Table of Contents

1. Zephyr threads communication and synchronization 2

1. Zephyr threads communication and synchronization

In this part, we will show methods of exchanging data between threads. In the example from the previous part, threads were defined using K_THREAD_DEFINE()macro.

```
K_THREAD_DEFINE(thread0_id, STACKSIZE, thread0, NULL, NULL, NULL,
        THREAD0_PRIORITY, 0, 0);
```

The arguments of the macro are:

- the name of the thread
- size of the stack that is created for individual use of the thread
- address of the function of the thread
- three initial arguments of the thread which can be passed at the start of the thread (here all NULL)
- constant that specifies the priority of the thread
- thread options (here 0)
- delay to start the thread (0 meaning no delay)

As we can see every thread has its own stack, which means that if there is a need to exchange data between threads we have to use some shared variable visible for all threads interested in using it. In multitasking environments, especially running on a multi-core processor, there is a need to synchronise the access to shared resources. Zephyr offers some mechanisms for inter-thread data exchange and synchronisation. In the example available in the Zephyr framework the FIFO queue is used to pass data from two threads (blink0, blink1) which are used to blink two LEDs to the uart_out thread which displays received data in the terminal.

The FIFO queue can be defined with the macro K_FIFO_DEFINE. In our example, it defines the FIFO queue and gives it the name “printk_fifo”.

```
K_FIFO_DEFINE(printk_fifo);
```

The Blink0 thread as well as Blink1 calls the same function blink() passing three parameters:

1. address of the structure that represents LED (&led0 or &led1 respectively)
2. number of milliseconds the function goes to sleep between iterations
3. id of LED (0 or 1)

Notice that blink() is called only once from thread Blink0 and once from thread Blink1. After the initialisation of LED, it jumps into infinite while(1) loop. Every iteration toggles the LED using the least significant bit of the cnt variable

```
gpio_pin_set(spec->port, spec->pin, cnt % 2);
```

and inserts the content of the object tx_data into the printk_fifo using the dynamically allocated memory region of the size equal to printk_data_t size.

```
k_fifo_put(&printk_fifo, mem_ptr);
```

The uart_out thread gets the element from the thread with the function k_fifo_get

```
struct printk_data_t *rx_data = k_fifo_get(&printk_fifo, K_FOREVER);
```

After successful reading, prints the LED number and counter value (cnt) to the console, and releases the memory allocated by blink.

The full code looks as follows. If you face problems with compilation add the following line to the prj.conf file

```
CONFIG_HEAP_MEM_POOL_SIZE=1024
```

```
/*
 * Copyright (c) 2017 Linaro Limited
 *
 * SPDX-License-Identifier: Apache-2.0
 */

#include <zephyr.h>
#include <device.h>
#include <drivers/gpio.h>
#include <sys/printk.h>
#include <sys/__assert.h>
#include <string.h>

/* size of stack area used by each thread */
#define STACKSIZE 1024

/* scheduling priority used by each thread */
#define PRIORITY 7

#define LED0_NODE DT_ALIAS(led0)
#define LED1_NODE DT_ALIAS(led1)

#if !DT_NODE_HAS_STATUS(LED0_NODE, okay)
#error "Unsupported board: led0 devicetree alias is not defined"
#endif

#if !DT_NODE_HAS_STATUS(LED1_NODE, okay)
#error "Unsupported board: led1 devicetree alias is not defined"
#endif

struct printk_data_t {
    void *fifo_reserved; /* 1st word reserved for use by fifo */
    uint32_t led;
    uint32_t cnt;
};

K_FIFO_DEFINE(printk_fifo);
```

```
struct led {
    struct gpio_dt_spec spec;
    const char *gpio_pin_name;
};

static const struct led led0 = {
    .spec = GPIO_DT_SPEC_GET_OR(LED0_NODE, gpios, {0}),
    .gpio_pin_name = DT_PROP_OR(LED0_NODE, label, ""),
};

static const struct led led1 = {
    .spec = GPIO_DT_SPEC_GET_OR(LED1_NODE, gpios, {0}),
    .gpio_pin_name = DT_PROP_OR(LED1_NODE, label, ""),
};

void blink(const struct led *led, uint32_t sleep_ms, uint32_t id)
{
    const struct gpio_dt_spec *spec = &led->spec;
    int cnt = 0;
    int ret;

    if (!device_is_ready(spec->port)) {
        printk("Error: %s device is not ready\n", spec->port->name);
        return;
    }

    ret = gpio_pin_configure_dt(spec, GPIO_OUTPUT);
    if (ret != 0) {
        printk("Error %d: failed to configure pin %d (LED '%s')\n",
            ret, spec->pin, led->gpio_pin_name);
        return;
    }

    while (1) {
        gpio_pin_set(spec->port, spec->pin, cnt % 2);

        struct printk_data_t tx_data = { .led = id, .cnt = cnt };

        size_t size = sizeof(struct printk_data_t);
        char *mem_ptr = k_malloc(size);
        __ASSERT_NO_MSG(mem_ptr != 0);

        memcpy(mem_ptr, &tx_data, size);

        k_fifo_put(&printk_fifo, mem_ptr);

        k_msleep(sleep_ms);
        cnt++;
    }
}
```

```
void blink0(void)
{
    blink(&led0, 100, 0);
}

void blink1(void)
{
    blink(&led1, 1000, 1);
}

void uart_out(void)
{
    while (1) {
        struct printk_data_t *rx_data = k_fifo_get(&printk_fifo,
                                K_FOREVER);
        printk("Toggled led%d; counter=%d\n",
               rx_data->led, rx_data->cnt);
        k_free(rx_data);
    }
}

K_THREAD_DEFINE(blink0_id, STACKSIZE, blink0, NULL, NULL, NULL,
        PRIORITY, 0, 0);
K_THREAD_DEFINE(blink1_id, STACKSIZE, blink1, NULL, NULL, NULL,
        PRIORITY, 0, 0);
K_THREAD_DEFINE(uart_out_id, STACKSIZE, uart_out, NULL, NULL, NULL,
        PRIORITY, 0, 0);
```

Note that there is no main() function in this code, everything happens in thread functions.

From:
https://dokuwiki.zmitac.aei.polsl.pl/ - **RAU6 Dokuwiki**

Permanent link:
https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:s2:aih:lab:nrf52threads2?rev=1701597596

Last update: **2023/12/03 10:59**