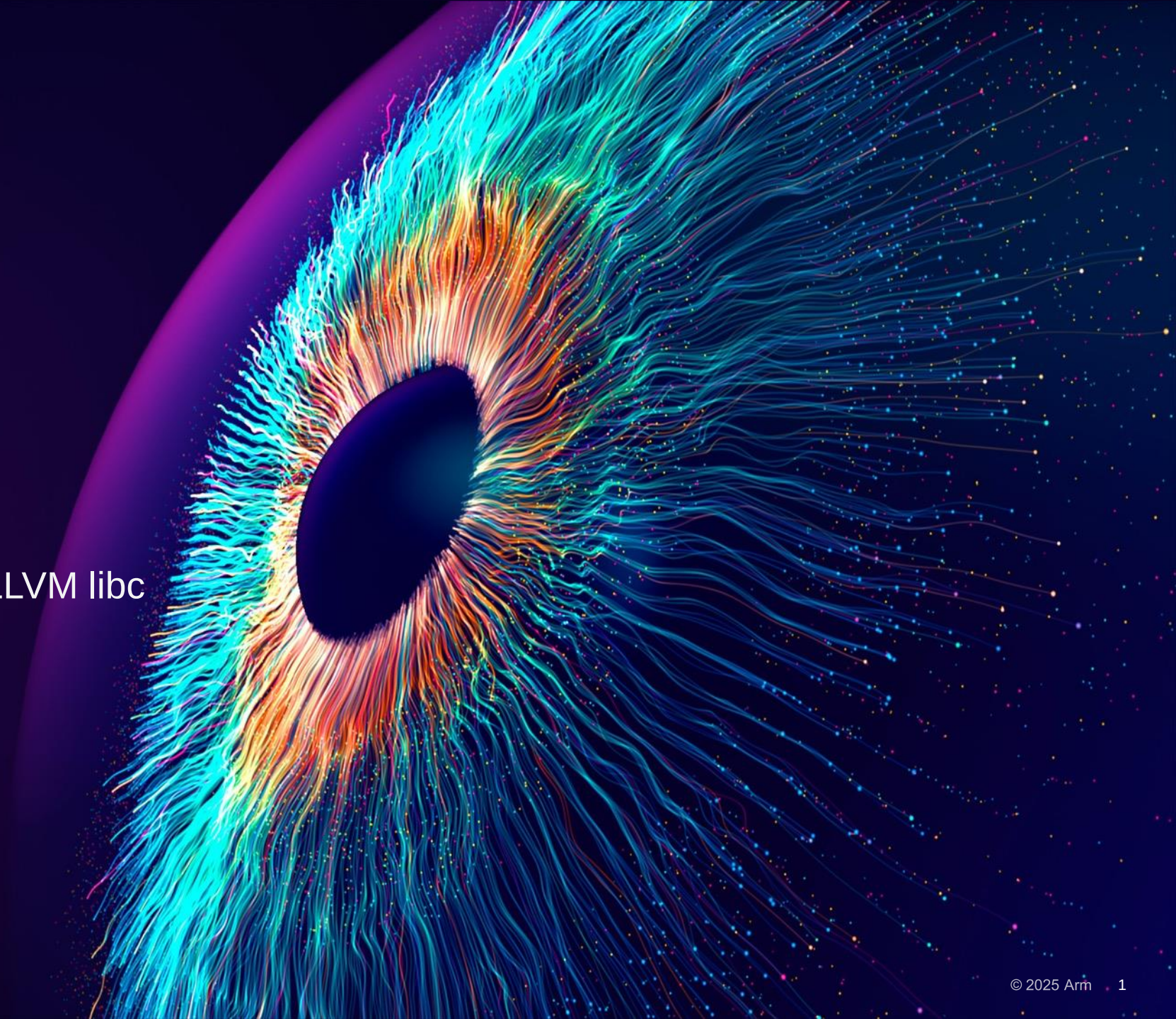
arm

LLVM libc

Introduction to building and using LLVM libc



Peter Smith

2025-01-24

Agenda

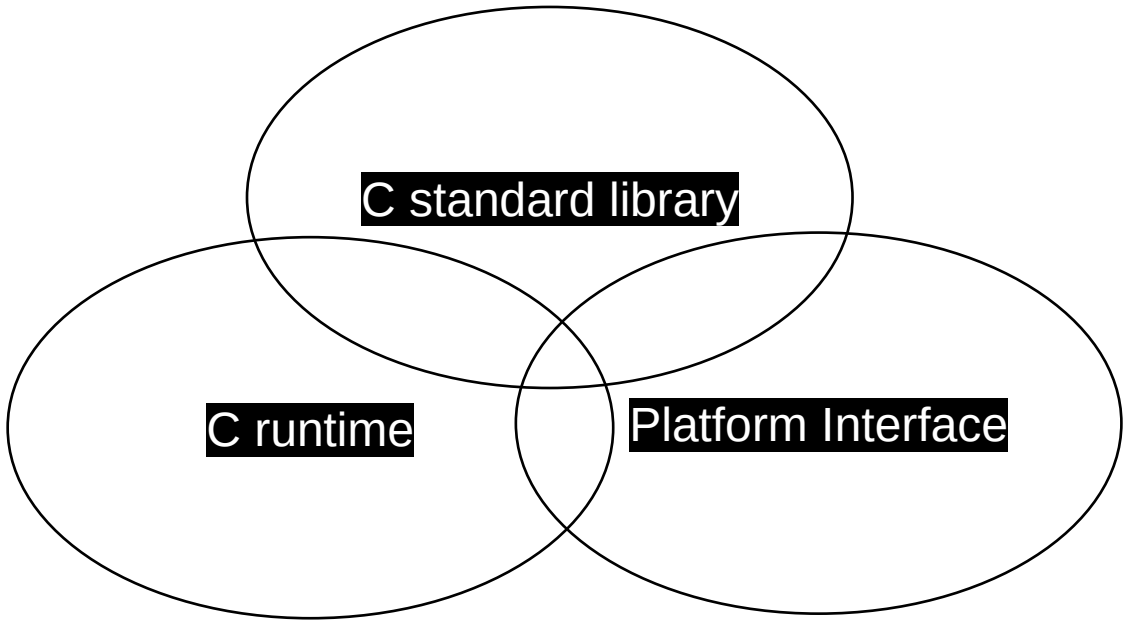
- What is LLVM libc and what is the motivation behind it?
- LLVM libc status.
- Building and using LLVM libc.
- LLVM libc in embedded systems.
- How to get involved?

arm

What is LLVM libc and why?

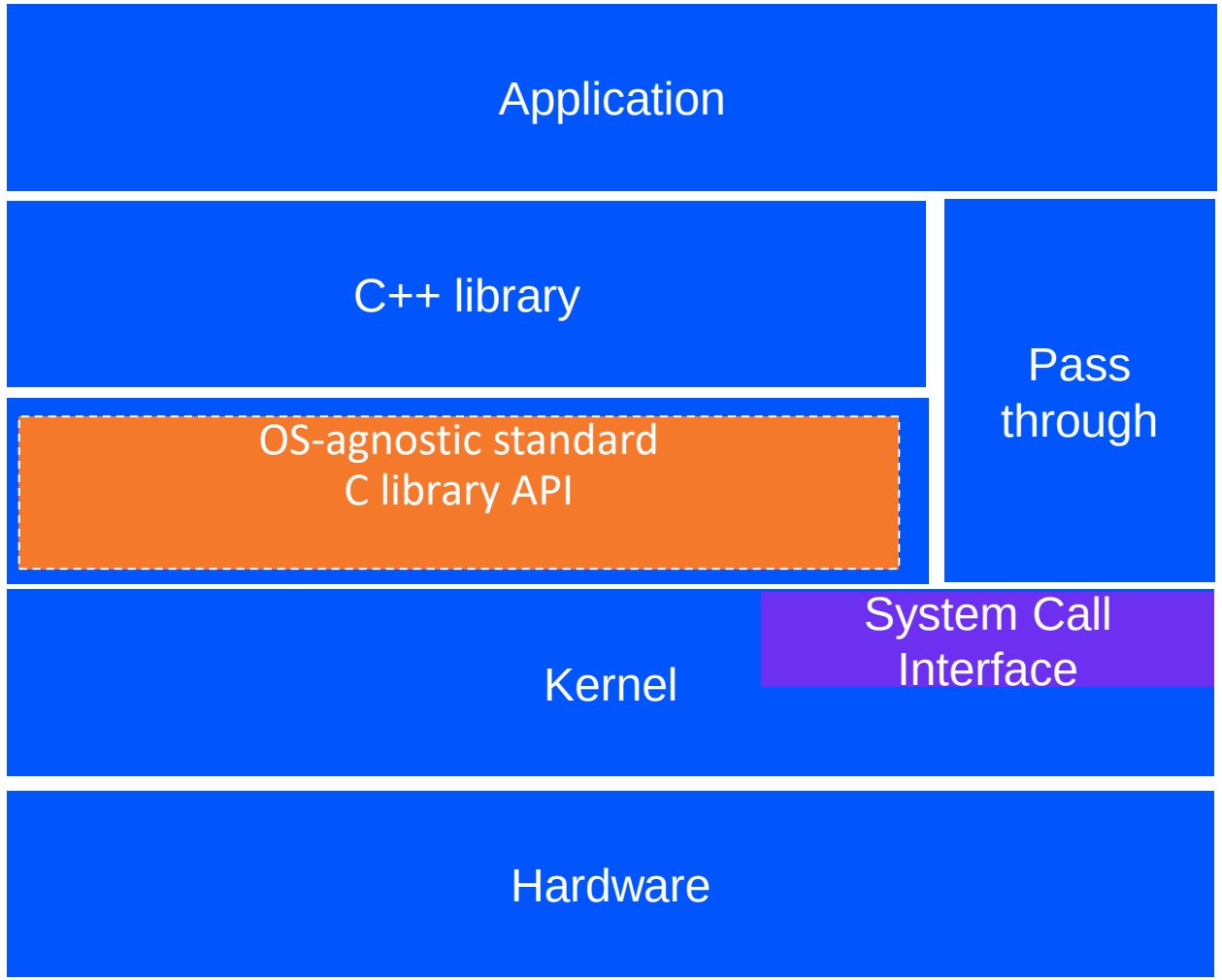
Peter Smith
2025-01-24

A libc is more than just the C standard library



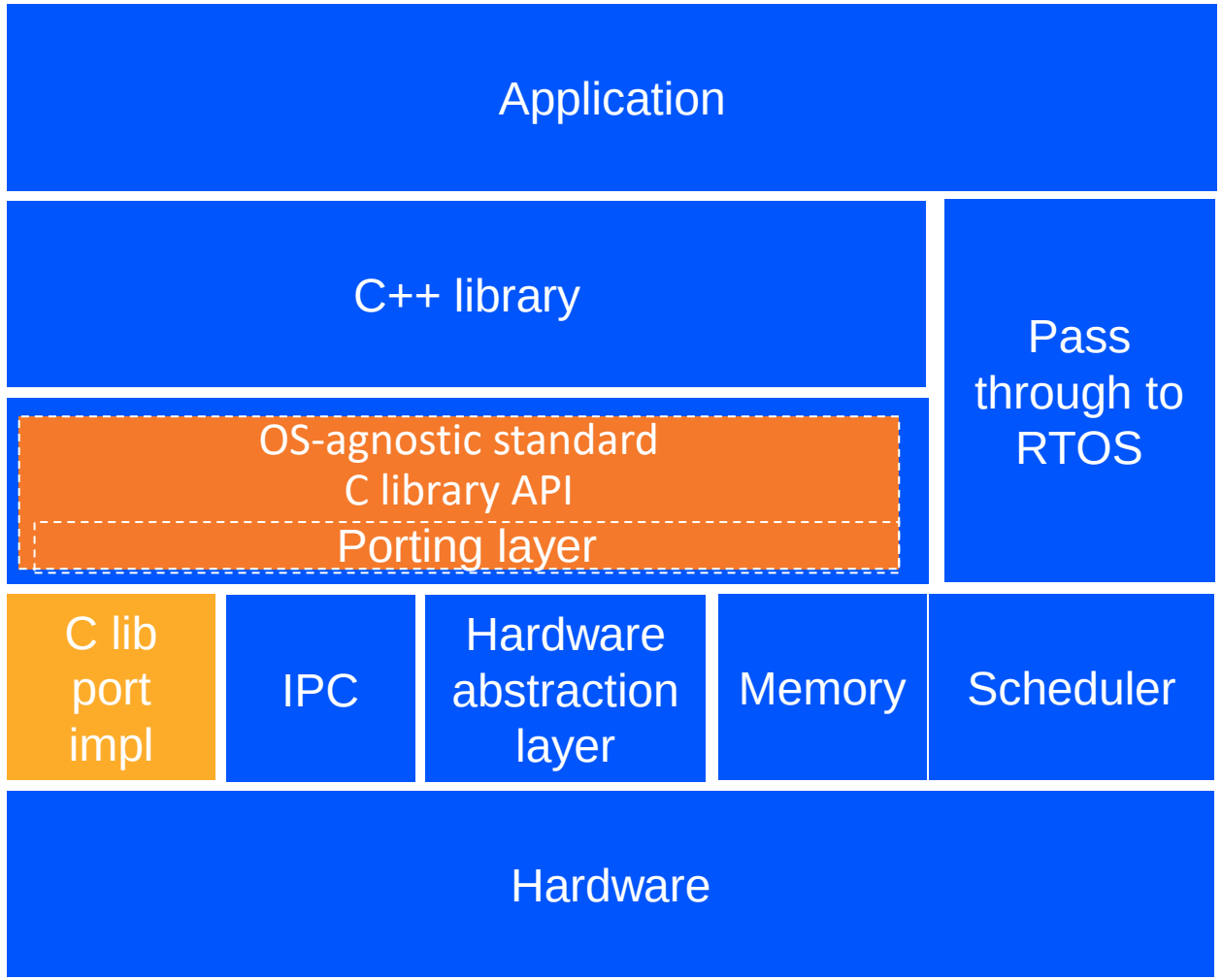
- C standard library
  - A subset of the functions defined by the C-standard.
- Platform interface
  - C standard functions that use platform features.
  - Standardized like C POSIX library.
  - Platform specific like Linux syscalls.
  - Dynamic linker.
- C-runtime
  - Functions used by the compiler to implement C.
- Boundaries between components are platform dependent.
- Parts are inescapably platform dependent.

Hosted library on an OS like Linux



- Standard C++ library
  - OS-agnostic at the top (application) end
  - BUT targeted at specific C library at the bottom
- OS-agnostic standard C library API
  - Implements the C standard library spec.
  - Can use kernel functionality.
- OS-specific functionality
  - C POSIX library.
  - Syscall wrappers.

Freestanding “Bare metal” with C++ and C library with RTOS



- Standard C++ library
  - OS-agnostic at the top (application) end
  - BUT targeted at specific C library at the bottom
- OS-agnostic standard C library API
  - Implements the standard C library spec.
  - Porting layer such as newlib’s libgloss. Consists of functions an RTOS must implement.
- RTOS functionality
  - Any POSIX extensions supported by OS is implemented by the OS.
  - All one executable so these are just function calls.

Motivations behind LLVM libc

- A configurable implementation that can be customized for the use case
  - Omit parts of the C-library that aren't used.
  - Chance to choose between different implementation strategies.
  - Permit use in targets such as the GPU.
- Creation of fully static binaries without license implications.
- Reproducible math library results across platform
  - Use correct rounding while retaining competitive performance.
- Opportunities from Implementation inside llvm-project
  - Can share code with libc++.
  - Can use and work well with llvm testing infrastructure.
  - Implementation can be done in C++.
  - Can build a complete toolchain from just llvm-project!

Challenges in creating LLVM libc

- Libc is more platform dependent than compiler dependent
  - On some platforms the libc is integral to the system.
  - Designing an abstraction layer to suit all platforms is difficult.
- Every libc must make trade-offs between performance, size and functionality
  - A microcontroller and a server have different requirements.
- Existing platforms have one or more libc implementations already
  - glibc and musl on Linux; bionic on Android.
- Several open-source bare-metal C-libraries are available
  - newlib and picolibc.

arm

LLVM libc status

Peter Smith

2025-01-24

What does LLVM libc support now?

- Can be built in two configurations
  - As an **overlay** build, that supplements an existing platform libc
  - A **full** build of libc that is the only libc on the platform. Can be host or cross-compiled.
- Tested architectures
  - AArch64
  - AMDGPU
  - Arm
  - NVPTX
  - RISCV32
  - RISCV64
  - X86_64
- Platforms
  - Baremetal (embedded)
  - GPU (amgcn-amd-amdhsa and nvptx64-nvidia-cuda targets)
  - Linux
  - Fuchsia and Android (via overlay)
  - Darwin (partial)
  - Windows (partial)

Why might you want to use LLVM libc now?

- Require static linking without any license implications
- On a host with an existing libc
  - Correctly rounded Math library with consistent results across platforms.
  - Fixed point library support unique to LLVM libc.
- Running C/C++ programs on a GPU
  - llvm libc calls back to host via RPC, in a similar way to semihosting for features not supported on a GPU.
- On Embedded systems
  - Fixed point library support unique to LLVM libc.
- It’s fun to try new things!

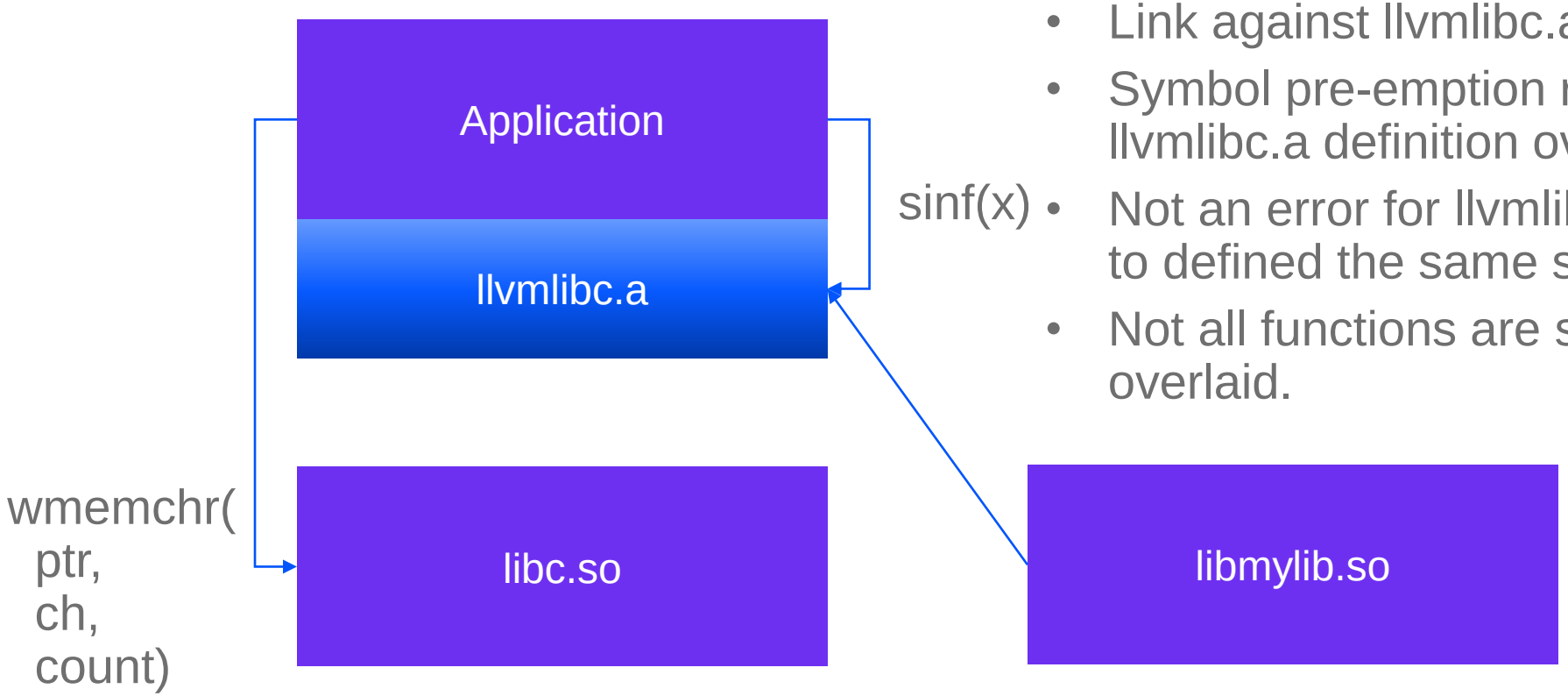
arm

Building and Using LLVM libc

Peter Smith

2025-01-24

LLVM libc in overlay mode



- Requires dynamic linking
- Link against llvmlibc.a and libc.so.
- Symbol pre-emption rules will prefer llvmlibc.a definition over libc.so.
- Not an error for llvmlibc.a and libc.so to defined the same symbol.
- Not all functions are suitable to be overlaid.

Building and using LLVM libc in overlay mode

- See llvm-project/libc/docs/overlay_mode.rst
- Add `-DLLVM_ENABLE_RUNTIMES="libc"` in a full bootstrap or runtimes build

```
CC=clang CXX=clang++ cmake ../runtimes –G Ninja \
  -DLLVM_ENABLE_RUNTIMES="libc" \
  -DCMAKE_INSTALL_PREFIX=<prefix>

ninja libc          # build libllvmlibc.a
ninja check-libc    # unit tests only
ninja install-libc # <prefix>/lib
```

- To use, add `-L <prefix>/lib -lllvmlibc` to your project's linker flags.
- Use the platform's C header files.

LLVM libc in standalone mode on Linux

- LLVM libc is the only C-library.
- Includes startup code in crt*
- Static linking only, no dynamic-loader implementation.
  - Dynamic linker planned.
- Can only use what has been implemented, no fall back to the platform libc.
- Used by the GPU build. See llvm-project/libc/docs/gpu/building.rst

Building LLVM libc in standalone mode on Linux

- See llvm-project/libc/docs/full_host_build.rst
- May require symlink from `/usr/include/asm` to `/usr/include/<TARGET TRIPLE/asm`
  - Example triple x86_64-linux-gnu.
- SCUDO from compiler-rt needed for malloc

```
SYSROOT=/path/to/sysroot
CC=clang CXX=clang++ cmake ../runtimes -G Ninja \
  -DLLVM_ENABLE_RUNTIMES="libc;compiler-rt" \
  -DLLVM_LIBC_FULL_BUILD=ON \
  -DLLVM_LIBC_INCLUDE_SCUDO=ON \
  -DLLVM_COMPILER_RT_BUILD_SCUDO_STANDALONE_WITH_LLVM_LIBC \
  -DLLVM_COMPILER_RT_BUILD_GWP_ASAN=OFF \
  -DLLVM_COMPILER_RT_SCUDO_STANDALONE_BUILD_SHARED=OFF \
  -DCMAKE_INSTALL_PREFIX=${SYSROOT}

ninja libc libm check-libc # build and test libc.a and libm.a
ninja install-libc # ${SYSROOT}
```

Using the full host build

- Recommended to build a full toolchain into a sysroot
  - Contains bin, include and lib directories
  - Clang as the compiler
  - Default to compiler-rt and ld.lld.
- Can’t include libc++ yet, but hopefully soon.
- Linux headers need to be installed into the include/sys directory
- Details in llvm-project/libc/docs/full_host_build.rst

```
cmake -G Ninja ../llvm \
  -DLLVM_PROJECTS_TO_BUILD="clang;lld" \
  -DLLVM_RUNTIMES_TO_BUILD="compiler-rt;libc" \
  ...
/path/to/sysroot/bin/clang -static main.c
```

arm

LLVM libc in embedded systems

Peter Smith

2025-01-24

LLVM libc baremetal for embedded systems

- A baremetal build is a form of full build, without the POSIX support.
- Instructions in llvm-project/libc/full_cross_build.rst
  - See Bootstrap cross build.
- Uses the baremetal config
  - LLVM libc provides sample configs for Arm, AArch64 and RISCV.
  - Can provide your own config to customize the output.
- Configs define the headers to include and entrypoints (functions).
- Configs can alter the configuration of functions
  - errno accessed via a user-defined function (LIBC_ERRNO_MODE_EXTERNAL)
  - printf is limited in functionality to limit code-size.
  - qsort uses heap sort
  - math library tuned for a small device.
- Does not come with startup code, see appendix at end of presentation for example
  - stdio initialization
  - Heap location and bounds
  - errno
  - exit

Functionality

Embedded (red shows missing headers)

Headers

- assert.h
- ctype.h // ASCII only
- errno.h
- fenv.h
- float.h
- inttypes.h
- limits.h
- locale.h
- math.h
- setjmp.h
- signal.h
- stdfix.h
- stdint.h
- stdio.h // No FILE*, float printf
- stdlib.h
- string.h
- strings.h
- time.h
- threads.h
- uchar.h
- wchar.h
- wctype.h

Prebuilt and integrated toolchains

- Rasberry Pi Pico pigweed sdk
  - Arm and RISCV support for the CPUs on the Pico.
  - Follow the Sense tutorial https://pigweed.dev/docs/showcases/sense/tutorial/
- Arm Toolchain for Embedded
  - A LLVM based equivalent to the GNU Arm Embedded Toolchain.
  - Uses picolibc as the C-library.
- LLVM-libc can be built as an add on package
  - Supports same set of architectures as picolibc.
    - AArch64.
    - Arm A, R and M-profile.
    - Includes a sample with startup code.
- C-only at the moment
  - Work ongoing to build libc++ against llvm-libc upstream.
  - Possible to do today by disabling some libc++ functionality and substituting some functions.

arm

Conclusion and reference

Peter Smith
2025-01-24

Conclusion

- LLVM libc is new libc implementation within the LLVM project.
- Complete enough for many use cases.
  - Many ermbedded projects can be built with what is available.
  - Overlay mode to supplement the platform libc.
- The best or only choice for some use cases
  - Fixed point math library functions.
  - GPU targets.
- LLVM libc complete enough to be used able for many use cases.
- Available in prebuilt form for embedded toolchains
  - Arm Toolchain for Embedded.
  - Pigweed SDK for Raspberry Pi Pico.
- GPU port a demonstration of the configurability of LLVM libc.

How to get involved?

- Use it and give feedback
  - libc tag on llvm-project GitHub for issues.
  - libc channel on LLVM Discord
  - libc tag on Discourse https://discourse.llvm.org/tag/libc
- LLVM libc still at early stages of development
  - Scope to make ABI breaking changes.
- Open to new contributors.
  - Ports to new and existing platforms.
  - Implementing missing functions.
  - Outstanding work recorded in llvm-project issues using libc tag.
  - Bootstrapping clang with LLVM libc an early goal.
- Contribution guide
  - https://github.com/llvm/llvm-project/blob/main/libc/docs/contributing.rst
- Monthly Community Meetup
  - Every 4 weeks on Thursday at 18:00 (CET). See LLVM Community Calendar for details.
  - https://discourse.llvm.org/t/monthly-llvm-libc-meeting/74259
  - Often mentioned in the LLVM Embedded Toolchains meetup.

References

- Home page
  - https://libc.llvm.org/
- Tutorial on building and using LLVM (from 2022)
  - https://llvm.org/devmtg/2022-11/slides/Tutorial1-UsingLLVM-libc.pdf
- Introduction thread on llvm-dev
  - https://groups.google.com/g/llvm-dev/c/TnzMbasBBw8
- Collected list of presentations on LLVM libc
  - https://github.com/llvm/llvm-project/blob/main/libc/docs/talks.rst
- Introducing Pigweed SDK
  - https://opensource.googleblog.com/2024/08/introducing-pigweed-sdk.html
- Arm Toolchain for Embedded
  - https://github.com/arm/arm-toolchain
- A Case for Correctly Rounded Math Libraries
  - https://www.youtube.com/watch?v=vAcf6d26kiM
- LLVM Community Meetups
  - https://calendar.google.com/calendar/u/0/embed?src=calendar@llvm.org

arm

Merci
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Thank You
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה
ధన్యవాదములు
Köszönöm

arm

Backup

LLVM minimal startup code for Arm v6-M

Peter Smith
2025-01-24

Startup code

- Adapted from baremetal sample application for Armv6-m
  - https://github.com/arm/arm-toolchain/tree/arm-software/arm-software/embedded/llvmlibc-samples/src/llvmlibc/baremetal-semihosting
- Vector table and _start are similar to any baremetal startup code.
- Stdio, exit, errno and heap are specific to LLVM libc.

Startup code for a v6-m semihosted system using LLVM libc

```
extern int main(int argc, char** argv);

extern void _platform_init();
// From Linker script
extern char __data_source[];
extern char __data_start[];
extern char __data_size[];
extern char __bss_start[];
extern char __bss_size[];

// Called on reset. Copy RW data and ZI .bss
// _platform_init() sets up stdio.
// Call main via _Exit wrapper
void _start(void) {
  memcpy(__data_start, __data_source,
         (size_t) __data_size);
  memset(__bss_start, 0, (size_t) __bss_size);
  _platform_init();
  _Exit(main(0, NULL));
}
```

```
extern uint8_t __stack[];
extern void _start(void);
typedef void(*VECTOR_TABLE_Type)(void);
const VECTOR_TABLE_Type __VECTOR_TABLE[496]
__attribute__((section(".vectors"))) __attribute__((aligned(128))) = {
  (VECTOR_TABLE_Type)__stack,               /*     Initial Stack Pointer */
  _start,                                   /*     Reset Handler */
  NMI_Handler,                              /*     NMI Handler */
  HardFault_Handler,                        /*     Hard Fault Handler */
  MemManage_Handler,                        /*     MPU Fault Handler */
  BusFault_Handler,                         /*     Bus Fault Handler */
  UsageFault_Handler,                       /*     Usage Fault Handler */
  0,                                        /*     Reserved */
  0,                                        /*     Reserved */
  0,                                        /*     Reserved */
  0,                                        /*     Reserved */
  SVC_Handler,                              /*     SVC Handler */
  DebugMon_Handler,                         /*     Debug Monitor Handler */
  0,                                        /*     Reserved */
  PendSV_Handler,                           /*     PendSV Handler */
  SysTick_Handler,                          /*     SysTick Handler */
  /* Unused */
};
```

Additional functions for LLVM libc stdio support

```
struct __llvm_libc_stdio_cookie { int handle; };
struct __llvm_libc_stdio_cookie __llvm_libc_stdin_cookie;
struct __llvm_libc_stdio_cookie __llvm_libc_stdout_cookie;
struct __llvm_libc_stdio_cookie __llvm_libc_stderr_cookie;
// stdio_read and stdio_write are similar, just forward to the appropriate semihosting id.
static void stdio_open(struct __llvm_libc_stdio_cookie *cookie, int mode) {
  size_t args[3];
  args[0] = (size_t) ":tt";
  args[1] = (size_t)mode;
  args[2] = (size_t)3; /* name length */
  cookie->handle = semihosting_call(SYS_OPEN, args);
}

void _platform_init(void) {
  stdio_open(&__llvm_libc_stdin_cookie, OPENMODE_R);
  stdio_open(&__llvm_libc_stdout_cookie, OPENMODE_W);
  stdio_open(&__llvm_libc_stderr_cookie, OPENMODE_W);
}
```

Errno and exit

```
// Redirect fault handlers to exit.
void HardFault_Handler() { __llvm_libc_exit(); }
…
// Exit via semihosting
#define ADP_Stopped_ApplicationExit 0x20026
void __llvm_libc_exit(int status) {
  semihosting_call(SYS_EXIT, (const void *)ADP_Stopped_ApplicationExit);
  /* semihosting call doesn't return */
  __builtin_unreachable();
}
// Simple global errno
int *__llvm_libc_errno() {
  static int internal_err;
  return &internal_err;
}
```

Linker Script

```
/* Make the rest of memory available for heap storage
 * LLVM libc denotes heap with [__end, __llvm_libc_heap_limit)
 */
PROVIDE (__llvm_libc_heap_limit = __stack - (DEFINED(__stack_size) ? __stack_size : 4K));
```

arm

Backup

Headers available in standalone build

Peter Smith

Functionality

Standalone headers.

Headers available

- assert.h
- ctype.h
- complex.h
- dirent.h
- dlfcn.h
- elf.h
- errno.h
- fcntl.h
- features.h
- fenv.h float.h
- inttypes.h
- limits.h
- link.h
- locale.h
- malloc.h
- math.h
- setjmp.h
- signal.h
- spawn.h
- stdbit.h
- stdfix.h
- stdint.h
- stdio.h
- stdlib.h
- string.h
- strings.h
- termios.h
- threads.h
- time.h
- uchar.h
- unistd.h
- wchar.h