Zephyr: Timers

Medical Electrical Equipment (BME590L)

Mark L. Palmeri, M.D., Ph.D.

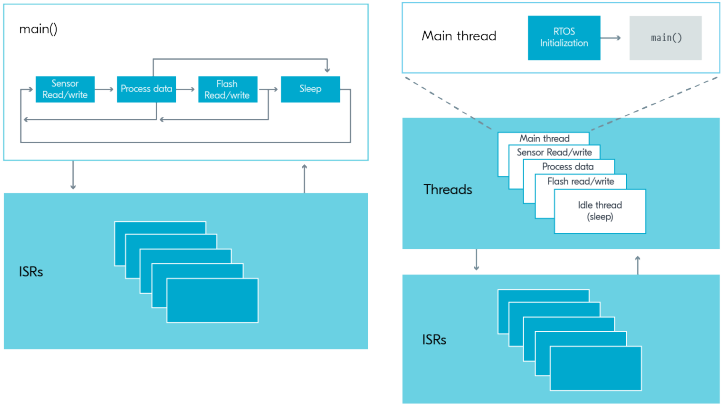
March 01, 2023

Duke | BME

What are the challenges with sleep statements?

- Overall timing of the main loop hard to estimate with multiple "tasks".
- Adding / removing features can disrupt all of the timing.
- ISRs can make things really difficult.

Threads

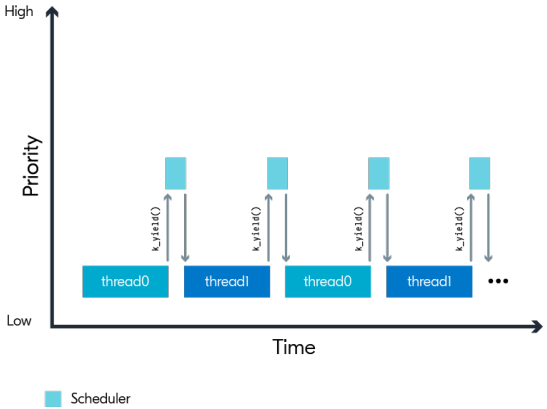


Duke | BME

System vs. User Threads / Work Queues

- Zephyr's **kernel** starts a system thread and a user-space thread by default.
- The user can add tasks to the system thread and/or create user threads.
- System & user work queues also exist (lower-priority FIFO).

Thread Priority



Duke | BME

Timers Threads

In addition to ISR priority and thread priority yielding, the system also has **timers** for high temporal accuracy tasks (one-time or repeated).

- Timers are managed by the kernel (system).
- Timers are always available (i.e., no need to include libraries or add project configuration options)
- Like ISRs, timer events should not consume significant resources / take much time to execute.
- The kernel gives timing events relatively high priority.

Before main(): K_TIMER_DEFINE

K_TIMER_DEFINE is a macro that sets up the timer for you.

```
K_TIMER_DEFINE(name_of_timer, func_to_exec_on_timer_start,
↪  func_to_exec_on_timer_stop);
```

If the timer will run indefinitely, or doesn't need a stopping procedure, then func_to_exec_on_timer_stop isn't needed and can be set to NULL.

```
K_TIMER_DEFINE(name_of_timer, func_to_exec_on_timer_start, NULL);
```

Duke | BME

Declare timer start/stop functions

```
void func_to_exec_on_timer_start(struct k_timer *name_of_timer);
void func_to_exec_on_timer_stop(struct k_timer *name_of_timer);
```

The `k_timer` struct was defined by the `K_TIMER_DEFINE` macro.

Define Timer Functions

```
void func_to_exec_on_timer_start(struct k_timer *name_of_timer)
{
    gpio_pin_toggle_dt(&somepin);
    LOG_DBG("Did something");
}
```

Within main(): `k_timer_start()`

```
k_timer_start(&name_of_timer, K_MSEC(DURATION_OF_FIRST_EVENT),
↪  K_MSEC(DURATION_OF_REPEATED_EVENTS));
```

- `K_MSEC` is a macro that converts a time, specified in ms (`DURATION_OF_FIRST_EVENT`) to whatever time unit the function requires.
- Other useful macros include `K_SECONDS`, `K_MINUTES`, `K_HOURS`, etc.[1]
- If you just want to do something once (one-shot behavior):

```
k_timer_start(&name_of_timer, K_MSEC(DURATION_OF_FIRST_EVENT), 0));
```

[1] https://docs.zephyrproject.org/latest/kernel/services/timing/clocks.html

If you need to stop the timer...

```
k_timer_stop(&name_of_timer);
```

This will call `func_to_exec_on_timer_stop()` that you associated with the timer with the `K_TIMER_DEFINE` macro.

Next lab exercise...

- Refactor your code from the blinking light lab to:
  - Eliminate the use of sleep statements.
  - Use timers for all LED illumination events.
- Be carefuly using GPIO "toggle".
  - Okay for the heartbeat
  - Not okay for "event" LEDS as you might lose the explicit state of each LED.
  - Instead, explicitly "set" all 3 LEDs for each event.
    - The mutually exclusive state of the LEDs means that one function should set all 3 states explicitly at once.
    - Capture the state of the LEDs in a struct.

Duke | BME