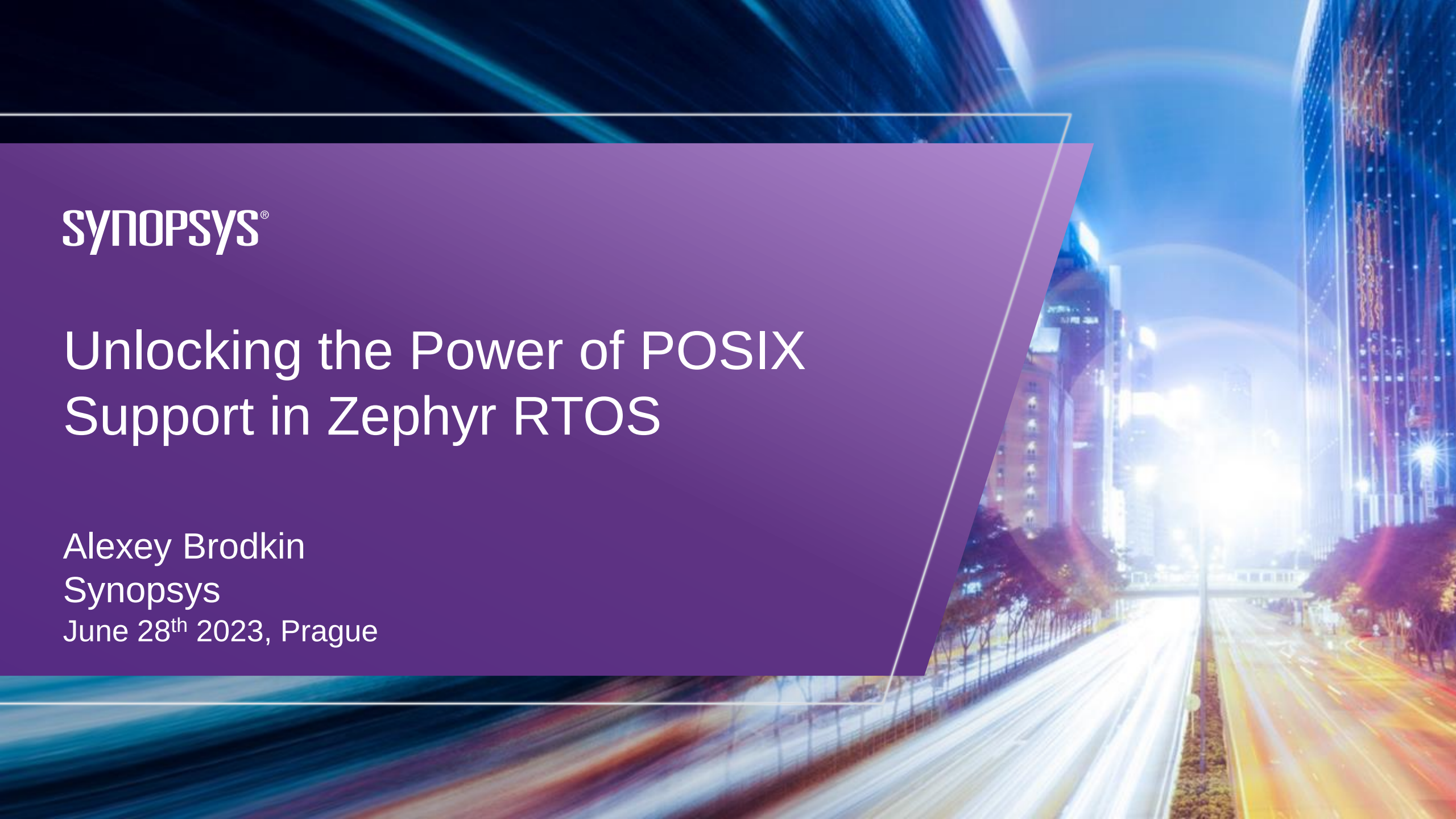


SYNOPSYS®

Unlocking the Power of POSIX Support in Zephyr RTOS

Alexey Brodkin
Synopsys
June 28th 2023, Prague

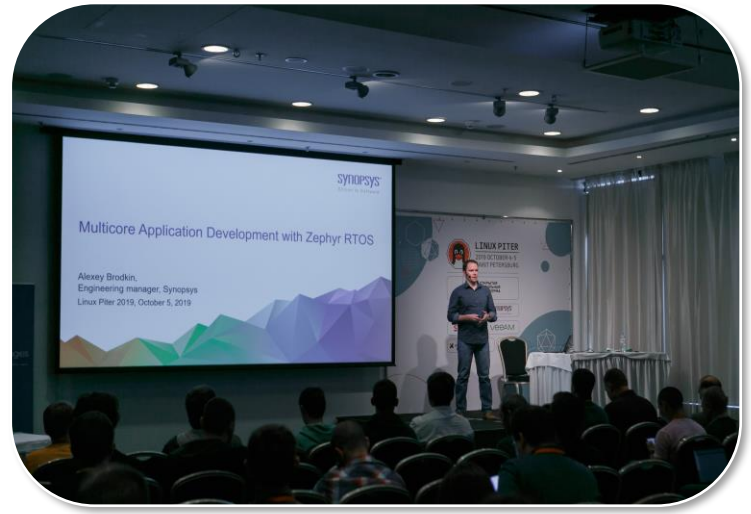
Alexey Brodkin

Engineering manager at Synopsys

5 years in Zephyr development

ARC Synopsys

Manage Zephyr development in Synopsys



Zephyr ambassador

Agenda

- Introduction: Zephyr Project & POSIX
- POSIX Support in Zephyr RTOS
- Use-cases

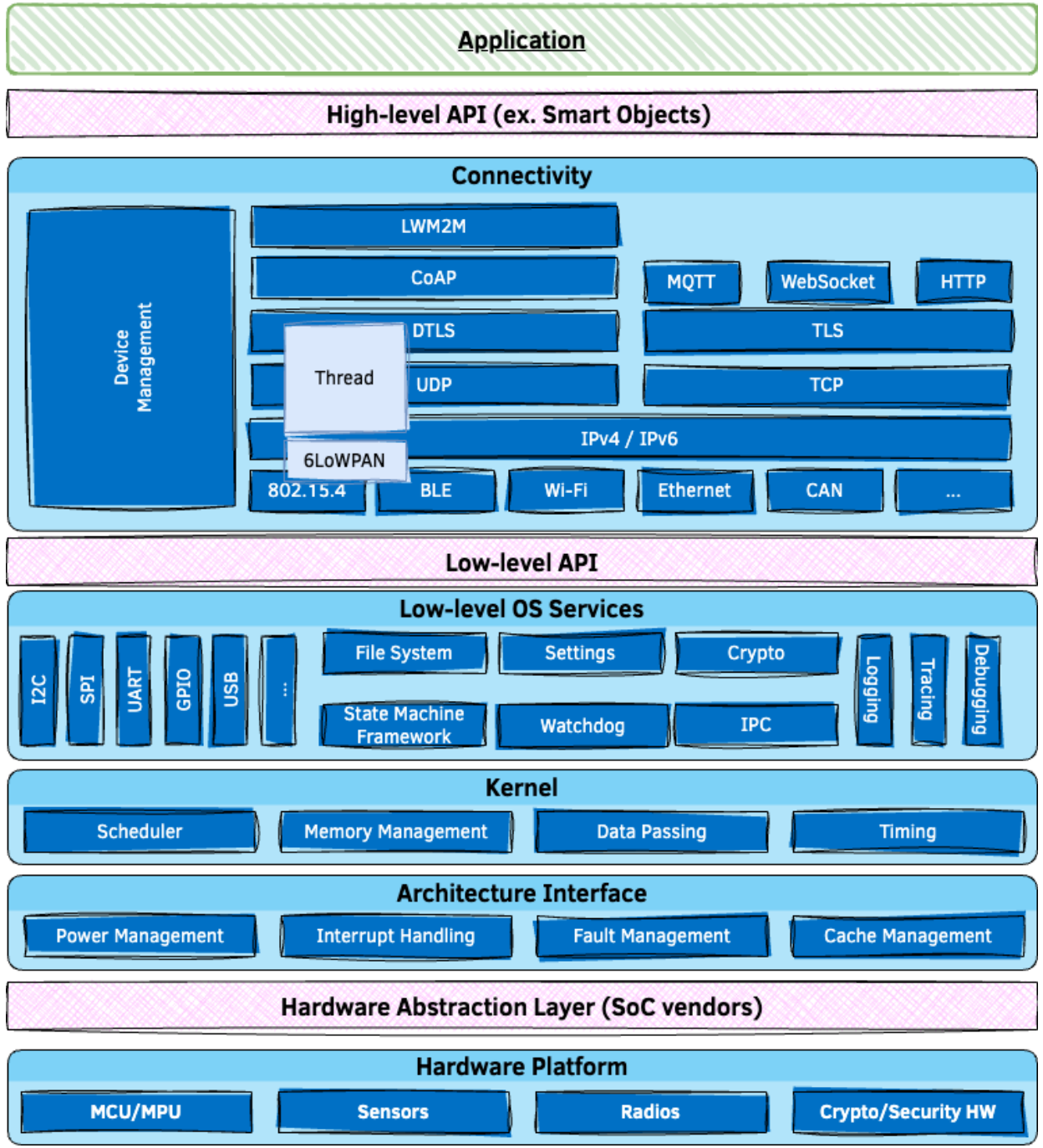
Introduction: Zephyr Project & POSIX

- Zephyr Project
- POSIX

Features overview

- Comprehensive, **lightweight**, kernel & supporting services
  - Fits where Linux is too big
- Inherently **portable** & **secure**
- **Highly connected**
  - Bluetooth 5.0 & BLE
  - Wi-Fi, Ethernet, CANbus, …
  - IoT protocols: CoAP, LwM2M, MQTT, OpenThread, …
  - USB & USB-C
- **Developer-friendly**
  - Logging, tracing, debugging, built-in shell, Windows/Linux/macOS support, …

Architecture



Basic facts about POSIX

- Acronym for **P**ortable **O**perating **S**ystem **I**nterface.
- **IEEE Std 1003.1** “Portable Operating System Interface (POSIX™)”
  - Originally focused on the UNIX systems
- **IEEE Std 1003.13** “Standardized Application Environment Profile (AEP)—POSIX™ Realtime and Embedded Application Support”
- First version in 1988, latest version in 2018
- Specifies application programming interfaces (APIs) at the source level, and is about source code portability, not binary compatibility

POSIX threads	Device I/O functions	Basic C language functions
• `pthread_create()` • `pthread_join()` • `pthread_mutex_lock()` • `pthread_cond_wait()`	• `open()` • `close()` • `fputc()` • `fputc()`	• `abs()` • `atoi()` • `div()` • `rand()`

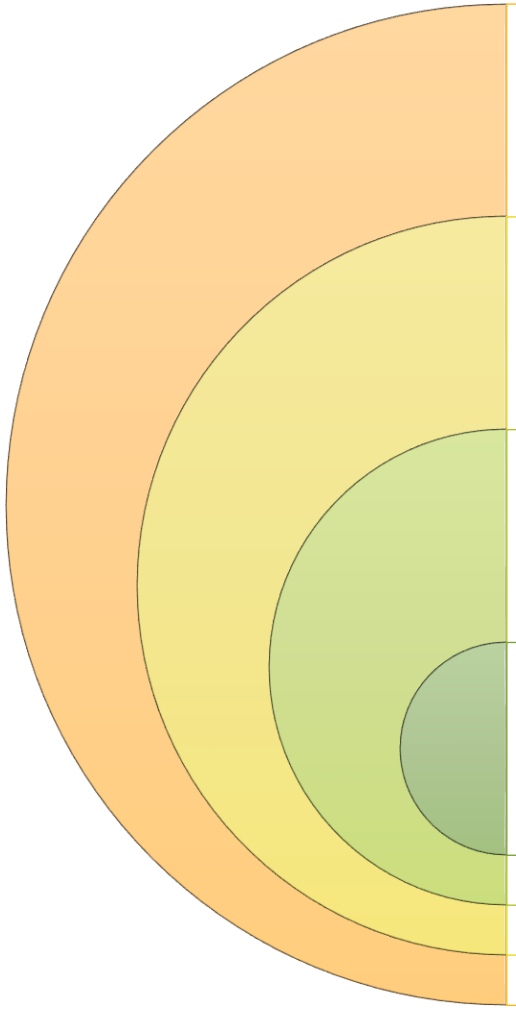
POSIX in the context of embedded systems

It’s all about source code portability

- Portable, cross platforms libraries
- Same application running on multiple platforms
- OS/execution environment independence

POSIX.13 Realtime and Embedded Application Support

IEEE Std 1003.13-2003, Realtime System Profiles: PSE51-54



PSE54 Multi-purpose real-time	• Shell & utilities • Multiple users • Full file system
PSE53 Dedicated real-time	• Networking • Asynchronous I/O • Multiple processes
PSE52 Realtime controller	• Simple file system • Message queues • Tracing
PSE51 Minimal real-time	• C language support • Signals • Single process • Threads

POSIX Support in Zephyr RTOS

- The history of POSIX support in Zephyr RTOS
- Overview of Zephyr's POSIX implementation

"POSIX Roadmap for LTSv3"
by Chris Friedt
on June 29th

History of POSIX support in Zephyr RTOS

2017

- Jul: POSIX threads base (barriers, condition variables, mutexes)
- Oct: “native_posix” board

2018

- Feb: POSIX clock, sleep, timers, threads
- Mar: POSIX semaphores
- Apr: POSIX message queue
- May: Add Posix Style File System API support

2019

- Jul: Civetweb HTTP sample
- Jul: Docs about POSIX implementation in Zephyr

2022

- Jun: Civetweb sample removed

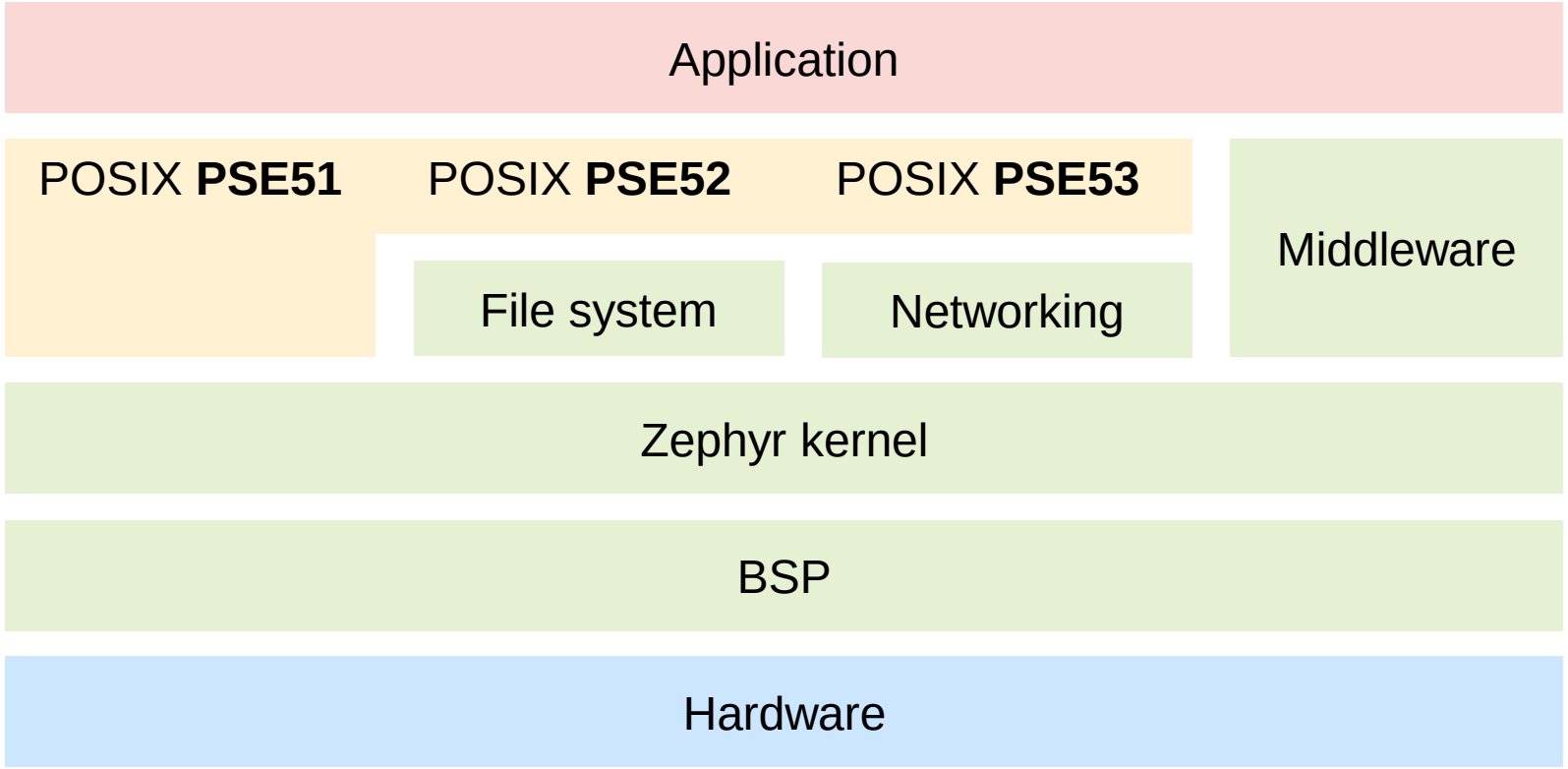
2023

- Feb: GSoC project “HTTP server for Zephyr”
- Jan - : Significant overhaul of POSIX support by Chris Friedt

Maintainers

- Youvedeep Singh
- Niranjhana Narayanan
- Ramakrishna Pallala
- Paul Sokolovsky
- Chris Friedt

POSIX in Zephyr RTOS



https://docs.zephyrproject.org/latest/services/portability/posix.html

- POSIX_C_LANG_SUPPORT
  - abs(), calloc(), free() etc.
- POSIX_THREADS_BASE
  - pthread_attr_*()
  - pthread_cancel()
  - pthread_create()
  - pthread_detatch()
  - pthread_cond_*()
  - pthread_mutex_*()
- POSIX_SIGNALS
  - abort()
- POSIX_DEVICE_IO
  - fprintf(), fputc(), fputs(), fwrite()
  - open(), perror(), printf(), putc(), puts(), read()
  - vfprintf(), vprintf()

Limitations and differences from traditional POSIX

- It’s yet another abstraction layer
- Automatic pthread stack and object allocation
- Some headers need to be moved or harmonized with native Zephyr headers
- More C library functions need to be implemented or updated their current behavior
- sysconf() utility is missing

Real examples of Zephyr RTOS with POSIX support

- Built-in sample: Civet Web
- 3rd party application: EEMBC CoreMark Pro

HTTP web-server example: Civet Web

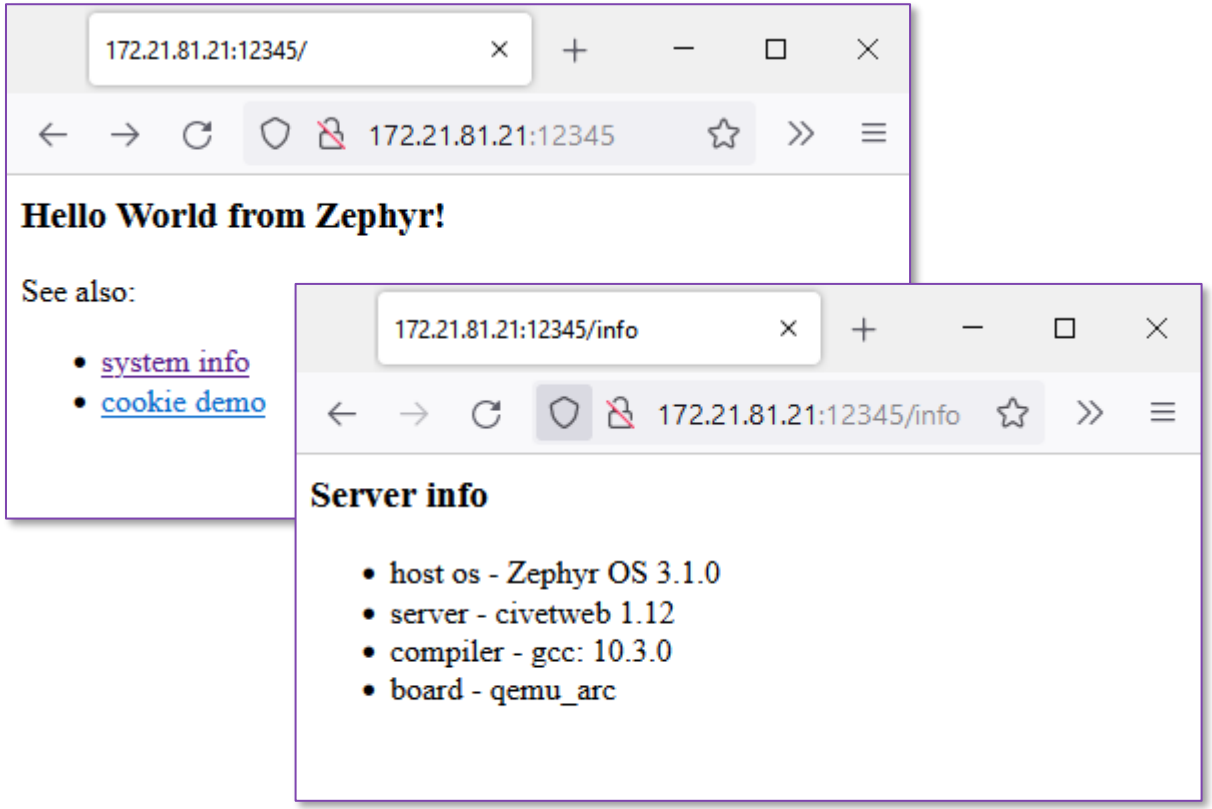

```
$ west build -b qemu_arc_hs
          samples/net/civetweb/http_server



$ west build -t run

*** Booting Zephyr OS build zephyr-v3.1.0 ***
<inf> net_config: Initializing network
<inf> net_config: IPv4 address: 192.0.2.1

<dbg> mg_cry_internal_impl: log_access:
192.0.2.2 - "GET /?null HTTP/1.0" 200
```



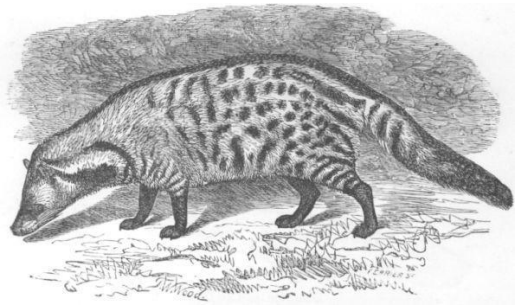
Civet Web: Complex application on top of Zephyr RTOS

Services and API's available in Zephyr RTOS make it possible

Written mostly on C and uses:

- POSIX API
  - POSIX Threads
  - POSIX Message Queues
  - BSD Sockets
- DNS client
- IPv4
- TCP
- Ethernet device drivers

- Independent project developed on GitHub: https://github.com/civetweb/civetweb
- Very minimal changes for Zephyr support, see commit 67cdc
- Removed from Zephyr RTOS in v3.2.0 due to missing maintainer
- Replacement to be worked on during GSoC 2023



EEMBC CoreMark Pro

Primer for easy porting of POSIX compliant application to Zephyr RTOS

Input data

- Original sources: https://github.com/eembc/coremark-pro
- Uses EEMBC's Multi-Instance Test Harness (MITH)
- MITH abstraction layer is configured to work with the POSIX pthread

Port to Zephyr

- Integrate with Zephyr build system*
  - `CMakeLists.txt`
  - `prj.conf`
- Pre-populate **argc** & ***argv[]**

Requires porting!

* Zephyr freestanding application: https://docs.zephyrproject.org/latest/develop/application/index.html#zephyr-freestanding-app

CoreMark Pro: Integrate with Zephyr build system

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.20.0)

find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
project(core)

# MITH headers
include_directories(mith/include)
include_directories(mith/al/include)

# MITH sorces
FILE(GLOB mith_sources mith/al/src/*.c mith/src/*.c)
target_sources(app PRIVATE ${mith_sources})
```

```
# Input arguments
target_sources(app PRIVATE main-args.c)

# CoreMark Pro sources
FILE(GLOB app_sources benchmarks/core/*.c)
target_sources(app PRIVATE ${app_sources})
target_sources(app PRIVATE workloads/core/core.c)

set_ifndef(W_WORKERS_NUM 4)
target_compile_options(app PRIVATE "-D W_WORKERS_NUM=${W_WORKERS_NU

set_ifndef(C_CONTEXT_NUM 4)
target_compile_options(app PRIVATE "-D C_CONTEXT_NUM=${C_CONTEXT_NU

set_ifndef(I_ITERATION_NUM 1)
target_compile_options(app PRIVATE "-D I_ITERATION_NUM=${I_ITERATIO
```

CoreMark Pro: Integrate with Zephyr build system

`prj.conf`

```
# Enable POSIX API
CONFIG_POSIX_API=y
CONFIG_PTHREAD_IPC=y
CONFIG_POSIX_CLOCK=y

# Disable getopt() in Zephyr, as CoreMark uses its own version
# CONFIG_GETOPT is not set

# Use full-featured libc
CONFIG_NEWLIB_LIBC=y

# Optimize for speed (default is for size)
CONFIG_SPEED_OPTIMIZATIONS=y
```

CoreMark Pro: Pre-populate `argc` & `*argv[]`

Embedded application doesn't accept command line arguments, work it around

```
// main-args.c

#define _g_xstr(a) _g_str(a)
#define _g_str(a) #a

int argc = 5;
char *argv[] = {
          "core.exe",
          "-v0",
          "-w" _g_xstr(W_WORKERS_NUM),
          "-c" _g_xstr(C_CONTEXT_NUM),
          "-i" _g_xstr(I_ITERATION_NUM)
};
```

```
// workloads/core/core.c

#ifdef __ZEPHYR__
extern int argc;
extern char *argv[];

void main(void)
#else
int main(int argc, char *argv[])
#endif
{
          char name[MITH_MAX_NAME];
          char dataname_buf[MITH_MAX_NAME];

...
```

CoreMark Pro: execute “core” benchmark

```
west build -b qemu_arc_hs core -t run
```

```
-- west build: running target run

*** Booting Zephyr OS build zephyr-v3.4.0 ***
-  Info: Starting Run...
-- Workload:core=490760323
-- core:time(ns)=1137920
-- core:contexts=4
-- core:workers=4
-- core:iterations=4
-- core:time(secs)= 1137.92
-- core:secs/workload=  284.48
-- core:workloads/sec=0.00351519
-- Done:core=490760323
```

Conclusion

- Ongoing work since 2017 done by multiple people
- POSIX support development in Zephyr is driven by real application needs, i.e. enough POSIX features to get application X working on Zephyr
- A lot could be done with POSIX compatibility layer in Zephyr already
- There are pros & cons of using POSIX

Thank You

Backup slides

Additional resources

- "POSIX Roadmap for LTSv3", presentation by Chris Friedt on June 29th, 14:10 CET
- Standards documentation
  - 1003.1-2017 – IEEE Standard for Information Technology – Portable Operating System Interface (POSIX(TM)) Base Specifications, Issue 7
  - 1003.13-2003 – IEEE Standard for Information Technology – Standardized Application Environment Profile (AEP) – POSIX(TM) Realtime and Embedded Application Support
- Interesting material
  - Early experience with POSIX 1003.4 and POSIX 1003.4 A by B.O. Gallmeister, Lynx Real-Time Systems
  - "Isn't it About Time for a Standard RTOS API?" article by Bill Lamie (father of ThreadX & PX5)
  - The Open Group POSIX Profile 52 Test Suite – VSPSE52-2003
  - "Real-time POSIX: An overview" by Michael González Harbour, Departamento de Electrónica, Universidad de Cantabria
  - GitHub issues on the matter:
    - [RFC] Proposed development plan for Zephyr's POSIX subsystem
    - RFC: POSIX Roadmap for LTSv3

List of POSIX Base Standards

POSIX.1	System Interface (basic reference standard)
POSIX.2	Shell and Utilities
POSIX.3	Methods for Testing Conformance to POSIX
POSIX.4	Real-time Extensions
POSIX.4a	Threads Extensions
POSIX.4b	Additional Real-time Extensions
POSIX.6	Security Extensions
POSIX.7	System Administration
POSIX.8	Transparent File Access
POSIX.12	Protocol Independent Network Interfaces
POSIX.15	Batch Queuing Extensions
POSIX.17	Directory Services

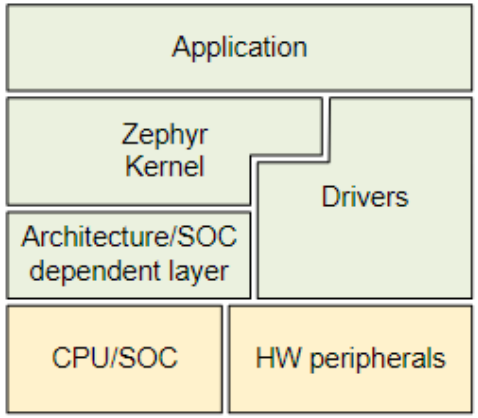
List of POSIX Application Environment Standards

POSIX.0	Guide to POSIX Open System Environment
POSIX.10	Supercomputing Application Environment Profile
POSIX.11	Transaction Processing Application Environment Profile
POSIX.13	Real-time Application Environment Profiles
POSIX.14	Multiprocessing Application Environment Profile
POSIX.18	POSIX Platform Application Environment Profile

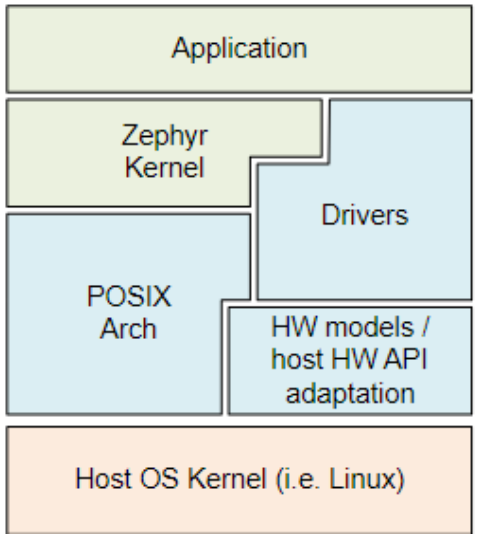
Native POSIX execution (native_posix)

Convenient tool for Zephyr RTOS exploration with great visibility of host tools

Normal
Zephyr layering



"Native_posix"
Zephyr layering



- Designed and tested to run in Linux
- Produces native binary for the host
- Each Zephyr thread is mapped to one POSIX Thread
- Transparent to application
- Allow functional debugging, instrumentation and analysis of the code with native tooling

https://docs.zephyrproject.org/latest/boards/posix/doc/arch_soc.html