

Using LLVM's libc

Guillaume Chatelet
Michael Jones
Siva Chandra
Tue Ly

Agenda

1. Brief introduction to LLVM's libc and its current status
2. Building and installing the libc
3. Using the libc to link real applications
4. Quick guide to bringing up LLVM's libc for a new platform/architecture
5. Future plans and guide to participating in the libc development



Introduction

What is LLVM's libc?

- A greenfield libc developed with certain goals
 - Sanitizer friendly
 - Implemented in C/C++ source code without assembly
 - Tested
 - Unit tests, integration tests, exhaustive tests, all running on CI.
 - Modular
 - Configurable
 - Performance sensitive server side use-cases
 - Size-sensitive embedded use-cases
 - Use only the parts you need (e.g. omit i18n, floating point, etc.)

➤ Visit <https://libc.llvm.org/> for more information



Implementation Status

- About 400 functions (from the C standard and POSIX) are available [*]
 - Most of the single precision math functions ([link](#))
 - String functions not sensitive to locale ([link](#))
 - A thread library which supports both the pthreads and the C11 threads ([link](#))
 - A large part of stdio is available ([link](#))
 - Scudo standalone allocator can be packaged with the libc
 - See <https://llvm.org/docs/ScudoHardenedAllocator.html> for more information on Scudo
- Startup code to support fully static Linux/ELF executables

[*] - Some of the implementations are incomplete with respect to standards conformance.




CI Coverage - Compiler, Platform and Architecture

Currently Supported Platforms

- Linux - x86-64, aarch64, arm32
- Windows - x86-64
- MacOS - arm64
- Gradually integrating into Fuchsia libc

Continuous Integration

- Linux - x86-64, aarch64 and arm32
 - Windows - x86-64
 - Compilers - clang
- 

Building the libc

The libc can be built in two different modes:

- **Overlay mode**
 - Use system headers and system libc for missing functions.
- **Fullbuild mode**
 - Use LLVM libc's headers and only functions provided by LLVM's libc.



The Overlay Mode

- In the overlay mode, LLVM's libc cannot be used by itself
 - Exploit link order semantics to use whatever is available in LLVM's libc and get the rest from the system libc
 - User programs use headers from the system libc
 - Startup objects and the libraries like libc.a not provided by LLVM's libc are also picked from the system libc
 - Only pieces which are not dependent on the implementation specific ABI are included
 - Functions like `strlen`, `round` are included
 - Functions like `fopen` are not included - they depend on the implementation specific definition of the `FILE` data structure



Building the Overlay Mode libc

- Least complicated and straightforward way to use LLVM's libc
- Build using the standard LLVM conventions:
 - Build libc by itself

```
$> cmake ../llvm -G Ninja -DLLVM_ENABLE_PROJECTS="libc" \
  -DCMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++ \
  -DCMAKE_BUILD_TYPE=<Debug|Release> \
  -DCMAKE_INSTALL_PREFIX=<Your prefix of choice>
$> ninja llvmlibc
$> ninja install-llvmlibc
```

- Build libc as part of the bootstrap build

```
$> cmake ../llvm -G Ninja -DLLVM_ENABLE_PROJECTS="clang" \
  -DCMAKE_ENABLE_RUNTIME="libc" \
  -DCMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++ \
  -DCMAKE_BUILD_TYPE=<Debug|Release> \
  -DCMAKE_INSTALL_PREFIX=<Your prefix of choice>
$> ninja llvmlibc
$> ninja install-llvmlibc
```

➤ See https://libc.llvm.org/overlay_mode.html

Building the Overlay Mode libc

- Least complicated and straightforward way to use LLVM's libc
- Build using the standard LLVM conventions:
 - Build libc by itself

```
$> cmake ../llvm -G Ninja -DLLVM_ENABLE_PROJECTS="libc" \
  -DCMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++ \
  -DCMAKE_BUILD_TYPE=<Debug|Release> \
  -DCMAKE_INSTALL_PREFIX=<Your prefix of choice>
$> ninja llvmlibc
$> ninja install-llvmlibc
```

- Build libc as part of the bootstrap build

```
$> cmake ../llvm -G Ninja -DLLVM_ENABLE_PROJECTS="clang" \
  -DCMAKE_ENABLE_RUNTIME="libc" \
  -DCMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++ \
  -DCMAKE_BUILD_TYPE=<Debug|Release> \
  -DCMAKE_INSTALL_PREFIX=<Your prefix of choice>
$> ninja llvmlibc
$> ninja install-llvmlibc
```

➤ See https://libc.llvm.org/overlay_mode.html

Building the Overlay Mode libc

- Least complicated and straightforward way to use LLVM's libc
- Build using the standard LLVM conventions:
 - Build libc by itself

```
$> cmake ../llvm -G Ninja -DLLVM_ENABLE_PROJECTS="libc" \
  -DCMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++ \
  -DCMAKE_BUILD_TYPE=<Debug|Release> \
  -DCMAKE_INSTALL_PREFIX=<Your prefix of choice>
$> ninja llvmlibc
$> ninja install-llvmlibc
```

- Build libc as part of the bootstrap build

```
$> cmake ../llvm -G Ninja -DLLVM_ENABLE_PROJECTS="clang" \
  -DCMAKE_ENABLE_RUNTIME="libc" \
  -DCMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++ \
  -DCMAKE_BUILD_TYPE=<Debug|Release> \
  -DCMAKE_INSTALL_PREFIX=<Your prefix of choice>
$> ninja llvmlibc
$> ninja install-llvmlibc
```

➤ See https://libc.llvm.org/overlay_mode.html

The Overlay Mode in Action

- Use the link order to overlay symbols from `libllvmlibc.a`

```
$> clang <...> file.<c|cpp> -L <path to libllvmlibc.a> -llvmlibc
```

- Try it:

- Add `llvmlibc` as a `target_link_library` to `llvm-objcopy`
 - The number of symbols it pulls from `glibc` drops from 112 to 58
 - Run `ninja check-llvm` to make sure that linking `libllvmlibc.a` did not cause any regressions
- If you are bold enough, add `llvmlibc` as a `target_link_library` to all LLVM tools
 - *NOTE: Running `ninja check-llvm` shows some regressions*



The Full Build Mode

- In the full mode LLVM's libc is used as the only libc
 - User programs use headers from LLVM's libc
 - The main libc.a static archive and the startup objects like crt1.o come from LLVM's libc

NOTE: Currently the full build mode only supports fully statically linked binaries (no dynamic loader etc.)



Building Full Build Mode

- Building the full libc is straightforward
- Installation is more involved than installing the overlay static archive
- Install a sysroot with an LLVM only toolchain
 - Install clang, lld and compiler-rt along with the libc
- Cannot use a C++ standard library or build C++ programs yet
 - The libc is not complete enough to satisfy libc++ requirements



Building and Install the full toolchain

- Standard LLVM CMake conventions

```
$> cmake ../llvm -G Ninja -DLLVM_ENABLE_PROJECTS="libc;lld;compiler-rt;clang" \
-DMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++ \
-DMAKE_BUILD_TYPE=<Debug|Release> \
-DMAKE_INSTALL_PREFIX=<Your prefix of choice>
-DLLVM_LIBC_FULL_BUILD=ON \ # We want the full libc
-DLLVM_LIBC_INCLUDE_SCUDO=ON \ # Include Scudo in the libc
-DCOMPILER_RT_BUILD_SCUDO_STANDALONE_WITH_LLVM_LIBC=ON \ # Build Scudo against libc headers
-DCOMPILER_RT_BUILD_GWP_ASAN=OFF \ # Do not include GWP-ASAN with Scudo
-DCOMPILER_RT_SCUDO_STANDALONE_BUILD_SHARED=OFF # Do not build the Scudo shared object
```

- Install:

```
$> ninja install-clang install-builtins install-compiler-rt \
install-core-resource-headers install-libc install-llvm
```

- Linux Headers:

- libc headers include few safe linux headers
- Install linux headers into the sysroot

Using the libc - Building and Install the full toolchain

- Standard LLVM CMake conventions

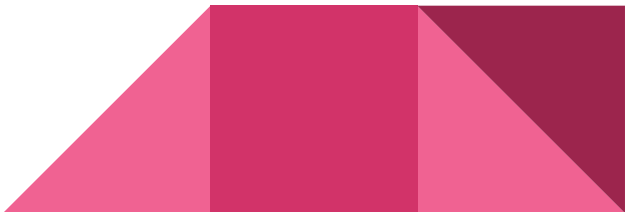
```
$> cmake ../llvm -G Ninja -DLLVM_ENABLE_PROJECTS="libc;lld;compiler-rt;clang" \
-DMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++ \
-DMAKE_BUILD_TYPE=<Debug|Release> \
-DMAKE_INSTALL_PREFIX=<Your prefix of choice> \
-DLLVM_LIBC_FULL_BUILD=ON \ # We want the full libc
-DLLVM_LIBC_INCLUDE_SCUDO=ON \ # Include Scudo in the libc
-DCOMPILER_RT_BUILD_SCUDO_STANDALONE_WITH_LLVM_LIBC=ON \ # Build Scudo against libc headers
-DCOMPILER_RT_BUILD_GWP_ASAN=OFF \ # Do not include GWP-ASAN with Scudo
-DCOMPILER_RT_SCUDO_STANDALONE_BUILD_SHARED=OFF \ # Do not build the Scudo shared object
```

- Install:

```
$> ninja install-clang install-builtins install-compiler-rt \
install-core-resource-headers install-libc install-llvm
```

- Linux Headers:

- libc headers include few safe linux headers
- Install linux headers into the sysroot



Using the libc - Building and Install the full toolchain

- Standard LLVM CMake conventions

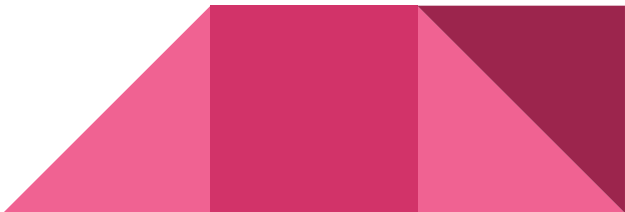
```
$> cmake ../llvm -G Ninja -DLLVM_ENABLE_PROJECTS="libc;lld;compiler-rt;clang" \
-DMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++ \
-DMAKE_BUILD_TYPE=<Debug|Release> \
-DMAKE_INSTALL_PREFIX=<Your prefix of choice> \
-DLLVM_LIBC_FULL_BUILD=ON \ # We want the full libc
-DLLVM_LIBC_INCLUDE_SCUDO=ON \ # Include Scudo in the libc
-DCOMPILER_RT_BUILD_SCUDO_STANDALONE_WITH_LLVM_LIBC=ON \ # Build Scudo against libc headers
-DCOMPILER_RT_BUILD_GWP_ASAN=OFF \ # Do not include GWP-ASAN with Scudo
-DCOMPILER_RT_SCUDO_STANDALONE_BUILD_SHARED=OFF \ # Do not build the Scudo shared object
```

- Install:

```
$> ninja install-clang install-builtins install-compiler-rt \
install-core-resource-headers install-libc install-llvm
```

- Linux Headers:

- libc headers include few safe linux headers
- Install linux headers into the sysroot



Using the libc - Building and Install the full toolchain

- Standard LLVM CMake conventions

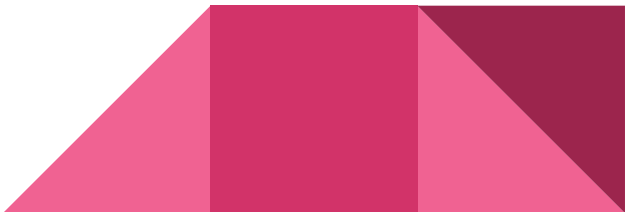
```
$> cmake ../llvm -G Ninja -DLLVM_ENABLE_PROJECTS="libc;lld;compiler-rt;clang" \
-DMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++ \
-DMAKE_BUILD_TYPE=<Debug|Release> \
-DMAKE_INSTALL_PREFIX=<Your prefix of choice> \
-DLLVM_LIBC_FULL_BUILD=ON \ # We want the full libc \
-DLLVM_LIBC_INCLUDE_SCUDO=ON \ # Include Scudo in the libc \
-DCOMPILER_RT_BUILD_SCUDO_STANDALONE_WITH_LLVM_LIBC=ON \ # Build Scudo against libc headers \
-DCOMPILER_RT_BUILD_GWP_ASAN=OFF \ # Do not include GWP-ASAN with Scudo \
-DCOMPILER_RT_SCUDO_STANDALONE_BUILD_SHARED=OFF \ # Do not build the Scudo shared object
```

- Install:

```
$> ninja install-clang install-builtins install-compiler-rt \
install-core-resource-headers install-libc install-lld
```

- Linux Headers:

- libc headers include few safe linux headers
- Install linux headers into the sysroot



Using the libc - Building and Install the full toolchain

- Standard LLVM CMake conventions

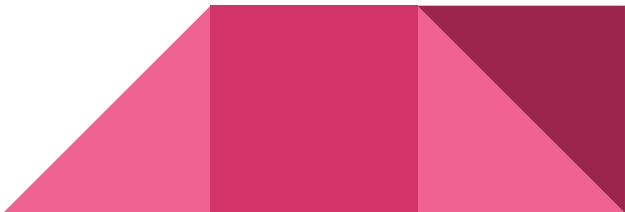
```
$> cmake ../llvm -G Ninja -DLLVM_ENABLE_PROJECTS="libc;lld;compiler-rt;clang" \
-DMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++ \
-DMAKE_BUILD_TYPE=<Debug|Release> \
-DMAKE_INSTALL_PREFIX=<Your prefix of choice> \
-DLLVM_LIBC_FULL_BUILD=ON \ # We want the full libc \
-DLLVM_LIBC_INCLUDE_SCUDO=ON \ # Include Scudo in the libc \
-DCOMPILER_RT_BUILD_SCUDO_STANDALONE_WITH_LLVM_LIBC=ON \ # Build Scudo against libc headers \
-DCOMPILER_RT_BUILD_GWP_ASAN=OFF \ # Do not include GWP-ASAN with Scudo \
-DCOMPILER_RT_SCUDO_STANDALONE_BUILD_SHARED=OFF # Do not build the Scudo shared object
```

- Install:

```
$> ninja install-clang install-builtins install-compiler-rt \
install-core-resource-headers install-libc install-llvm
```

- Linux Headers:

- libc headers include few safe linux headers
- Install linux headers into the sysroot



Using the libc - Building and Install the full toolchain

- Standard LLVM CMake conventions

```
$> cmake ../llvm -G Ninja -DLLVM_ENABLE_PROJECTS="libc;lld;compiler-rt;clang" \
-DMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++ \
-DMAKE_BUILD_TYPE=<Debug|Release> \
-DMAKE_INSTALL_PREFIX=<Your prefix of choice> \
-DLLVM_LIBC_FULL_BUILD=ON \ # We want the full libc \
-DLLVM_LIBC_INCLUDE_SCUDO=ON \ # Include Scudo in the libc \
-DCOMPILER_RT_BUILD_SCUDO_STANDALONE_WITH_LLVM_LIBC=ON \ # Build Scudo against libc headers \
-DCOMPILER_RT_BUILD_GWP_ASAN=OFF \ # Do not include GWP-ASAN with Scudo \
-DCOMPILER_RT_SCUDO_STANDALONE_BUILD_SHARED=OFF \ # Do not build the Scudo shared object
```

- Install:

```
$> ninja install-clang install-builtins install-compiler-rt \
install-core-resource-headers install-libc install-llvm
```

- Linux Headers:

- libc headers include few safe linux headers
- Install linux headers into the sysroot



Using the libc - Full Build Mode in Action

- Try the examples available in the libc directory
<https://github.com/llvm/llvm-project/tree/main/libc/examples>
- Compile options:
 - `-sysroot=<>`
- Linker options:
 - `-sysroot=<>`
 - `-rtlib=compiler-rt`
 - `-fuse-ld=lld`

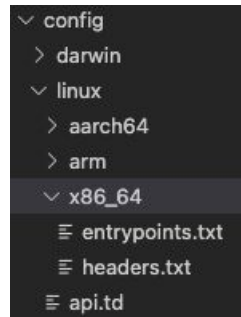
➤ **See the CMake logic for the examples:**

[<...>/libc/examples/examples.cmake](https://github.com/llvm/llvm-project/tree/main/libc/examples/examples.cmake)



Bringing up LLVM's libc on a new Platform/Architecture

- Platform and architecture configs are specified in the `libc/config` directory
- Add a new platform by creating
 - `<Target OS>/<Target Architecture>/entrypoints.txt`
- Architecture and platform independent entrypoints such as `strcpy`, `strlen` can be brought in straightforward manner
- Some functions will need specialization for new platforms
 - E.G. `src/___support/OSUtil/` has subdirectories for different target architectures
- For fullbuild you will need a few more things:
 - `<Target OS>/api.td`
 - `<Target OS>/<Target Architecture>/headers.txt`




➤ See <https://libc.llvm.org/porting.html> for more information

Near-Term Focus Areas

- Math library
 - Implement double and higher precision flavors of the transcendental math function
- Stdio and Pthread
 - Improve coverage
 - Not all functions are available
 - Improve configurability
 - Add options for shrinking code size for embedded use cases
 - Improve standards conformance
 - A few corners of POSIX are not fully implemented



Near-Term Focus Areas (2)

- Startup Subsystem
 - Add support for static-PIE (position independent executable) linking
 - Platform and Architecture Coverage
 - Continue integration in to Fuchsia's libc
 - Maybe bring up for RISC-V?
 - Improve arm32 coverage
 - CI for darwin (both -intel and -arm64)?
 - Miscellaneous
 - Move away from table-gen
 - Mechanical code style clean up
- 

Contributing

- Want to contribute a port for a new target or platform?
 - Coding aspect: See: <https://libc.llvm.org/porting.html> for setting up the various configs for the new port.
 - Engineering aspect: Along with the code, we also want to see a plan for standing up CI builders



Contributing (2)

- Want to help with other open areas?
 - Cleaning up coding style
 - Adding CMake options to link overlay libc with other LLVM binaries
 - Put plumbing in place to start shipping the overlay libc binaries with LLVM binary releases
 - Implement Linux syscall wrappers
 - Better random number generator
 - Double and higher precision math functions
 - Tue Ly will be talking about our math functions in depth tomorrow

➤ See: <https://libc.llvm.org/contributing.html>

Communication

- Discord Channel:

<https://discord.com/channels/636084430946959380/636732994891284500>

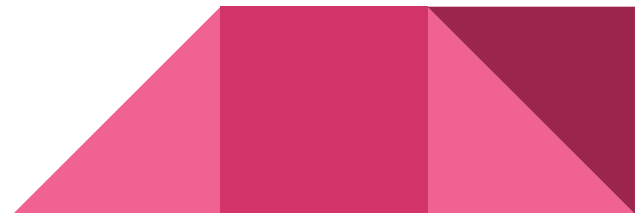
- Discourse:

<https://discourse.llvm.org/c/runtimes/libc>

- Bug reports:

<https://github.com/llvm/llvm-project/labels/libc>

- **The above links are available under “External Links” on libc.llvm.org**



Thank You

