Lua in MPU-Backed Userspace

Louis Goessling // Entropic Engineering

Who am I?

- Finishing up a B.S. at the University of Minnesota
- Embedded Systems Engineer at Entropic Engineering
- Previous work with MELT compiler tech group at the U of M

louis goessling
louis@goessling.com
louis.goessling.com

Challenge

- Smart Devices
  - IoT, but also crockpots, lighting equipment, simon says, etc.
  - 64MHz Cortex M4-class (nrf52840)
- Deep and flexible end-user configurability
  - Control hardware outputs, make complex decisions based on hardware inputs, be easily configurable by end-users
  - Paper over different SKUs
- Facilitate a end-user marketplace of configurations
  - Users can re/configure device at runtime

Challenge: Constraints

- Need to maintain invariants for safety, efficiency, warranty, etc.

- Memory-safety: must not enter undefined state, musn’t corrupt firmware
- Time-safety: must keep firm realtime constraints
- Hardware-safety: must not drive I/O devices outside of safe + reliable ranges

Approaches: Custom configuration scheme

- + Fun project!
- + Allows designed-in invariant guarantees
  - Can design out unbounded runtime, bad accesses, etc.
- - “Any sufficiently advanced configuration is indistinguishable from code”
  - State machines, flags, backwards-compatibility constraints
- - Maintenance nightmare
- - No existing end-user familiarity

Approaches: MicroPython

- + Tons of people are familiar with Python
- + Existing project with supporting community
- ~ First party Zephyr port exists as ‘work-in-progress’
- - Large codebase to trust (+`CONFIG_NEWLIB_LIBC`)
- - Developed to be in control of hardware, “backwards” of what we want
- - Heavyweight (recommended devices are 96MHz)
- - Python is a complex language; uPython isn’t “really” python

Approaches: Lua

- + Not as popular as Python, but still popular
- + Existing project with supporting community
- + Tiny language! So we can embed the real thing
- + Easy to embed (designed for embedding in C applications)
- - Smaller but still large codebase to trust

Lua in MPU-backed userspace + preemption

- + Not as popular as Python, but still popular
- + Existing project with supporting community
- + Tiny language! So we can embed the real thing
- + Easy to embed (designed for embedding in C applications)
- **+ Complex Lua code sandboxed from hardware resources and critical memory**
- **+ Timeslicing to sandbox from computational resources**

```lua
function periodic(dt_ms, inputs)
    local scaler = inputs[1]
    local period = compute_period(scaler)
    local fraction = compute_fraction(dt_ms, period)

    return fraction * scaler * 0xFFFF
end
```

Building Lua for Zephyr

- Build the Lua interpreter with CMake instead of raw Make
- Integrate Lua's `luaconf.h` settings into KConfig
- Carry minimal patches vs. upstream Lua
  - On the order of 100 lines of diff to core lua interpreter
- Support Lua's math library and string manipulation *without* `CONFIG_NEWLIB_LIBC` or `CONFIG_MINIMAL_LIBC_MALLOC`
  - musl (MIT) math routines + GCC's `__builtin_setjmp` + custom allocator on top of `sys_heap`

- Prior art: Civetweb
  - Embedded HTTP server with lua scripting support available for Zephyr
  - Caveat: Makefile build, no sandboxing

Building Lua for Userspace

- Lua already keeps all it's interpreter state in a `lua_State*` with a `void* ud`
- ... and supports a user-defined allocator - yay!

- Ensure memory allocation and libc implementations use userspace memory
- Pass data and control back-and-forth between 'kernel' and userspace
- Gotcha: it is **not safe** to load arbitrary Lua bytecode!
  - Reduce attack-surface by compiling source Lua in userspace also

```
#define IN_USERLAND_BSS         K_APP_BMEM( USERLAND_PART )
#define IN_USERLAND_INITIALIZED K_APP_DMEM( USERLAND_PART )
```

```
IN_USERLAND_BSS uint8_t user_task_memory[TASK_COUNT][USER_TASK_HEAP_SIZE];
```

```
k_mem_domain_init( &userland_domain, 0, NULL );
k_mem_domain_add_partition( &userland_domain, &USERLAND_PART );
```

```
k_mem_domain_add_thread( &userland_domain, task->tid );
```

Using Lua in Userspace

- `CONFIG_USERSPACE`, `CONFIG_FPU_SHARING`
- Pass data and control back-and-forth between ‘kernel’ and userspace
  - Considered: Zephyr’s syscall infrastructure, semaphores, Lua’s lightweight userdata
  - Selected: `CONFIG_POLL`, `lua_pushcfunction`
  - ```
    IN_USERLAND_BSS bool button_state_u;
    ```

- Recover from crashes via `k_sys_fatal_error_handler`
  - ```
    if (k_current_get()->base.user_options & K_USER) { recover ... }
    ```
  - Finding kernel bugs means you’re doing something fun!
    https://github.com/zephyrproject-rtos/zephyr/issues/42496

Using Lua for cool stuff

- It’s fast - 1000 Hz+!
- Multiple Lua tasks!
- Noncontiguous Lua heaps!
- Lua script persistence in flash!
- Over-the-air script upload!
- Complex-typed script I/O!
- Off-device emulation and debugging!
  - `native_posix` target + non-Zephyr Lua environments

Demo!

Find artifacts online at
https://gitlab.com/prv-labs/zephyr-lua-demo