @thakis
Goma

https://chromium.googlesource.com/infra/goma/client/
cmake -GNinja -DLLVM_ENABLE_ASSERTIONS=ON -DCMAKE_BUILD_TYPE=Release
-DLLVM_TARGETS_TO_BUILD=X86 ../llvm-rw/ -DLLVM_ENABLE_DIA_SDK=NO
-DCMAKE_C_COMPILER=c:/path/to/bin/clang-cl.exe
-DCMAKE_CXX_COMPILER=c:/path/to/bin/clang-cl.exe
-DCMAKE_C_COMPILER_LAUNCHER=c:/goma/goma-win64/gomacc.exe
-DCMAKE_CXX_COMPILER_LAUNCHER=c:/goma/goma-win64/gomacc.exe
-DCMAKE_C_FLAGS="-m32 -Wno-nonportable-include-path" -DCMAKE_CXX_FLAGS="-m32 -Wno-nonportable-include-path"
LLVM should keep using cmake
IMHO, not great for hacking on LLVM
Slow, so caches. Now needs to solve one of the two hard problems.
cmake -GNinja -DLLVM_ENABLE_ASSERTIONS=ON -DCMAKE_BUILD_TYPE=Release -DLLVM_TARGETS_TO_BUILD=X86 ../llvm-rw/ -DLLVM_ENABLE_DIA_SDK=NO -DCMAKE_C_COMPILER=c:/path/to/bin/clang-cl.exe -DCMAKE_C_COMPILER_LAUNCHER=c:/goma/goma-win64/gomacc.exe -DCMAKE_CXX_COMPILER=c:/path/to/bin/clang-cl.exe -DCMAKE_CXX_COMPILER_LAUNCHER=c:/goma/goma-win64/gomacc.exe -DCMAKE_C_FLAGS="-m32 -Wno-nonportable-include-path" -DCMAKE_CXX_FLAGS="-m32 -Wno-nonportable-include-path"
Build file syntax workable but not fun.
gn: fast, fun
“generate ninja”

Used by Chrome, Fuchsia, ...

https://gn.googlesource.com/gn
https://is.gd/gn_intro
Manually converted LLVM, LLD, Clang to gn
Time to generate .ninja files

- Cmake, cold cache
- Cmake, warm cache
- gn

Seconds
What works?

- Everything needed for check-llvm, check-lld, check-clang
- On Linux, Mac, Win hosts
- Targets X86, ARM, AArch64
- Debug/Release, Asserts on/off, some other build flags
Workflow

---

- gn gen some/dir
- ninja -C some/dir check-lld
- Put something like this in some/dir/args.gn:

  use_goma = true

  clang_base_path = "c:/path/to"
Workflow

---

- gn gen some/dir
- ninja -C some/dir check-llvm
- Put something like this in some/dir/args.gn:

```gn
is_debug = true / false
```

```
gn args --list some/dir` for list of toggles
```
[component_0]
type = Tool
name = llvm-undname
parent = Tools
required_libraries = Demangle Support

set(LLVM_LINK_COMPONENTS
    Demangle
    Support
)
add_llvm_tool(llvm-undname
    llvm-undname.cpp
)
toolchain("posix") {
    cc = "cc"
    if (clang_base_path != "") {
        cc = "$clang_base_path/bin/clang"
    }
    if (use_goma) {
        cc = "$goma_dir/gomacc $cc"
    }
    tool("cc") {
        depfile = "{{output}}.d"
        command = "$cc -MM -MF $depfile -o {{output}} -c {{source}} {{defines}} {{include_dirs}} {{cflags}} {{cflags_c}}"
        depsformat = "gcc"
        description = "CC {{output}}"
        outputs = [
            "{{source_out_dir}}/{{target_output_name}}.{{source_name_part}}.o",
        ]
    }
}
mostly simple
`gn format` means build files are consistently formatted
“configure” step runs at build time! Lld part of build can run while clang configures.
configure bad: serially at start of build & monolithic config.h causes needless rebuilds
Cool features

- Targets can list data deps; easy to zip up all files needed for e.g. “check-llvm”, send to other machine, run tests there
- `gn desc --json` dumps description of build; can convert to bazel BUILD files, Android blueprint, … from there
- Can create MSVC, Xcode, Eclipse, QTCreator... project files (which shell out to ninja for actual building)
- Great support for builds using multiple toolchains (e.g. cross builds, multi-stage builds in one build dir, …)
If you want to try it

---

Get gn as described on https://gn.googlesource.com/gn

In your monorepo:

git remote add nico
https://github.com/nico/llvm-project-20170507

git fetch nico gn && git checkout nico/gn

gn gen out/gn && ninja -C out/gn

(`gn args --list out/gn` to see build toggles)
Keeping gn files in sync annoying? Did it for the last 8 months, no big deal