An Incubator Project in the Apache Software Foundation

http://mynewt.apache.org/
Apache Mynewt

Open Source OS for Constrained IoT
  - MCU / Hardware independent
  - ARM Cortex-M*, AVR, MIPS, more...
  - RISC-V

http://mynewt.apache.org/
Problem and Context

First release of a successful IoT product...

...now make that repeatable, please.
IoT is being driven by the System on Chip (SoC)

**Apache Mynewt** addresses everything needed for SoC development

Characteristics

- **CPU:** 48MHz-300MHz, Cortex-M
- **Radios:** BLE, Wi-Fi, 802.15.4(g)
- **Flash/RAM:** 512KB/64KB (today), 1MB/256KB (this year)
- **Size:** 3-12mm

Benefits

- Inexpensive
- Low Power
- Easy to manufacture

Apache Mynewt is “Linux” for devices that cannot run Linux
Runtime with Mynewt Modernizes Development for SoCs

Without Runtime

Choose Compiler
Build Bootloader
Develop Logging
Buy Net Stack
Write Gateway
Buy RTOS
Write Flash Drivers
Support Chip
MFRG Support
Develop SW Update

With Runtime

Use Apache Mynewt
Connect to Runtime Management Service
Provision AWS environment
Develop Application

Development: Faster, Consistent, Repeatable. Escape HW lock-in.
Another View: Initial Bluetooth Low Energy System

Cloud
- Device Provisioning
- Software Upgrade
- Failure Analysis

Gateway
- NewtMgr
  - Discovery
  - Registration
  - Provisioning
  - Stats & Logging
  - Configuration Management
  - Core Management
  - Image Upgrade

OS
- NewtMgr
- Bluetooth Low Energy (BLE)
- Stats + Logging
- Kernel / RTOS
- Secure Bootloader, FFS
- Manufacturing Support

Only end-to-end platform built atop FOSS; usage-based cloud
A Community Effort

Why the ASF?

- Liberal Apache 2.0 license
- Individuals, not Corporations
- Meritocracy
- Strong licensing and IP policies
- Long history of working with large organizations: IBM, Oracle, Pivotal/EMC
- Many years experience managing large, complex projects: Apache Web Server, Hadoop, Cassandra, Kafka, Subversion, etc.

Community-driven Open Source: Best Way to Maintain Healthy User Ecosystem
Apache Mynewt Users

(today)
- Bluetooth connected products
  - Medical: everything
  - Consumer/Enterprise: locks, lights
  - Industrial
- Makers
  - Home
  - Hardware Labs
  - Clothing

(tomorrow)
- Bluetooth Low Energy 4.2 → Bluetooth 5
- Industrial Wireless Sensor Networks
- Wi-Fi Products
- Who Knows?

• Power Optimization
• Mesh networking
• Security
• Sensor Algorithms
• Control Systems

Scale Makes Problems Interesting
Project Statistics

- 280,000 lines of code (March 2016)
- Initial support for Simulator, Nordic NRF51/52, STM32F3/4 and Arduino
  - Announced: Arduino Primo, Arduino Otto
- Active contributors, We Want You!
Build and Package Management

Goals

- Maintain and re-use packages across multiple products
- Manage debug and production build setups
- Make it easy to find, install 3rd party libraries
- Efficiency: use only what you need

Description

- Everything is a package. Each package describes its dependencies to the rest of the world
- A collection of packages is called an application
- There are a few special packages: BSP and Project. Project contains main() and BSP defines linker script and hardware layout
- Targets are used to combine projects and BSP
- Packages can be distributed, upgraded and installed remotely

Compose across multiple repos
Project Structure (coming soon)
“core” broken into appropriate sub-projects

- ASF governance structure (PMCs) corresponds with sub-package structure
- ASF repositories clean, Apache 2.0 license
- Users can assemble projects sourced from multiple repos

Composability across multiple repos provides flexibility
BSP and HAL

Goals

Provide quality drivers for major MCU platforms
Design for cross-platform: well-defined APIs for HAL, BSP and drivers
Make it easy to add board specific definitions

Description

- BSP definition is provided in <app>/hw/bsp
- HAL definitions are in hw/hal and contain uniform, cross-platform APIs
- MCU definitions are in hw/mcu and provide implementations for various MCUs
  - Hierarchy allows code-reuse within MCU families
- BSPs for common dev kits are available as packages (e.g., Nordic nRF51/2 DK)
- BSPs depend on MCUs
- BSP + MCU provides implementation for HAL APIs

Chip Vendors: We Want You!
Kernel

- Pre-emptive, multi-tasking RTOS
  - Strict, priority-based scheduling
  - Up to 253 different priority levels

- Tickless kernel

- Power management

- Resource utilization tracking

- Built-in tasks:
  - Idle
  - Sanity

```c
#include <os/os.h>
#include <assert.h>

#define TASK1_PRIO (1)
#define TASK1_STACK_SIZE OS_STACK_ALIGN(1024)

struct os_task task1;
static volatile int g_task1_loops;

void
task1_handler(void *arg)
{
    while (1) {
        ++g_task1_loops;
        /* Wait one second */
        os_time_delay(1000);
    }
}

int
main(int argc, char **argv)
{
    int rc;
    os_init();

    os_task_init(&task1, "task1", task1_handler, NULL,
                 TASK1_PRIO, OS_WAIT_FOREVER, stack1, TASK1_STACK_SIZE);
    os_start();
    assert(0);
    return (0);
}
```
Energy Efficient Event Model

- Event Queues provide a mechanism for “mostly sleeping” asynchronous tasks
  - Wake-up on:
    - Message from another task
    - Timer
    - I/O state change
    - Incoming packet
    - Watchdog
  - Perform operations:
    - Send an alert
    - Respond to a request
    - Schedule a wakeup
  - Go back to sleep

```c
struct os_eventq task1_evq;
struct os_eventq task2_evq;

#define OS_EVENT_T_PING (OS_EVENT_PERUSER)
#define OS_EVENT_T_PONG (OS_EVENT_PERUSER + 1)

void
task1_handler(void *arg)
{
    struct os_event *ev;
    struct os_event ping_ev;

    ping_ev.ev_type = OS_EVENT_T_PING;
    ping_ev.ev_arg = NULL;

    os_eventq_put(&task2_evq, &ping_ev);

    while (1) {
        ev = os_eventq_get(&task1_evq);
        assert(ev->ev_type == OS_EVENT_T_PONG);
        os_eventq_put(&task2_evq, &ping_ev);
        ++g_task1_loops;

        /* Wait one second */
        os_time_delay(1000);
    }
}

void
task2_handler(void *arg)
{
    struct os_event *ev;
    struct os_event pong_ev;

    pong_ev.ev_type = OS_EVENT_T_PONG;
    pong_ev.ev_arg = NULL;

    while (1) {
        ev = os_eventq_get(&task2_evq);
        assert(ev->ev_type == OS_EVENT_T_PING);
        os_eventq_put(&task1_evq, &pong_ev);
        ++g_task2_loops;
    }
}
```
• Bootloader can be located in ROM or Flash
  • Options for internal and external flashes
• Performs integrity check and swaps images
• Images contain SHA-256 hash and RSA signature
• NFFS optional
  • Provides a log-structured flash filesystem designed for small flashes
• CB (Circular Buffer) optional
  • Provides implementation of flash circular buffer
# System Security

<table>
<thead>
<tr>
<th>What We’re Protecting</th>
<th>How We Protect It</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bootloader</td>
<td>• First stage bootloader can operate from ROM and verify signature of second stage bootloader</td>
</tr>
</tbody>
</table>
| Images                | • All images have SHA-256 of image contents  
                         • Images support signing with ECC or RSA 2048 bit signatures  
                         • Second stage bootloader can verify image signature |
| Network Interfaces    | • Full support for BLE 4.2 security at 1.0 release, including link-layer and app-layer |
Simulator and Test Framework

- In addition to various MCU ports: OS, HAL, FS and the majority of packages can run on Mac, Linux

- Develop your code on the host and then port to the real hardware

- Unit test framework is incorporated to all of the packages: ability to run unit tests on simulated environment and real hardware

- OS and libraries are fully regression tested to ensure API compatibility between releases

```c
/* Test some error cases */
TEST_ASSERT(os_mutex_init(NULL) == OS_INVALID_PARM);
TEST_ASSERT(os_mutex_delete(NULL) == OS_INVALID_PARM);
TEST_ASSERT(os_mutex_release(NULL) == OS_INVALID_PARM);
TEST_ASSERT(os_mutex_pend(NULL, 0) == OS_INVALID_PARM);

/* Get the mutex */
err = os_mutex_pend(mu, 0);
TEST_ASSERT(err == 0,
    "Did not get free mutex immediately (err=\d)", err);

/* Check mutex internals */
TEST_ASSERT(mu->mu_owner == t && mu->mu_level == 1 &&
    mu->mu_prio == t->t_prio && SLIST_EMPT(Y(mu->mu_head),
    "Mutex internals not correct after getting mutex\n" "Mutex: owner=\p prio=\u level=\u head=\p \n" "Task: task=\p prio=\u",
    mu->mu_owner, mu->mu_prio, mu->mu_level, 
    SLIST_FIRST(&mu->mu_head),
    t, t->t_prio);

/* Get the mutex again; should be level 2 */
err = os_mutex_pend(mu, 0);
TEST_ASSERT(err == 0, "Did not get my mutex immediately (err=\d)", err);
```
Apache Mynewt Roadmap

Highlights

v0.8: First release, BLE 4.2, FFS, Kernel, Console, Shell, Secure Boot
v0.9: Expanded HW support and HAL
v0.10: Wi-Fi & IP support
v0.11: Full Bluetooth Support / Qualification
v1.0 (GA) API compatibility, Full Regression Testing
THANK YOU

http://mynewt.apache.org/

Mailing List: dev@mynewt.incubator.apache.org

IRC: #mynewt on freenode
### Mynewt Components/Composition

<table>
<thead>
<tr>
<th>Component</th>
<th>Min RAM</th>
<th>Min ROM/On-chip Flash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core OS kernel</td>
<td>&lt;1kB</td>
<td>&lt;6kB</td>
</tr>
<tr>
<td>Blinky application (incl. GPIO, HAL, console, shell)</td>
<td>17kB</td>
<td>23kB</td>
</tr>
<tr>
<td>Bootloader (incl. Newtron Flash File System)</td>
<td>4.5kB</td>
<td>25kB</td>
</tr>
<tr>
<td>NimBLE stack (incl. both peripheral and central roles, legacy pairing)</td>
<td>4.5kB</td>
<td>69kB</td>
</tr>
<tr>
<td>BLE example application “bleprph” (incl. OS, radio, NimBLE)</td>
<td>15kB</td>
<td>99kB</td>
</tr>
</tbody>
</table>

**Recommended RAM/Flash**

32-64kB/256-512kB
# RAM Requirements for BLE Applications

<table>
<thead>
<tr>
<th>BLE Component</th>
<th>BLE Configuration Element</th>
<th>Default (kB)</th>
<th>Element Size (kB)</th>
<th>Default Size (kB)</th>
<th>Size for x # of connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host (.bss+.data)</td>
<td></td>
<td></td>
<td>3828</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HCI buffer</td>
<td>3</td>
<td>64</td>
<td>204</td>
<td>Independent</td>
</tr>
<tr>
<td></td>
<td>Max # of concurrent connections</td>
<td>1</td>
<td>80</td>
<td>80</td>
<td>x*80</td>
</tr>
<tr>
<td></td>
<td>Max # of services</td>
<td>5</td>
<td>8</td>
<td>40</td>
<td>Independent</td>
</tr>
<tr>
<td></td>
<td>Max # of config descriptors (peripheral)</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>(x+1)*3</td>
</tr>
<tr>
<td></td>
<td>Max # of concurrent GATT procedures</td>
<td>2</td>
<td>40</td>
<td>80</td>
<td>Independent</td>
</tr>
<tr>
<td></td>
<td>Max. # of total ATT attributes</td>
<td>36</td>
<td>32</td>
<td>1152</td>
<td>Independent</td>
</tr>
<tr>
<td></td>
<td>max_prep_entries (for partial writes)</td>
<td>6</td>
<td>12</td>
<td>72</td>
<td>Independent</td>
</tr>
<tr>
<td></td>
<td>Max # of L2CAP channels (3 per connection)</td>
<td>3</td>
<td>28</td>
<td>84</td>
<td>x*3</td>
</tr>
<tr>
<td></td>
<td>Max # concurrent L2CAP signalling procedures</td>
<td>2</td>
<td>20</td>
<td>40</td>
<td>Independent</td>
</tr>
<tr>
<td></td>
<td>Max # concurrent Security Manager procedures</td>
<td>1</td>
<td>360</td>
<td>360</td>
<td>Independent</td>
</tr>
<tr>
<td>Host (runtime RAM reqs)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Controller (BSS + data)</td>
<td>Max # of concurrent connections</td>
<td>1</td>
<td>416</td>
<td>416</td>
<td>x*416</td>
</tr>
<tr>
<td></td>
<td># of duplicate scan advertisers</td>
<td>8</td>
<td>8</td>
<td>64</td>
<td>Independent</td>
</tr>
<tr>
<td></td>
<td># of scan response advertisers</td>
<td>8</td>
<td>8</td>
<td>64</td>
<td>Independent</td>
</tr>
<tr>
<td></td>
<td>Whitelist size</td>
<td>8</td>
<td>8</td>
<td>64</td>
<td>Independent</td>
</tr>
<tr>
<td></td>
<td>Resolvable private address list</td>
<td>4</td>
<td>40</td>
<td>160</td>
<td>Independent</td>
</tr>
<tr>
<td></td>
<td>RNG buffer size</td>
<td>32</td>
<td>1</td>
<td>32</td>
<td>Independent</td>
</tr>
</tbody>
</table>

**Runtime RAM requirement for NimBLE with defaults (including security): 6744 kB**

Dynamic Power Manager + Tickless OS

(Goal: to achieve the most power-efficient state)

Timeout-based system power policies
(e.g., “enact deepest sleep state whenever possible for application A”)

MCU power states, peripheral power consumption, constraints on state transitions
(e.g., “Chip C supports 3 sleep states each with transition delay $T_d$ and energy overhead $E_o$)

Enable/Disable MCU peripherals
(e.g., disable clock and power domains for serial port since constraints have been released)

Runtime: Confidential and Proprietary (2016)