Cross-compiling Chrome/win

As many Chromium developers are on Linux/Mac, cross-compiling Chromium for Windows targets facilitates development for Windows targets on non-Windows machines.

It's possible to build most parts of the codebase on a Linux or Mac host while targeting Windows. It's also possible to run the locally-built binaries on swarming. This document describes how to set that up, and current restrictions.

Limitations

What does *not* work:

- `js2gtest` tests are omitted from the build (bug)
- on Mac hosts, 32-bit builds don't work (bug has more information, and this is unlikely to ever change)

All other targets build fine (including `chrome`, `browser_tests`, ...).

Uses of `.asm` files have been stubbed out. As a result, Crashpad cannot report crashes, and NaCl defaults to disabled and cannot be enabled in cross builds (.asm bug).

.gclient setup

1. Tell gclient that you need Windows build dependencies by adding `target_os = ['win']` to the end of your `.gclient`. (If you already have a `target_os` line in there, just add `'win'` to the list.) e.g.

```
solutions = [
  {
    ...
  }
]
target_os = ['android', 'win']
```

2. `gclient sync`, follow instructions on screen.

If you're at Google, this will automatically download the Windows SDK for you. If this fails with an error:

```
Please follow the instructions at
https://chromium.googlesource.com/chromium/src/+/HEAD/docs/win_cross.md
```

then you may need to re-authenticate via:

```
cd path/to/chrome/src
# Follow instructions, enter 0 as project id.
download_from_google_storage --config
```

If you are not at Google, you can package your Windows SDK installation into a zip file by running the following on a Windows machine:

```
cd path/to/depot_tools/win_toolchain
# customize the Windows SDK version numbers
python package_from_installed.py 2017 -w 10.0.17134.0
```

These commands create a zip file named `<hash value>.zip`. Then, to use the generated file in a Linux or Mac host, the following environment variables need to be set:

```
export DEPOT_TOOLS_WIN_TOOLCHAIN_BASE_URL=<path/to/sdk/zip/file>
export GYP_MSVS_HASH_<toolchain hash>=<hash value>
```

`<toolchain hash>` is hardcoded in `src/build/vs_toolchain.py` and can be found by setting `DEPOT_TOOLS_WIN_TOOLCHAIN_BASE_URL` and running `gclient sync`:

```
gclient sync
...
Running hooks:  17% (11/64) win_toolchain
________ running '/usr/bin/python src/build/vs_toolchain.py update --force' in <chr
Windows toolchain out of date or doesn't exist, updating (Pro)...
current_hashes:
desired_hash: <toolchain hash>
```

GN setup

Add `target_os = "win"` to your args.gn. Then just build, e.g.

```
ninja -C out/gnwin base_unittests.exe
```

Goma

This should be supported by the default (Goma RBE) backend. However, there may be issues with arbitrary toolchain support on Linux (b/177871873). This can be disabled via:

```
GOMA_ARBITRARY_TOOLCHAIN_SUPPORT=false goma_ctl restart
```

Copying and running chrome

A convenient way to copy chrome over to a Windows box is to build the `mini_installer` target. Then, copy just `mini_installer.exe` over to the Windows box and run it to install the chrome you just built.

Note that the `mini_installer` doesn't include PDB files. PDB files are needed to correctly symbolize stack traces (or if you want to attach a debugger).

Running tests on swarming

You can run the Windows binaries you built on swarming, like so:

```
tools/run-swarmed.py out/gnwin base_unittests [ --gtest_filter=... ]
```

See the contents of run-swarmed.py for how to do this manually.

The linux-win_cross-rel buildbot does 64-bit release cross builds, and also runs tests. You can look at it to get an idea of which tests pass in the cross build.