Supporting native_posix on macOS

Zephyr Developer Summit, 2022

Chris Friedt
Embedded SWE, ASIC FW

ABOUT ME

EE / RFE by training, SWE by trade

17 years in mobile, consumer & industrial IoT, SDR

Long time Embedded Linux dev and Open Source advocate

Started in Zephyr as individual contributor in 2019

Picked up by Meta just over 1 year ago

COMMUNITY ROLES

Linux Plumbers IoT Microconference Organizer

- 12-14 September, Dublin, Ireland
- “IoTs a 4-Letter Word” (CFP)

Zephyr LTSv2 Release Manager

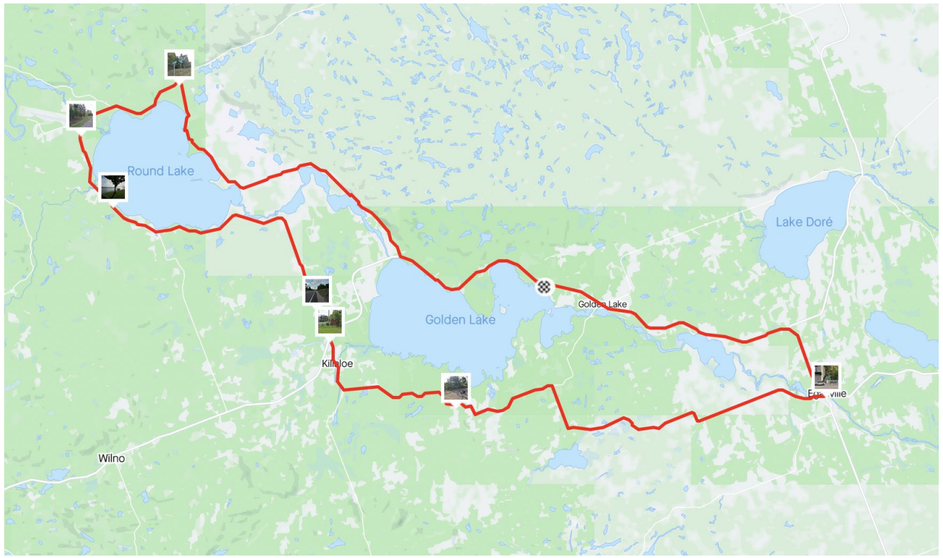
- Current LTS is v2.7.2, 145 changes since v2.7.1
- Backports for CVE + Bug Fixes for 2 years
- 6 month overlap between LTS releases (Process)

Zephyr TSC Member

[Summit Spin '22 Jersey]


gentoo linux™
Zephyr®
sched
Kconfig
timers
cc26xx
cc13xx
posix
spi
atomics
riscv
ieee802154
net
dns_sd
pthread
tls
libc
thread
sensor
clock
soc
pinctrl
emul
mdns
release
cc1352r1_launchxl
boards
cc1352r_sensortag
greybus
Devicetree
gpio
logging
ring_buffer
perf
bluetooth
LTS
kernel
arm
i2c
C++
yocto
PROJECT

Route Map



01 native_posix 🐧

02 macOS 🍎

03 Obstacles 🚧

04 In the garage 🔧

05 The road ahead 🚏

01 native_posix 🐧

What is native_posix?

A convenience boards for testing & development.

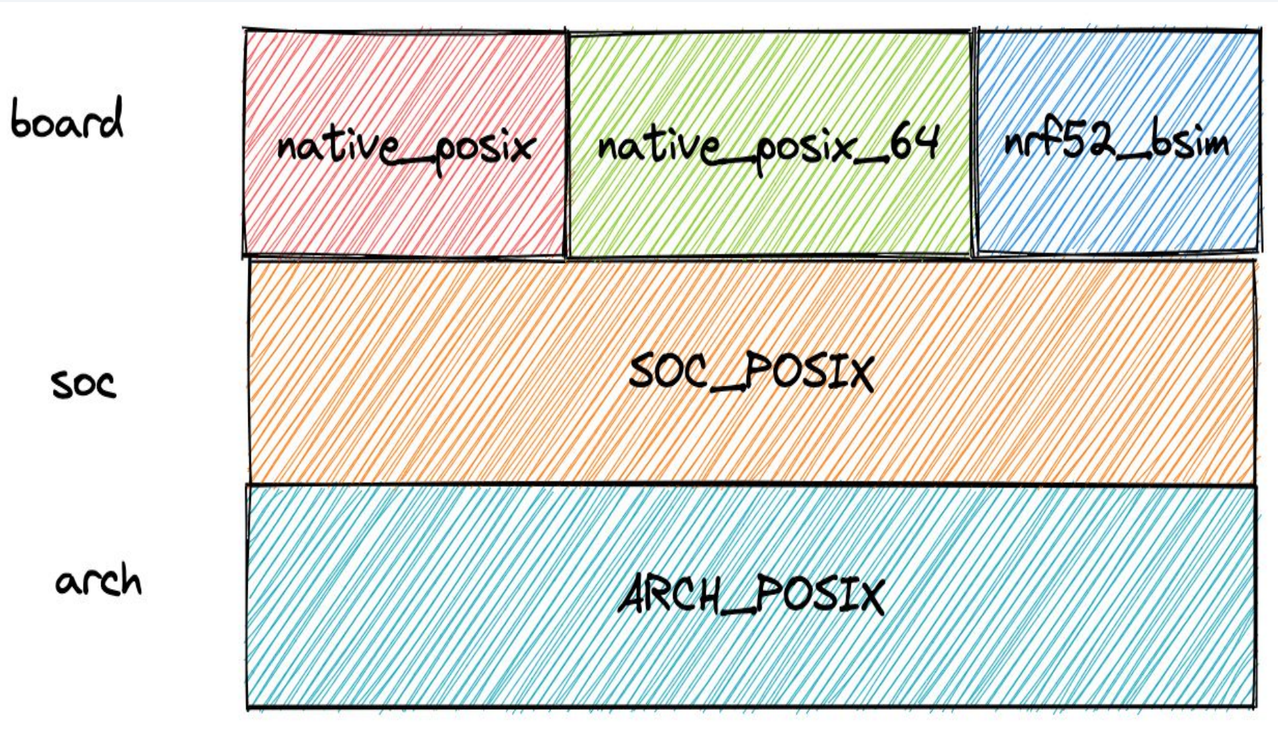
A collection of emulated devices and peripherals

The SoC layer contains functionality that would normally be at that layer for a physical chip. For example, special boot code, any special linker sections required, etc.

The POSIX arch layer contains core functionality that, for instance, maps zephyr's threading model to the POSIX model.

CONFIG_POSIX_API != CONFIG_ARCH_POSIX

CONFIG_POSIX_API is mutually exclusive with CONFIG_ARCH_POSIX




NATIVE C LIBRARY
ZEPHYR APPLICATION
MINIMAL C LIBRARY
imgflip.com

Why native_posix?

faster development cycle

functionally accurate

more resources (RAM, GHz)

native debugging

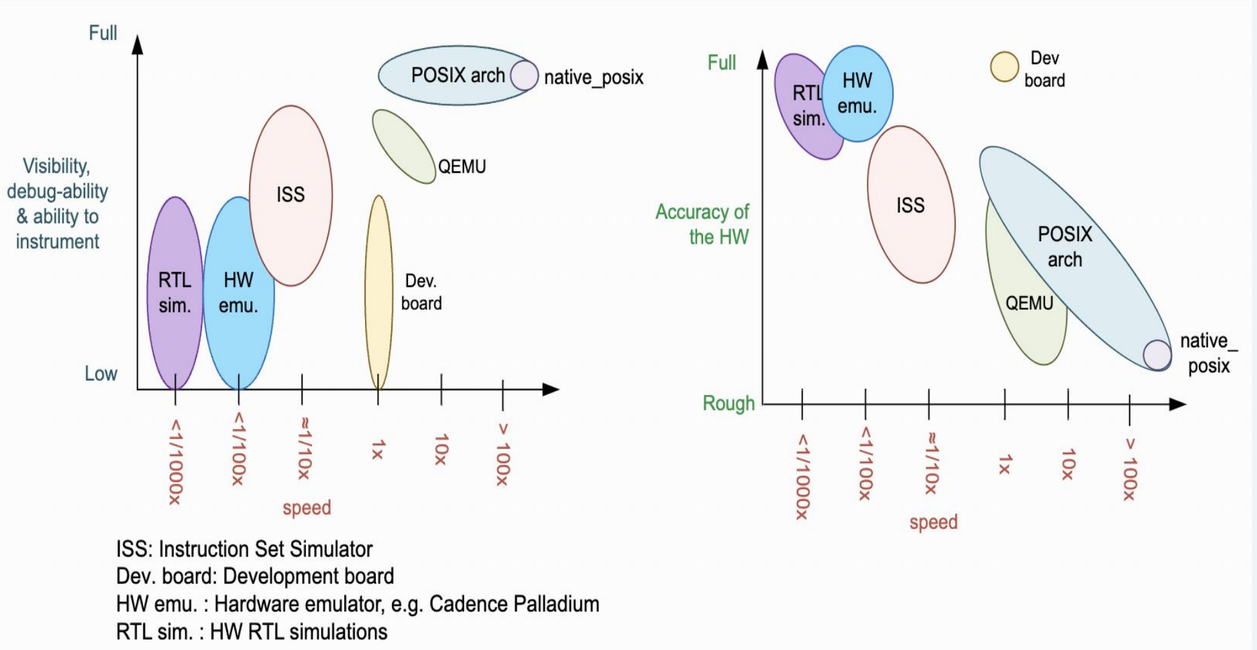
ASan (Address Sanitizer)

UBSan (Undefined Behaviour Sanitizer)

valgrind (memory leak detection)

gcov (code coverage)

perf (flamegraphs!)



https://bit.ly/3z3Zw4f

Why native_posix?

Emulated devices and peripherals!

Mentioned several times already this week

- Keynote by Simon Glass
- Talk by Keith Short
- Virtual Talk by Yuval Peress

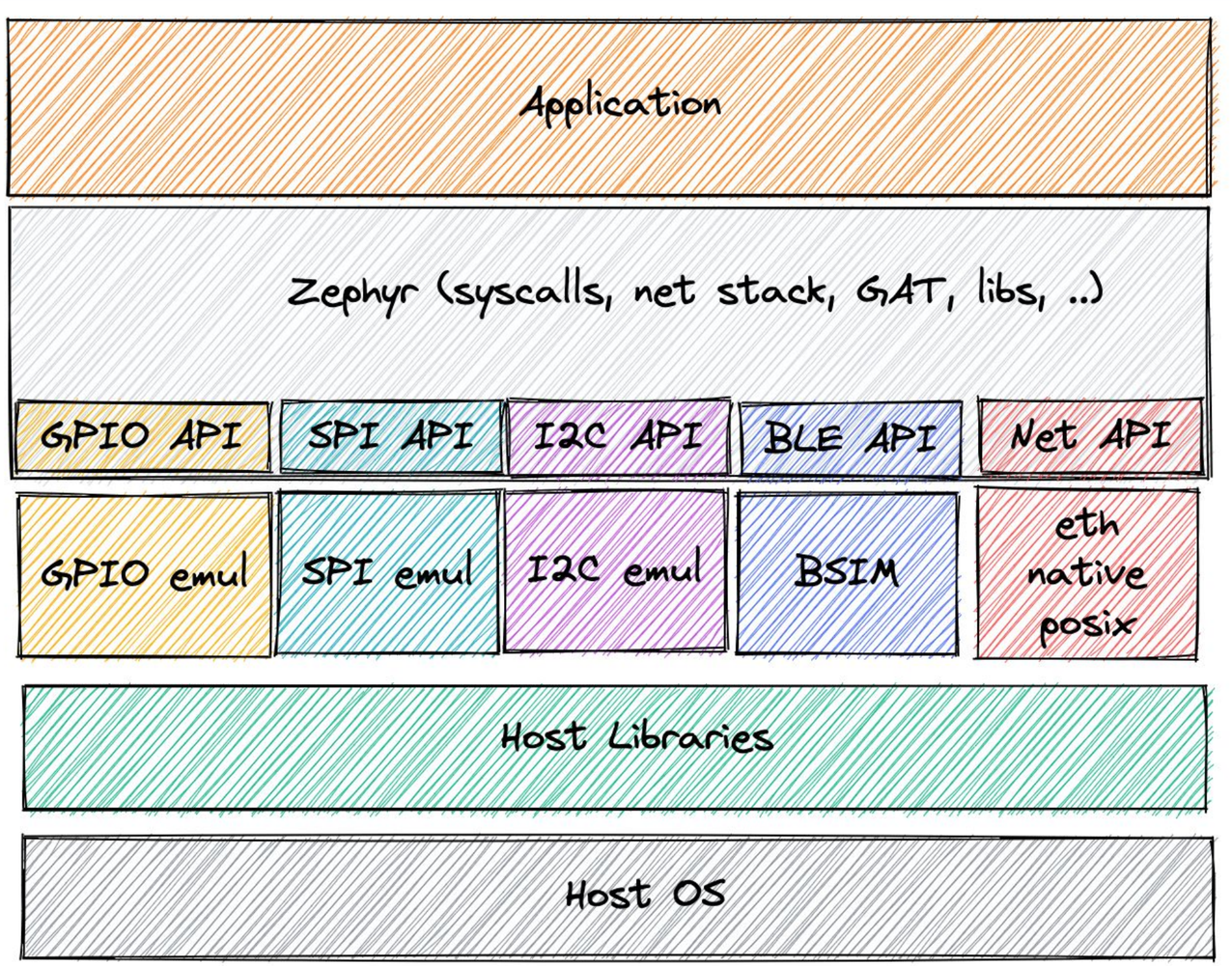
Fail Fast in Testing

Large category of bugs resolved at the API level (HW agnostic)

Testing on real HW takes significantly longer

Test the exact code paths through the app and OS

Inconsistent results with functional & system tests points to the bug



Why native_posix on macOS?

If we are inclusive in the Zephyr Project, by supporting a POSIX platform other than Linux, then we cast a larger net, and are able to attract the best developers in every community.

Everyone has a preference.

If we can support macOS, which is the only major POSIX OS that does not use ELF, then chances are we can support any POSIX OS.

How native_posix works today

```
NATIVE_TASK(np_add_uart_options, PRE_BOOT_1, 11);
NATIVE_TASK(np_cleanup_uart, ON_EXIT, 99);
```

Selected SW components register a native task with the macro

- Function pointer
- Level
- Priority

Tasks are sorted via a linker script fragment

Specified native tasks are executed prior to Zephyr startup

soc.c

soc > posix > inf_clock > C soc.c > run_native_tasks(int)

```
221
222 /**
223  * @brief Run the set of special native tasks corresponding to the given level
224  *
225  * @param level One of _NATIVE_*_LEVEL as defined in soc.h
226  */
227 void run_native_tasks(int level)
228 {
229     extern void (*__native_PRE_BOOT_1_tasks_start[])(void);
230     extern void (*__native_PRE_BOOT_2_tasks_start[])(void);
231     extern void (*__native_PRE_BOOT_3_tasks_start[])(void);
232     extern void (*__native_FIRST_SLEEP_tasks_start[])(void);
233     extern void (*__native_ON_EXIT_tasks_start[])(void);
234     extern void (*__native_tasks_end[])(void);
235
236     static void (**native_pre_tasks[])(void) = {
237         __native_PRE_BOOT_1_tasks_start,
238         __native_PRE_BOOT_2_tasks_start,
239         __native_PRE_BOOT_3_tasks_start,
240         __native_FIRST_SLEEP_tasks_start,
241         __native_ON_EXIT_tasks_start,
242         __native_tasks_end
243     };
244
245     void (**fptr)(void);
246
247     for (fptr = native_pre_tasks[level]; fptr < native_pre_tasks[level+1];
248         fptr++) {
249         if (*fptr) { /* LCOV_EXCL_BR_LINE */
250             (*fptr)();
251         }
252     }
253 }
```

How native_posix works today

The header file posix_cheats.h is included from the command line when preprocessing every source file, and it has two main responsibilities. The first of those is redefining main() to zephyr_app_main(). The second is redefining any referenced POSIX functions (e.g. “zap_open”). With that, any references to POSIX functions result in undefined symbol errors and the Zephyr binary cannot be linked. In the POSIX SoC and arch layers, the macro POSIX_NO_CHEATS disables the default behaviour of “zapping” POSIX symbols.

posix_cheats.h

arch > posix > include > C posix_cheats.h > ...

```
43  #ifndef main
44  #define main(...) zephyr_app_main(__VA_ARGS__)
45  #endif
```

main.c

boards > posix > native_posix > C main.c > ...

```
51  /**
52   * This is the actual main for the Linux process,
53   * the Zephyr application main is renamed something else thru a define.
54   */
55  int main(int argc, char *argv[])
56  {
57      run_native_tasks(_NATIVE_PRE_BOOT_1_LEVEL);
58
59      native_handle_cmd_line(argc, argv);
60
61      run_native_tasks(_NATIVE_PRE_BOOT_2_LEVEL);
62
63      hwm_init();
64
65      run_native_tasks(_NATIVE_PRE_BOOT_3_LEVEL);
66
67      posix_boot_cpu();
68
69      run_native_tasks(_NATIVE_FIRST_SLEEP_LEVEL);
70
71      hwm_main_loop();
72
73      /* This line should be unreachable */
74      return 1; /* LCOV_EXCL_LINE */
75  }
76
```

How native_posix works today

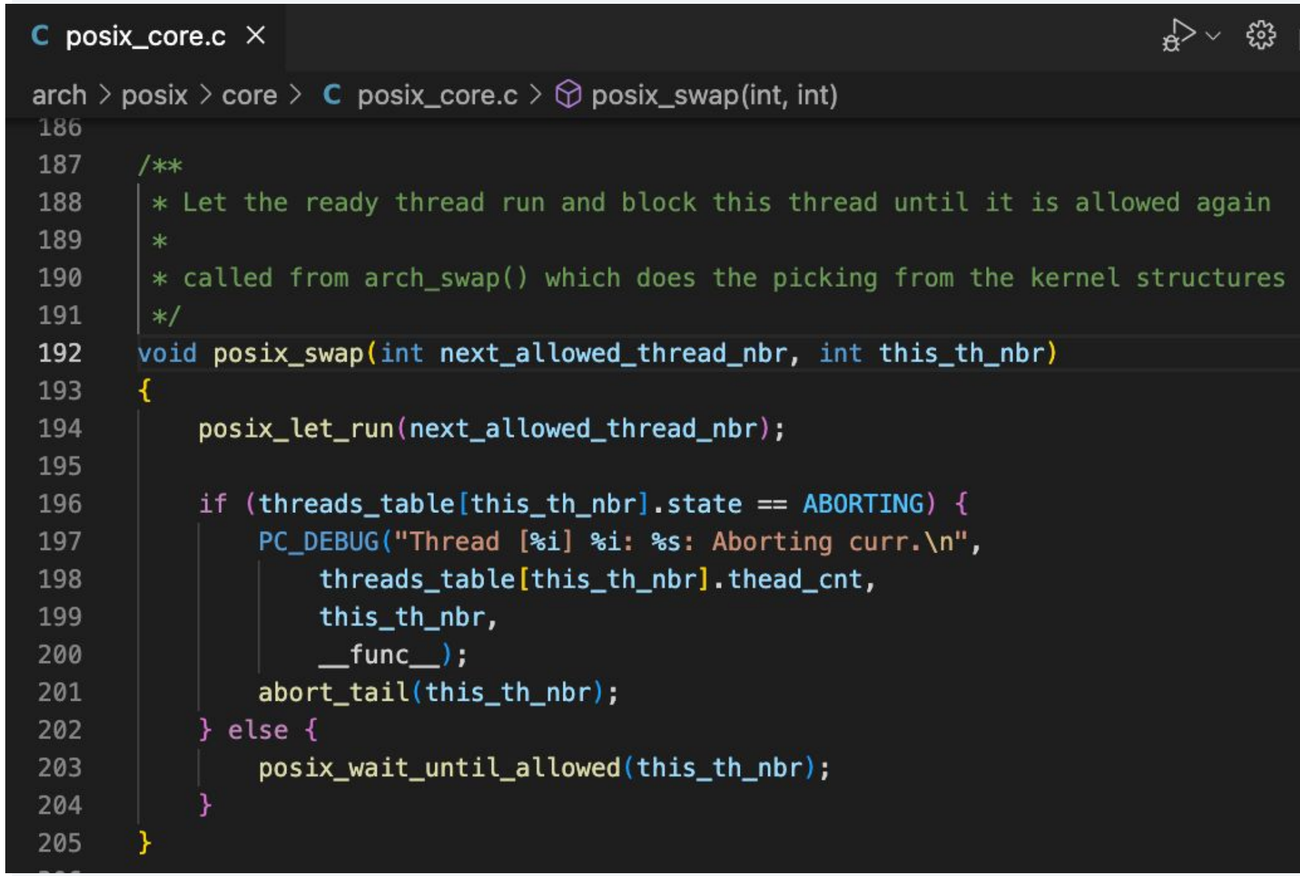
posix_boot_cpu() spawns a pthread which calls z_cstart()

Multithreading is initialized

The usual Zephyr SYS_INIT() and device init functions are invoked, via ordered iterable section.

arch_swap() calls posix_swap()

Off to the races!



02 macOS 🍎

02 macOS

macOS History

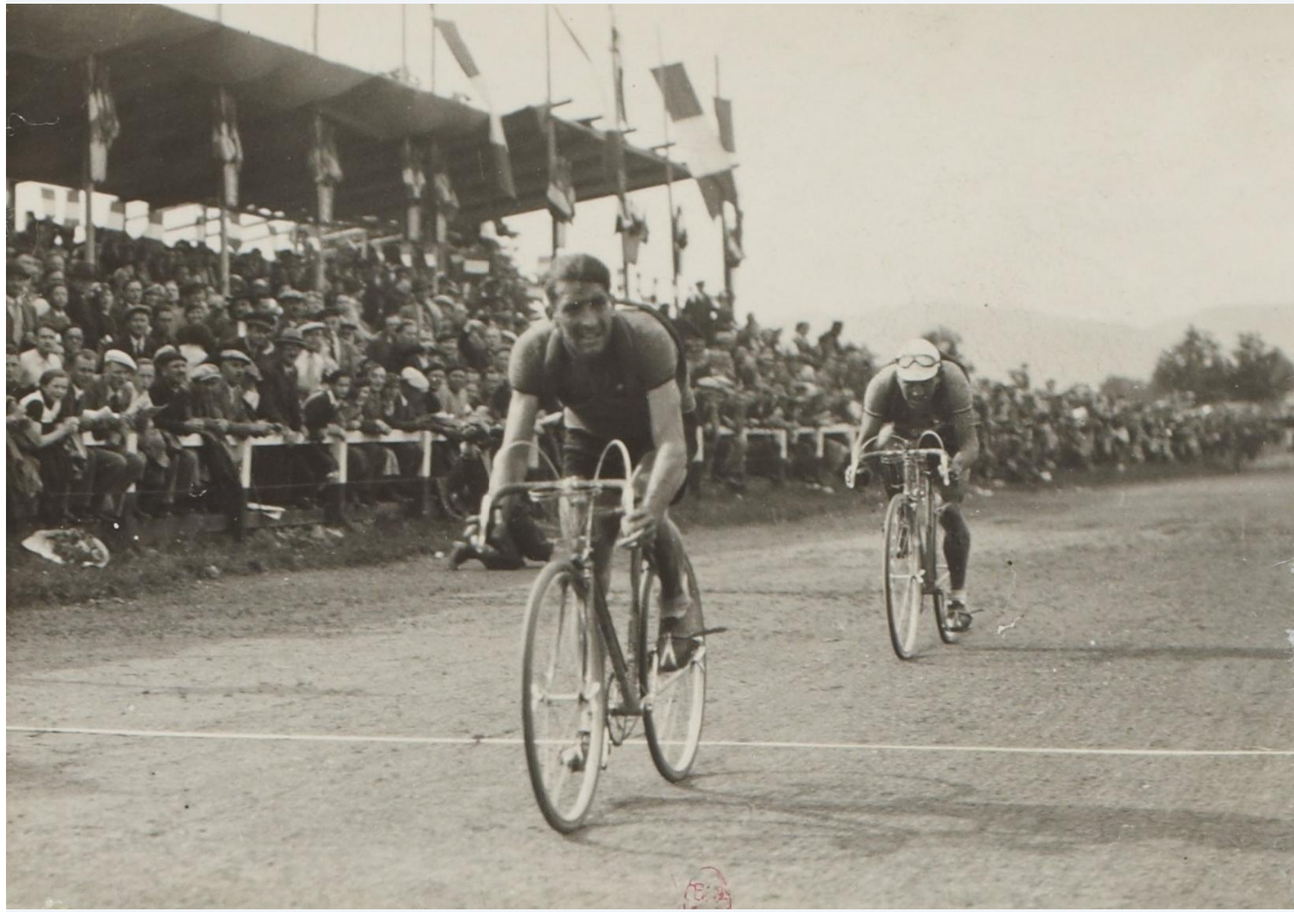
Macintosh Operating System (v9 in 1999)

- “The Best Operating System Ever”
- Cooperative multitasking

Mac OS X (2000 to 2016)

- Built on technology developed by NeXT, CMU
- XNU kernel (4.3 BSD kernel + Mach)
- Hybrid (monolithic and microkernel components)
- FreeBSD + Apple utils, NeXTSTEP, etc
- POSIX support, X11, etc

macOS (2016 to present)

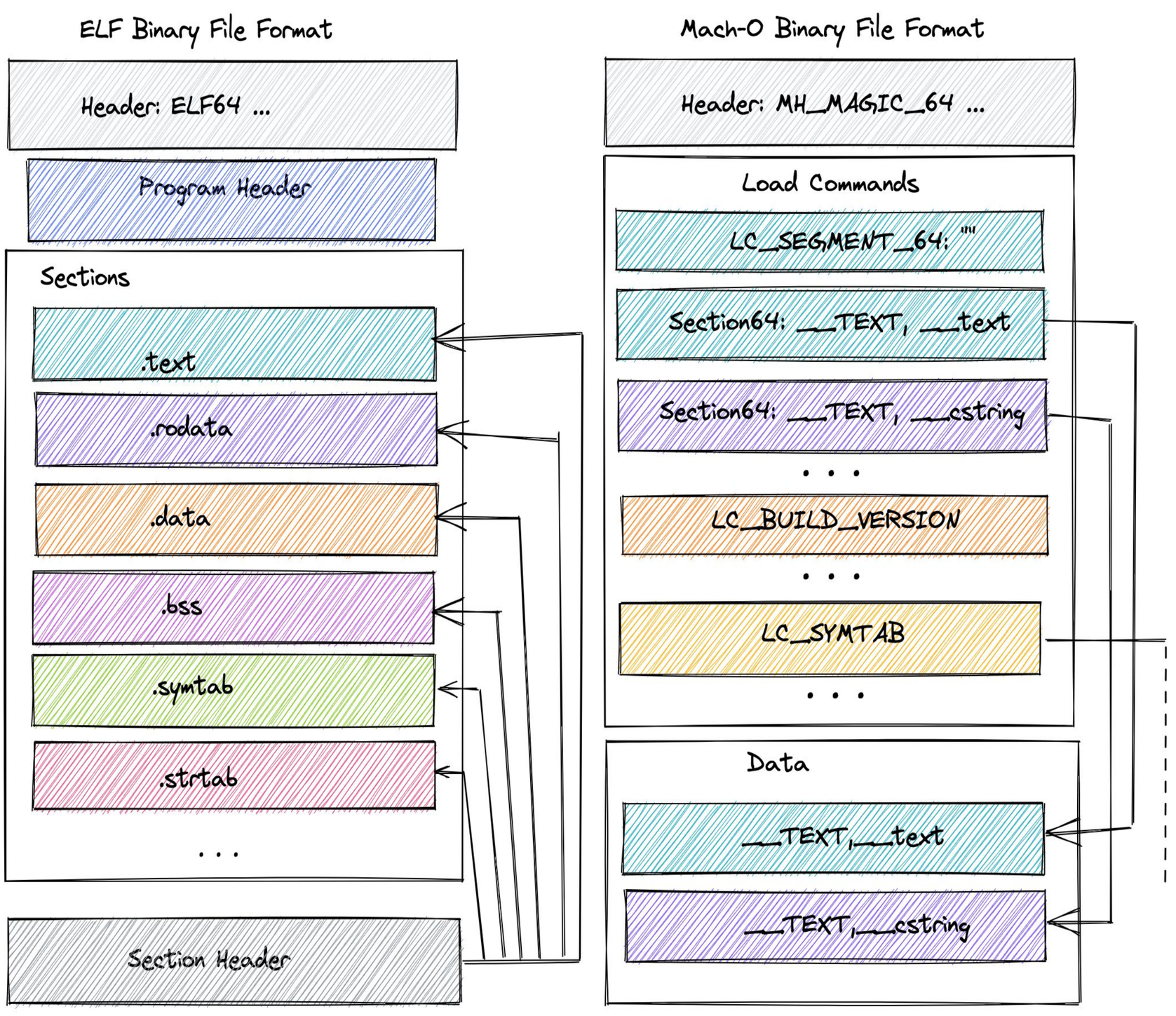


Linux vs macOS

	Linux	**macOS**
POSIX	Yes! (GNU)	Yes! (BSD)
Kernel	monolithic	~microkernel
Executable Format	ELF	Mach-O
Main library	libc.so.6	~libSystem.B.dylib
POSIX system calls	SWI + syscall nr + regs	fn calls + msg passing

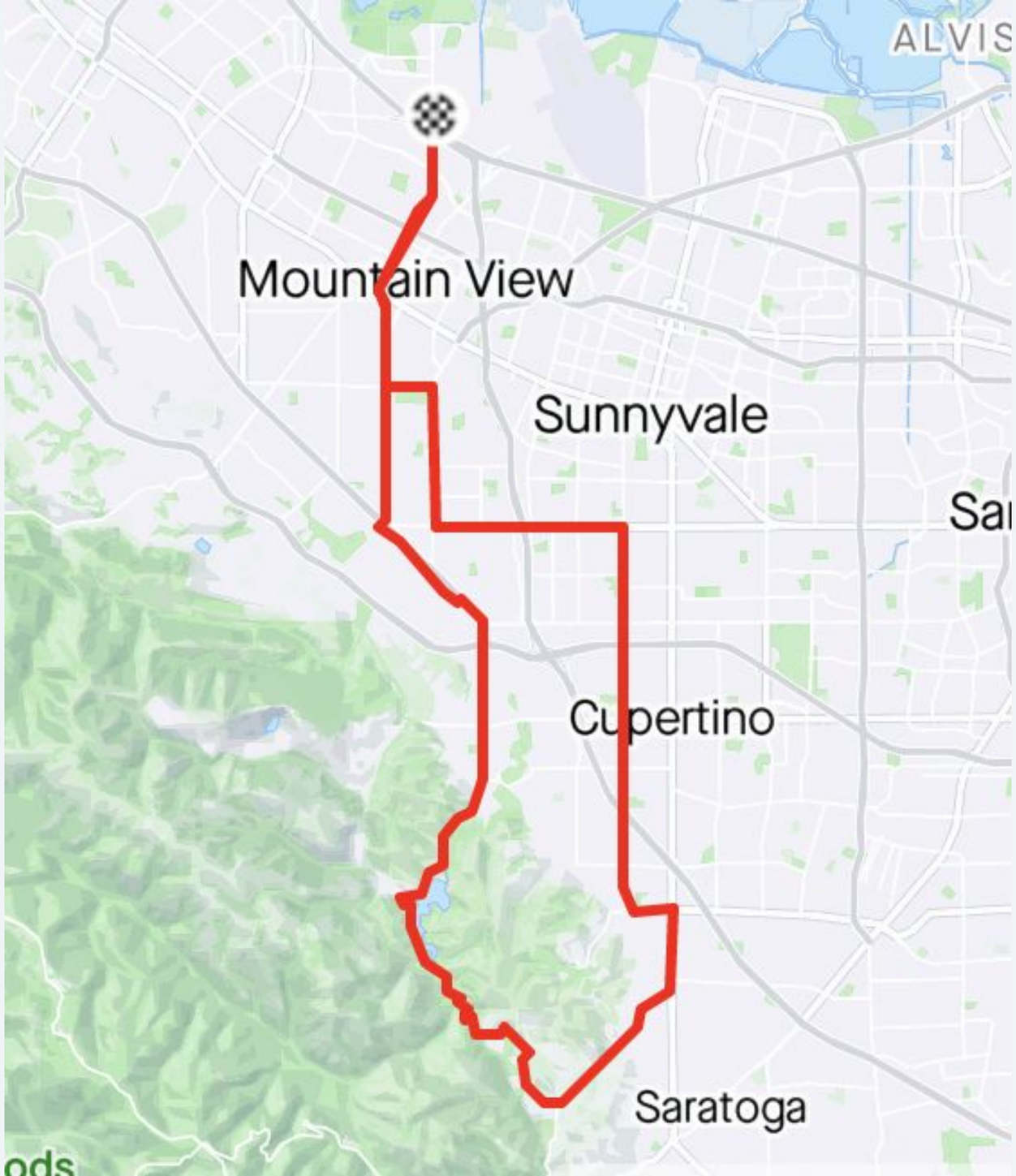
ELF vs Mach-O

Both binary object file formats achieve the same purposes. Mach-O is command oriented whereas ELF is section oriented. Mach-O has a header, ELF has a header. Notably, in Mach-O, each section is associated with a segment, and sections often holds a reference to a distant region within Data, whereas ELF sections are mostly atomic. The section header is also clearly at the end in ELF, and ELF uses back references rather than forward references.



Segment Mapping

ELF	Mach-O
.text	__TEXT,__text
.rodata	__DATA,__const
.bss	BSS,__bss, BSS,__common
.data	__DATA,__data
.got	__DATA,__got



03 Obstacles 🚧

The Mach-O Binary Format

Only executable format supported on macOS

Does not support weak symbol visibility

> warning: '__weak' only applies to pointer types; type here is 'void'
> duplicate symbol '_sys_clock_set_timeout'

Does not support symbol aliases

Requires a segment name for every section name

> error: argument to 'section' attribute is not valid for this target:
> mach-o section specifier requires a segment and section separated by a comma

Segment and section names must be 1 <= n <= 16 characters

> error: argument to 'section' attribute is not valid for this target:
> mach-o section specifier requires a section whose length is between 1 and 16 characters

ELF Hacks

Zephyr’s post-compile stages use ELF-hacks to generate constant data

- gen_offset_header.py (_OFFSET and _SIZEOF => .h)
- gen_handles.py (device handles)

Great for reducing binary size

Mach-O tooling lags far behind ELF tooling (e.g. pyelfultils / machotools)

Toolchain Abstractions

Relatively minor

The .type directive is not supported in assembly

Apple ld does not support --whole-archive

Apple ld does not support GNU linker scripts

- does support an order file
- not entirely clear if order_file is sufficient
- GNU linker scripts also generate symbols

No static linkage (all Apple executables are dynamic)

```
-order_file file
        Alters the order in which functions and data are laid out.  For
        each section in the output file, any symbol in that section that
        are specified in the order file file is moved to the start of its
        section and laid out in the same order as in the order file file.
        Order files are text files with one symbol name per line.  Lines
        starting with a # are comments.  A symbol name may be optionally
        preceded with its object file leaf name and a colon (e.g.
        foo.o:_foo).  This is useful for static functions/data that occur
        in multiple files.  A symbol name may also be optionally preceded
        with the architecture (e.g. ppc:_foo or ppc:foo.o:_foo).  This
        enables you to have one order file that works for multiple
        architectures.  Literal c-strings may be ordered by by quoting the
        string (e.g. "Hello, world\n") in the order file.
```

04 In the garage 🔧

Prototype

Bit of a dirty hack (#37123)

Simple workarounds for Toolchain abstractions

Use macros to create structs with __attribute__((constructor))

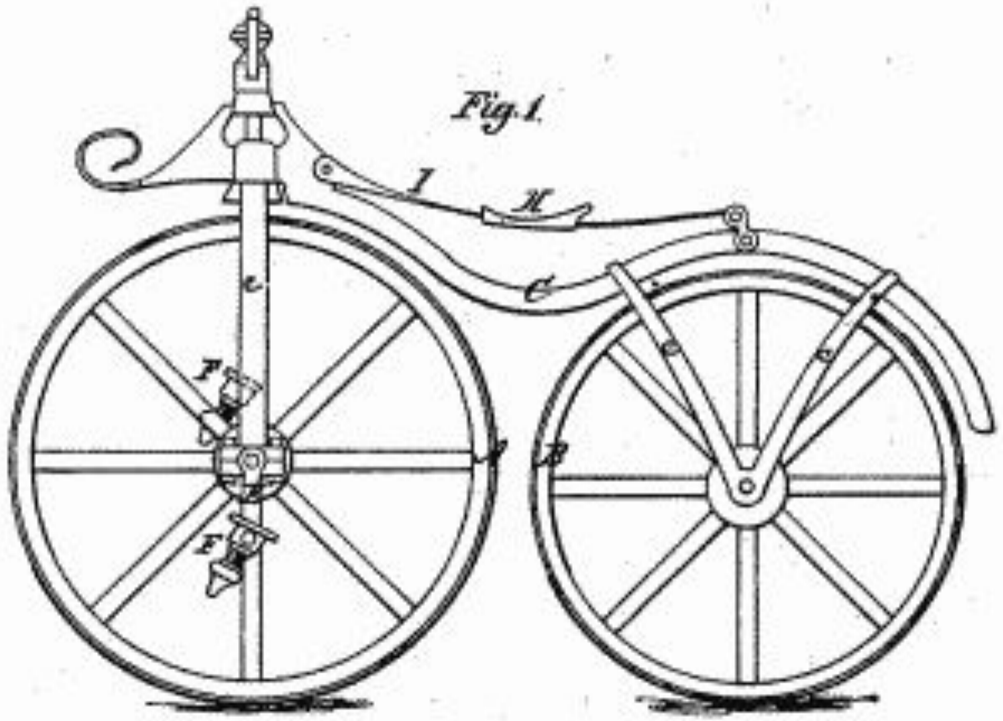
Have each constructor append details to a C++ vector<>

NATIVE_TASK() and SYS_INIT() only

Too many iterable sections to be feasible

Required manually tweaking section names / dumping iterable sections into 1 segment, section

Hard to pedal



Spare Parts Build

Basically the same as the prototype

Try to use Homebrew GCC

Same linker issues (same linker)

Completely ignores weak attribute

⇒ symbol collisions

⇒ at least better than UB

Might work for a ride or two but not reliably

Pro Model

Modify LLVM (clang, clang++) (Changes)

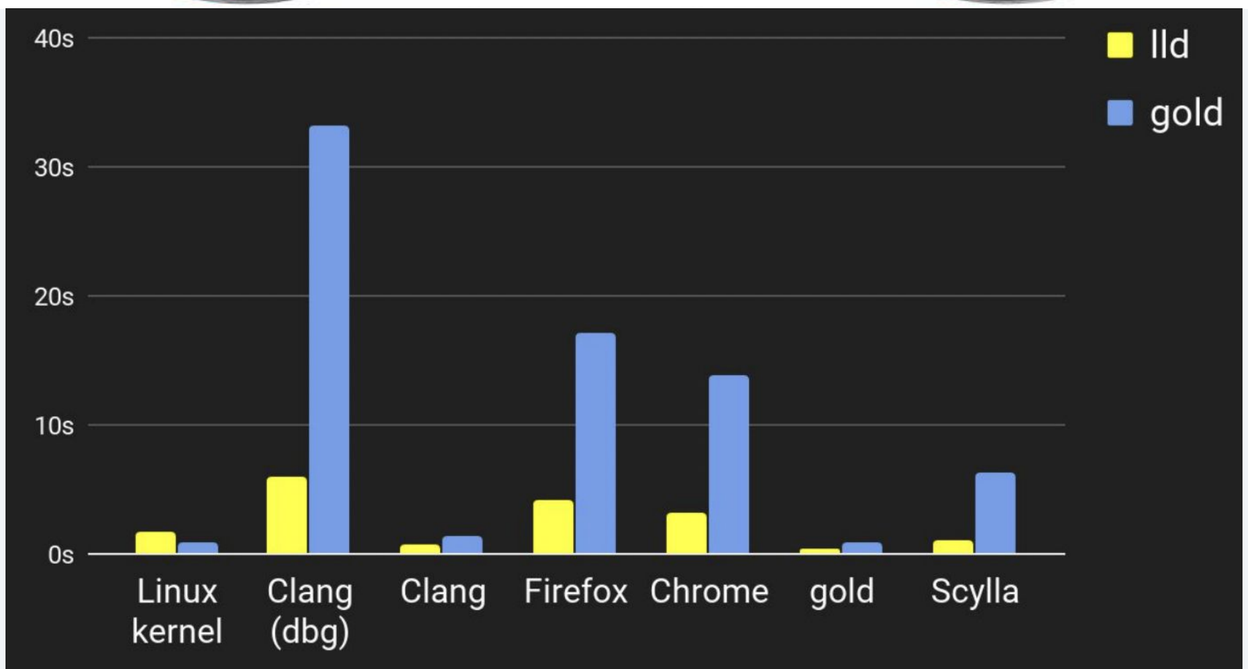
- -fhash-section-names=N
- -fhashed-section-names=outfile
- Sha256 of section name
- Base64 Encoding
- Truncate to N

We likely want LLVM to act as a drop-in replacement for gcc

- Including support for GNU Linker scripts on macOS

LLVM's linker, LLD is significantly faster than the GNU linker (Source)

- Additional talk: FOSDEM 2019, Peter Smith, Linaro



Promising results

Details in [#40709]

Short section names are preserved as-is

Longer section names truncated to 16

Output file shows a clear mapping

```
% objdump --headers build/zephyr/zephyr.exe

build/zephyr/zephyr.exe:        file format mach-o arm64

Sections:
Idx Name              Size     VMA
  0 __text            0000cbdc 000
  1 __stubs           000001d4 000
  2 __stub_helper     000001ec 000
  3 __cstring         00000f55 000
  4 __const           0000014d 000
  5 __literal8        00000058 000
  6 __literal16       00000010 000
  7 __got             00000090 000
  8 __const           00000080 000
  9 __la_symbol_ptr   00000138 000
 10 __data            00000460 000
 11 ee505i1iDSJ38lW6  00000020 000
 12 X3OkufZugPkMRV0u  00000008 000
 13 gSbov272cOSyitB9  00000008 000
 14 PWCnD+vO5fLbOTNC  00000010 000
 15 ISGPhW3oi0+QmD32  00000006 000
 16 .z_devstate       00000004 000
 17 g689RYj/NgI8BJm1  00000030 000
 18 KPvctDvcsEzL2GeJ  00000010 000
 19 kSBlhTrw6lGfb4bH  00000400 000
 20 2ykHVrz1H6C6zM7x  00000800 000
 21 M7LnbayK+Apx90Az  00000100 0000000100015228 DATA
 22 DSRqIUjHqO6d+Nny  00000040 0000000100015328 DATA
 23 yXwPRdgDQM/TZ0qv  00000030 0000000100015368 DATA
 24 AivJBhsGJLg4nCaa  00000050 0000000100015398 DATA
 25 0G3gaNFfoLgkTC6u  00000640 00000001000153e8 DATA
 26 xcc74YkFJpO/l+79  00000030 0000000100015a28 DATA
 27 kOons3eUg3xDONU5  00000050 0000000100015a58 DATA
 28 66dcYG27Liq2h08v  000004b0 0000000100015aa8 DATA
 29 oinUk81sKcoS+CT2  00000010 0000000100015f58 DATA
 30 U9Ufj7zPekAbDTSR  00000400 0000000100015f68 DATA
 31 __bss             00000555 0000000100016370 BSS
 32 __common          00000239 00000001000168c8 BSS
```

```
._k_mutex.static.fdtable_lock   ee505i1iDSJ38lW6
.z_init_PRE_KERNEL_199_ PWCnD+vO5fLbOTNC
.__device_handles_pass1 ISGPhW3oi0+QmD32
.z_device_PRE_KERNEL_20_        g689RYj/NgI8BJm1
.z_init_PRE_KERNEL_20_  KPvctDvcsEzL2GeJ
.z_init_PRE_KERNEL_130_ DSRqIUjHqO6d+Nny
.noinit."ZEPHYR_BASE/kernel/init.c".0   kSBlhTrw6lGfb4bH
.noinit."ZEPHYR_BASE/kernel/init.c".1   M7LnbayK+Apx90Az
.noinit."ZEPHYR_BASE/kernel/init.c".2   2ykHVrz1H6C6zM7x
.z_init_PRE_KERNEL_130_ DSRqIUjHqO6d+Nny
.noinit."ZEPHYR_BASE/kernel/mailbox.c".0        AivJBhsGJLg4nCaa
._k_stack.static.async_msg_free yXwPRdgDQM/TZ0qv
.noinit."ZEPHYR_BASE/kernel/mailbox.c".1        0G3gaNFfoLgkTC6u
.z_init_PRE_KERNEL_130_ DSRqIUjHqO6d+Nny
.noinit."ZEPHYR_BASE/kernel/pipes.c".0  kOons3eUg3xDONU5
._k_stack.static.pipe_async_msgs        xcc74YkFJpO/l+79
.noinit."ZEPHYR_BASE/kernel/pipes.c".1  66dcYG27Liq2h08v
.z_init_PRE_KERNEL_130_ DSRqIUjHqO6d+Nny
.noinit."ZEPHYR_BASE/kernel/system_work_q.c".0  U9Ufj7zPekAbDTSR
.z_init_POST_KERNEL40_  oinUk81sKcoS+CT2
```

Assisted Riding

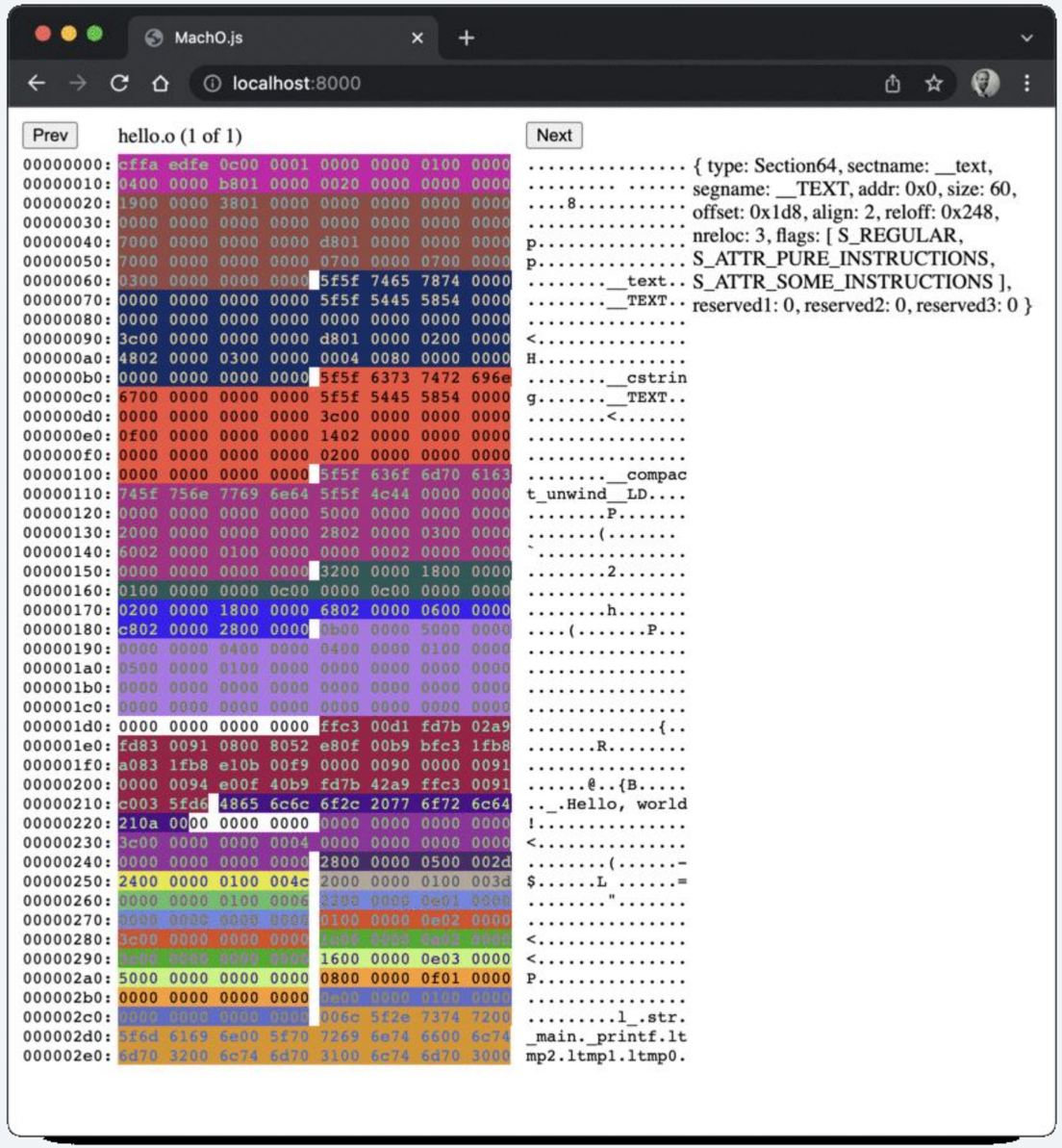
Macho.js a useful learning and visualization tool for exploring Mach-O files

Modifying gen_offset_header.py relatively easy to do in python wrapping around nm (Source)

Some progress with machotools (Mach-O variant of pyelfutils)

Strategy: make a “toy” linker in Python called LowD (Source)

- contrived test cases to start (hello.c, etc)
- leverage clang’s -fuse-ld=LowD option (actually, requires ld and symlink)
- pain point: Apple’s dynamic linker cache as of Big Sur (workaround)
- eventually add a new target to LLD and pretend its an ELF target
- Weird combination of LD64 (Mach-O) and LLD (ELF)



05 The road ahead 

The road ahead

native_posix on macOS

- Continue work on LowD in spare time
- Retrofit to LLVM, accept -fhashed-section-names=infile
- Add section reordering support to machotools
- Update gen_handles.py to handle Mach-O binaries via machotools
- Zephyr support (cmake integration, include/toolchain)
- Make it available for users via the Zephyr SDK [EXPERIMENTAL]
- What to do about weak symbols: use Kconfig to configure software 🤷

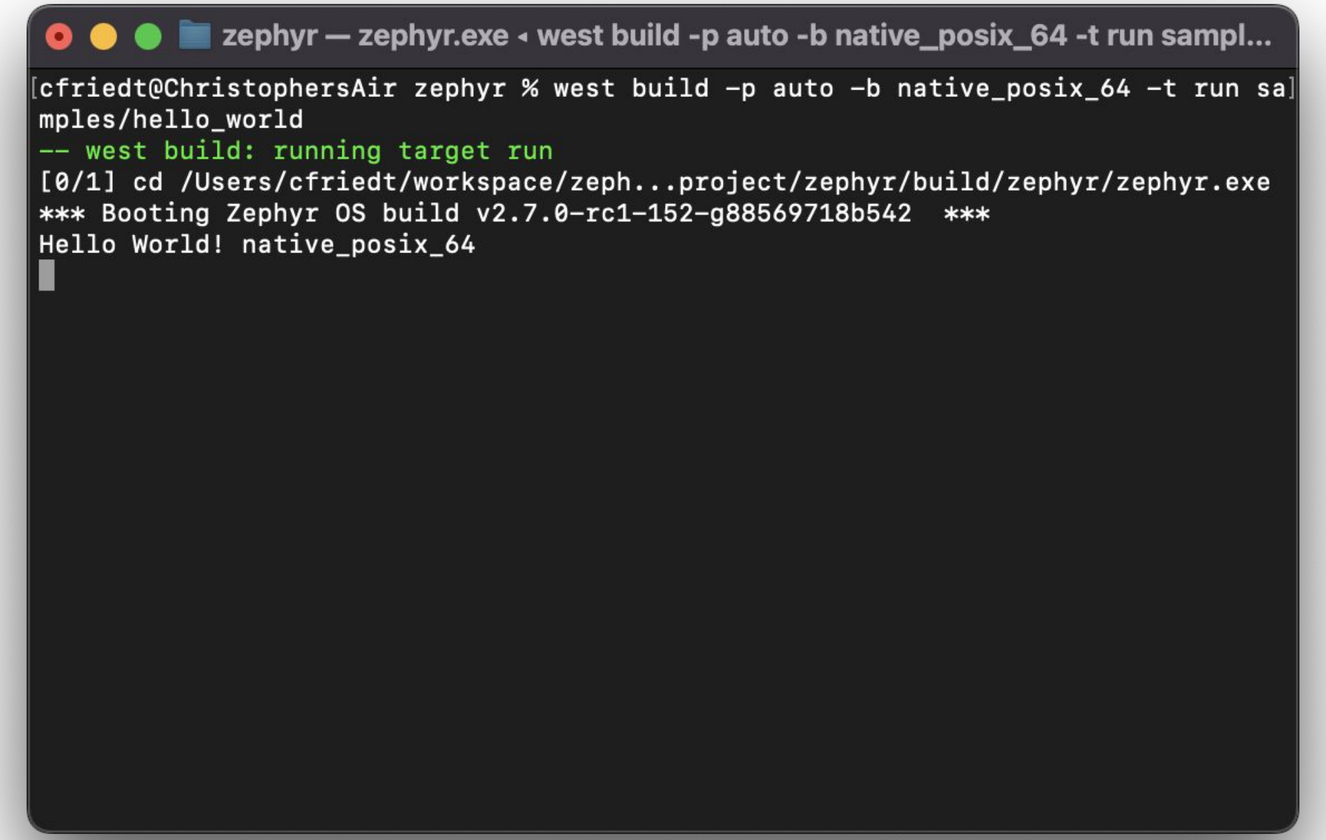
Beyond native_posix

- Qemu network support for macos (#35515)
- Allow CONFIG_POSIX_API with CONFIG_ARCH_POSIX (#45100)

Actually, the Mach-O section name limits are a surprisingly common rant on GitHub

- Other open-source projects could benefit from the work

Thanks You 🙏 Questions?




Meta