Table of Contents

1. Zephyr OS Threads 2
1.1. Thread switching methods 2
1.2. Thread switching example 2

1. Zephyr OS Threads

In this laboratory, we will learn thread management and switching in Zephyr OS. The thread is a sequential stream of instructions. Thread can be the process itself or more threads can form the process. In a computer system with a single-core processor currently executed thread occupies all processor resources. Multi-threading operation system run on such processors uses time-sharing of resources to execute many threads simultaneously. Of course simultaneously in this situation means that threads are executed one-by-one.

1.1. Thread switching methods

Switching between threads is made by the part of the operating system known as a scheduler. The scheduler decides which thread should be executed next. To do so it considers the priorities of the threads and history of execution. There are two general methods of switching between threads that should be executed in parallel:

- cooperative
- preemptive

Zephyr OS implements both methods of thread switching. In the cooperative method, the thread is executed until it gives back the control to OS. Thus thread can starve others while not releasing resources. This method is often used in embedded systems due to its low complexity and system overhead. Cooperative threads in Zephyr have a negative priority value (thus they have a relatively higher priority). In the preemptive method, the thread is automatically suspended by the operating system when it decides to do so. Usually, the thread is given some period and is interrupted at the end of this period if there exists another thread to be executed. This method is implemented in general-purpose operating systems like Windows or Linux. It is much safer but more complicated and resource-consuming. In Zephyr preemptive threads have a positive priority number (thus they have a relatively lower priority).

1.2. Thread switching example

A cooperative thread gives control to the operating system in a few situations:

- if it decides to terminate by returning from its main function or by calling the k_thread_abort().
- if it decides to yield by calling the function k_yield().
- if the thread is sleeping by calling some version of sleep function k_sleep(), k_msleep(), or k_usleep().
- if the thread has been suspended by calling k_thread_suspend().

While implementing the thread with the infinite loop it is recommended to call the k_yield() or some sleep function every pass of the loop.

A preemptive thread can be stopped by the operating system at any time if it finds the higher-priority thread ready to operate.

As the first example, we'll show the one available within the Nordic Developer Academy [1] because the sample provided in the Zephyr framework is a little more advanced and complex for the beginning. This example creates two threads writing to the console simple text identifying the thread. Notice that in the original program, both threads have a positive value of the priority which makes them the preemptive threads.

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

#include <zephyr/kernel.h>
#include <zephyr/sys/printk.h>

/* STEP 2 - Define stack size and scheduling priority used by each thread */
#define STACKSIZE 1024

#define THREAD0_PRIORITY 7
#define THREAD1_PRIORITY 7

void thread0(void)
{
    while (1) {
          /* STEP 3 - Call printk() to display a simple string "Hello, I am
thread0" */
          printk("Hello, I am thread0\n");
          /* STEP 6 - Make the thread yield */
                  //k_yield();
          /* STEP 10 - Put the thread to sleep */
          k_msleep(5);
          /* Remember to comment out the line from STEP 6 */
    }
}

void thread1(void)
{
    while (1) {
          /* STEP 3 - Call printk() to display a simple string "Hello, I am
thread1" */
                  printk("Hello, I am thread1n");
          /* STEP 8 - Make the thread yield */
          //k_yield();
          /* STEP 10 - Put the thread to sleep */
          k_msleep(5);
          /* Remember to comment out the line from STEP 8 */
    }
}

/* STEP 4 - Define and initialize the two threads */
K_THREAD_DEFINE(thread0_id, STACKSIZE, thread0, NULL, NULL, NULL,
```

```
        THREAD0_PRIORITY, 0, 0);
K_THREAD_DEFINE(thread1_id, STACKSIZE, thread1, NULL, NULL, NULL,
        THREAD1_PRIORITY, 0, 0);
```

To go through the lab please follow the steps:

1. Copy-paste the code into some ready example in VSCode
2. If you experience a problem with the include paths delete “zephyr/” from paths of *.h files
3. Compile and execute the program, observe the behaviour of threads in the terminal
4. Make some experiments as described below observing results in the terminal

- modify the priority of one thread
- modify sleep periods in k_msleep() functions
- change the priority of one and later of both threads to negative numbers making them cooperative
- change the method of passing control to OS - comment k_msleep() function call, uncomment k_yield()
- change version of k_msleep() to k_usleep(), k_sleep() - the latter one requires the k_timeout_t type structure as the parameter

[1)]
https://academy.nordicsemi.com/courses/nrf-connect-sdk-fundamentals/lessons/lesson-7-multithreaded-applications/topic/exercise-1-7/

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

Permanent link:
https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:s2:aih:lab:nrf52threads?rev=1701418688

Last update: **2023/12/01 09:18**