Google.

Hello, my name is Petr Hosek

phosek@google.com

#llvm/phosek

LLVM Developers' Meeting.

Compiling cross-toolchains with CMake and runtimes build

OCT. 2017

What is a cross-toolchain.

Clang is a cross-compiler, but that isn't sufficient to produce a working executable.

What we need is a cross-toolchain, which in addition to the cross-compiler also contains runtimes cross-compiled for the target platform.

Compiling a cross-toolchain

in two parts.

–

1. Cache files

to build toolchain components

2. Runtimes build

to cross-compile runtimes

Cache files.

Cache files are CMake scripts that can be used to populate the cache.

LLVM_DISTRIBUTION_COMPONENTS variable can be used to select specific components.

Link to source code

Fuchsia.cmake

```
set(LLVM_TARGETS_TO_BUILD X86;ARM;AArch64 CACHE STRING
    "")

set(CMAKE_BUILD_TYPE RelWithDebInfo CACHE STRING "")
set(CMAKE_C_FLAGS_RELWITHDEBINFO
    "-O3 -gline-tables-only -DNDEBUG" CACHE STRING "")
set(CMAKE_CXX_FLAGS_RELWITHDEBINFO
    "-O3 -gline-tables-only -DNDEBUG" CACHE STRING "")

set(LLVM_INSTALL_TOOLCHAIN_ONLY ON CACHE BOOL "")
set(LLVM_TOOLCHAIN_TOOLS
    llvm-ar
    llvm-cxxfilt
    llvm-nm
    llvm-objcopy
    llvm-objdump
    llvm-size
    ...
    CACHE STRING "")

set(LLVM_DISTRIBUTION_COMPONENTS
    clang
    lld
    LTO
    clang-format
    clang-headers
    ...
    ${LLVM_TOOLCHAIN_TOOLS}
    CACHE STRING "")
```

Cache files.

Cache files are CMake scripts that can be used to populate the cache.

LLVM_DISTRIBUTION_COMPONENTS variable can be used to select specific components.

Link to source code

Fuchsia.cmake

```
set(LLVM_TARGETS_TO_BUILD X86;ARM;AArch64 CACHE STRING
    "")

set(CMAKE_BUILD_TYPE RelWithDebInfo CACHE STRING "")
set(CMAKE_C_FLAGS_RELWITHDEBINFO
    "-O3 -gline-tables-only -DNDEBUG" CACHE STRING "")
set(CMAKE_CXX_FLAGS_RELWITHDEBINFO
    "-O3 -gline-tables-only -DNDEBUG" CACHE STRING "")

set(LLVM_INSTALL_TOOLCHAIN_ONLY ON CACHE BOOL "")
set(LLVM_TOOLCHAIN_TOOLS
    llvm-ar
    llvm-cxxfilt
    llvm-nm
    llvm-objcopy
    llvm-objdump
    llvm-size
    ...
    CACHE STRING "")

set(LLVM_DISTRIBUTION_COMPONENTS
    clang
    lld
    LTO
    clang-format
    clang-headers
    ...
    ${LLVM_TOOLCHAIN_TOOLS}
    CACHE STRING "")
```

Cache files.

Cache files are CMake scripts that can be used to populate the cache.

```
$ cmake -G Ninja \
    -C Fuchsia.cmake \
    -DFUCHSIA_x86_64_SYSROOT=<path> \
    -DFUCHSIA_aarch64_SYSROOT=<path> \
    ../llvm
```

Runtimes build.

Runtimes placed in the projects directory are built with the host toolchain for the default target.

```
llvm/
  projects/
    compiler-rt/
    libcxx/
    libcxxabi/
    libunwind/
    CMakeLists.txt
  runtimes/
```

Runtimes build.

Runtimes placed in the runtimes directory are built with the just-built compiler for selected targets.

```
llvm/
  projects/
  runtimes/
    compiler-rt/
    libcxx/
    libcxxabi/
    libunwind/
    CMakeLists.txt
```

Builtins.

Use the LLVM_BUILTIN_TARGETS to specify the compiler-rt builtin targets.

To pass a per target variable to the builtin build, you can set BUILTINS_<target>_<variable> where <variable> will be passed to the builtin build for <target>.

Link to source code

Fuchsia.cmake

```
set(LLVM_BUILTIN_TARGETS
    "x86_64-fuchsia;aarch64-fuchsia" CACHE STRING "")

foreach(target x86_64;aarch64)
  set(BUILTINS_${target}-fuchsia_CMAKE_SYSROOT
      "${FUCHSIA_${target}_SYSROOT}" CACHE PATH "")
  set(BUILTINS_${target}-fuchsia_CMAKE_SYSTEM_NAME
      Fuchsia CACHE STRING "")
endforeach()
```

Builtins.

Use the LLVM_BUILTIN_TARGETS to specify the compiler-rt builtin targets.

To pass a per target variable to the builtin build, you can set BUILTINS_<target>_<variable> where <variable> will be passed to the builtin build for <target>.

Link to source code

Fuchsia.cmake

```
set(LLVM_BUILTIN_TARGETS
    "x86_64-fuchsia;aarch64-fuchsia" CACHE STRING "")

foreach(target x86_64;aarch64)
  set(BUILTINS_${target}-fuchsia_CMAKE_SYSROOT
      "${FUCHSIA_${target}_SYSROOT}" CACHE PATH "")
  set(BUILTINS_${target}-fuchsia_CMAKE_SYSTEM_NAME
      Fuchsia CACHE STRING "")
endforeach()
```

Runtimes.

Use the LLVM_RUNTIME_TARGETS to specify the runtimes targets to be built.

To pass a per target variable to the runtimes build, you can set RUNTIMES_<target>_<variable> where <variable> will be passed to the runtimes build for <target>.

Link to source code

Fuchsia.cmake

```
set(LLVM_RUNTIME_TARGETS
    "x86_64-fuchsia;aarch64-fuchsia" CACHE STRING "")

foreach(target x86_64;aarch64)
  set(RUNTIMES_${target}-fuchsia_CMAKE_SYSROOT
      "${FUCHSIA_${target}_SYSROOT}" CACHE PATH "")
  set(RUNTIMES_${target}-fuchsia_CMAKE_SYSTEM_NAME
      Fuchsia CACHE STRING "")
  set(RUNTIMES_${target}-fuchsia_LLVM_ENABLE_LIBCXX
      ON CACHE BOOL "")
  set(RUNTIMES_${target}-fuchsia_LIBCXX_ABI_VERSION
      2 CACHE STRING "")
  ...
endforeach()
```

Build targets.

The build targets are available as builtins-<target> and runtimes-<target>. Use builtins and runtimes targets to build all targets.

Link to source code

Fuchsia.cmake

```
set(LLVM_DISTRIBUTION_COMPONENTS
    ...
    builtins
    runtimes
    ${LLVM_TOOLCHAIN_TOOLS}
    CACHE STRING "")
```

Distribution target.

Distribution is a target building only the components selected by the LLVM_DISTRIBUTION_COMPONENTS variable.

The check and install targets are accessible as check-<name>-<target> and install-<name>-<target> respectively.

```
$ cmake -G Ninja \
    -C Fuchsia.cmake \
    -DFUCHSIA_x86_64_SYSROOT=<path> \
    -DFUCHSIA_aarch64_SYSROOT=<path> \
    ../llvm
$ ninja distribution
$ ninja check-all
$ ninja install-distribution
```

Related Links.

Fuchsia.cmake and Fuchsia-stage2.cmake (source)

Developing and Shipping LLVM and Clang with CMake (video)