Lesson 3: Threads

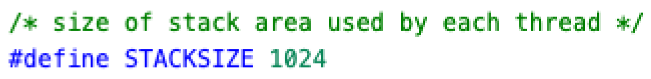
- ☑ 3.1 Introduction
- ☑ 3.2 Commands
- ☑ 3.3 Kconfig
- ☑ 3.4 Exercise

What are threads?

A thread is an isolated instance that is responsible for the execution of some task. While a microcontroller usually only has 1 CPU, the RTOS is able to have multiple tasks execute (seemingly) simultanously by exchanging the thread that gets run on the CPU as dictated by the scheduler.

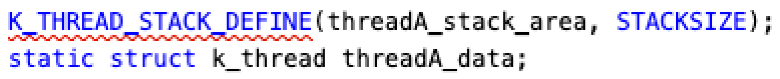
Some key concepts:

- **Stack area**: a region of memory used for the thread's stack. The size can be adjusted as required by the thread's processing.



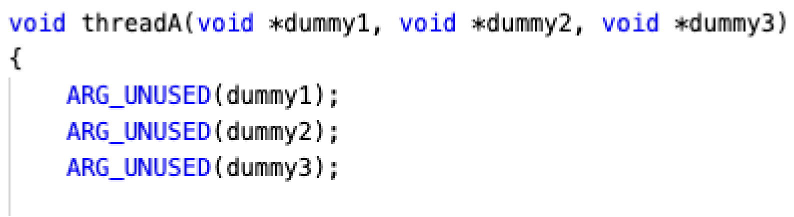
```
/* size of stack area used by each thread */
#define STACKSIZE 1024
```

- **Thread control block**: for internal bookkeeping of the thread's metadata. An instance of the type k_thread.



```
K_THREAD_STACK_DEFINE(threadA_stack_area, STACKSIZE);
static struct k_thread threadA_data;
```

- **Entry point function**: invoked when the thread is started. Up to 3 argument values can be passed to this function.



```
void threadA(void *dummy1, void *dummy2, void *dummy3)
{
    ARG_UNUSED(dummy1);
    ARG_UNUSED(dummy2);
    ARG_UNUSED(dummy3);
```

ARG_UNUSED is needed to indicate that the 3 arguments are not used in our thread function.

- **Scheduling policy**: intstructs the kernel's scheduler how to allocate CPU time to the thread. (This will be covered in Scheduling)

- **Execution mode**: can be supervisor or user mode. By default, threads run in supervisor mode and allow access to privileged CPU instructions, the entire memory address space, and peripherals. User mode threads have a reduced set of privileges.

The specifics of how to define a thread will be discussed in the next section

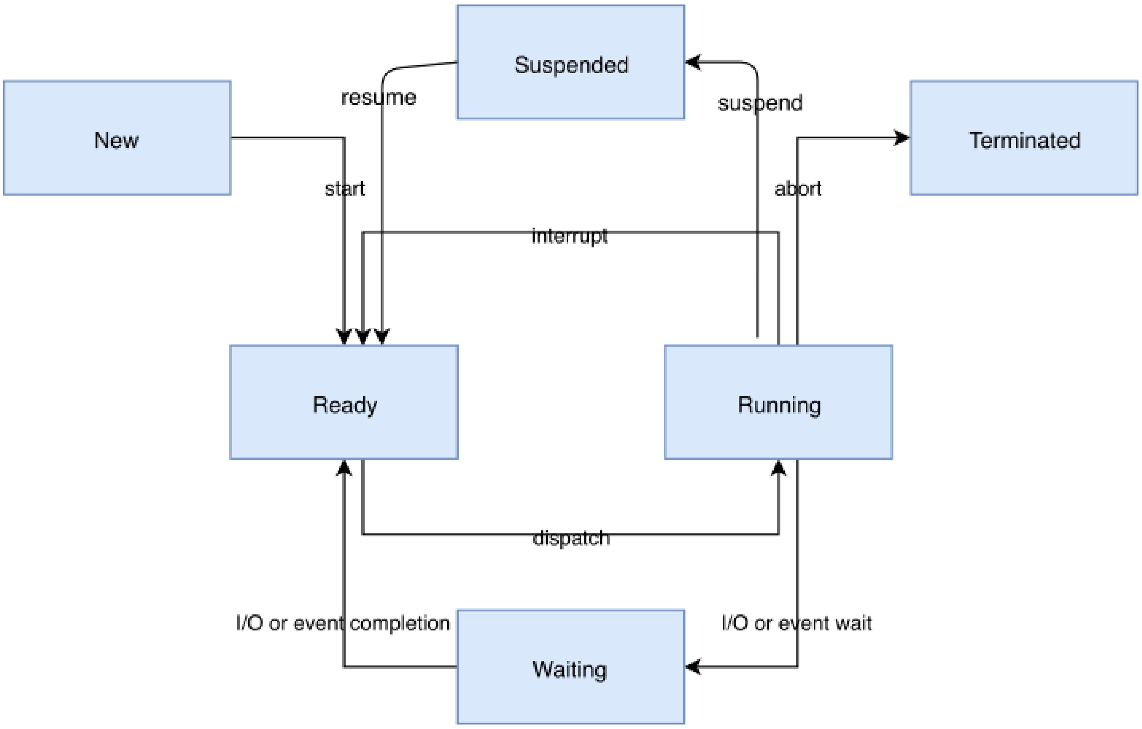
How does Zephyr choose which thread to run?

"Thread is ready" = eligible to be selected as the next running thread.

Following factors can make a thread unready:

- Thread has not been started
- Waiting for a kernel object to complete an operation (for example, the thread is taking semaphore that is unavailable)
- Waiting for a timeout to occur
- Thread has been suspended
- Thread has terminated or aborted

The following diagram shows all the possible states a thread can find itself:



How do I define threads in Zephyr?

A thread is spawned by defining its stack area and its thread control block, and then calling k_thread_create().

The stack area must be defined using K_THREAD_STACK_DEFINE or K_KERNEL_STACK_DEFINE to ensure it is properly set up in memory.

The thread spawning function returns its thread id, which can be used to reference the thread.

```
#define MY_STACK_SIZE 500
#define MY_PRIORITY 5

extern void my_entry_point(void *, void *, void *);

K_THREAD_STACK_DEFINE(my_stack_area, MY_STACK_SIZE);
struct k_thread my_thread_data;

k_tid_t my_tid = k_thread_create(&my_thread_data, my_stack_area,
                                 K_THREAD_STACK_SIZEOF(my_stack_area),
                                 my_entry_point,
                                 NULL, NULL, NULL,
                                 MY_PRIORITY, 0, K_NO_WAIT);
```

In order to define a thread you'll need to initiate some parameters:

```
k_tid_t k_thread_create(struct k_thread *new_thread, k_thread_stack_t *stack, size_t
stack_size, k_thread_entry_t entry, void *p1, void *p2, void *p3, int prio, uint32_t options,
k_timeout_t delay)
```

Parameters:

- new_thread – Pointer to uninitialized struct k_thread
- stack – Pointer to the stack space.
- stack_size – Stack size in bytes.
- entry – Thread entry function.
- p1 – 1st entry point parameter.
- p2 – 2nd entry point parameter.
- p3 – 3rd entry point parameter.
- prio – Thread priority.
- options – Thread options.
- delay – Scheduling delay, or K_NO_WAIT (for no delay).

Returns:

- ID of new thread.

Alternatively, a thread can be declared at compile time by calling K_THREAD_DEFINE. Observe that the macro defines the stack area, control block, and thread id variables automatically.

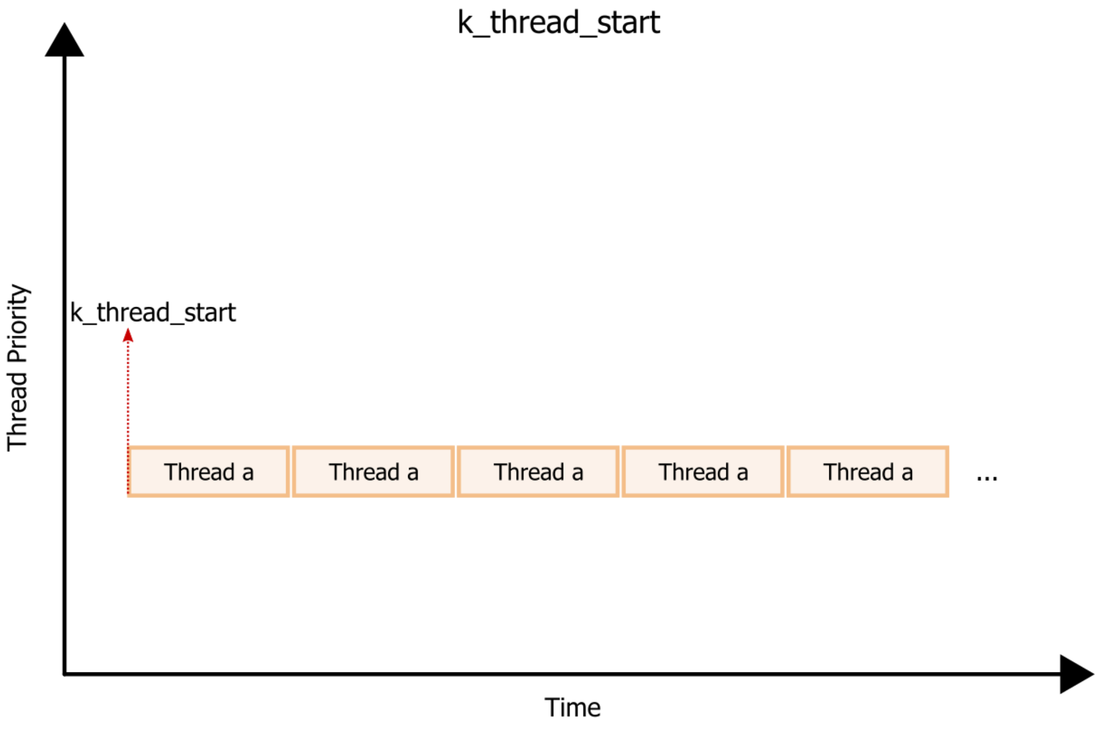
The following code has the same effect as the code segment above.

```
#define MY_STACK_SIZE 500
#define MY_PRIORITY 5

extern void my_entry_point(void *, void *, void *);

K_THREAD_DEFINE(my_tid, MY_STACK_SIZE,
                my_entry_point, NULL, NULL, NULL,
                MY_PRIORITY, 0, 0);
```

Thread commands

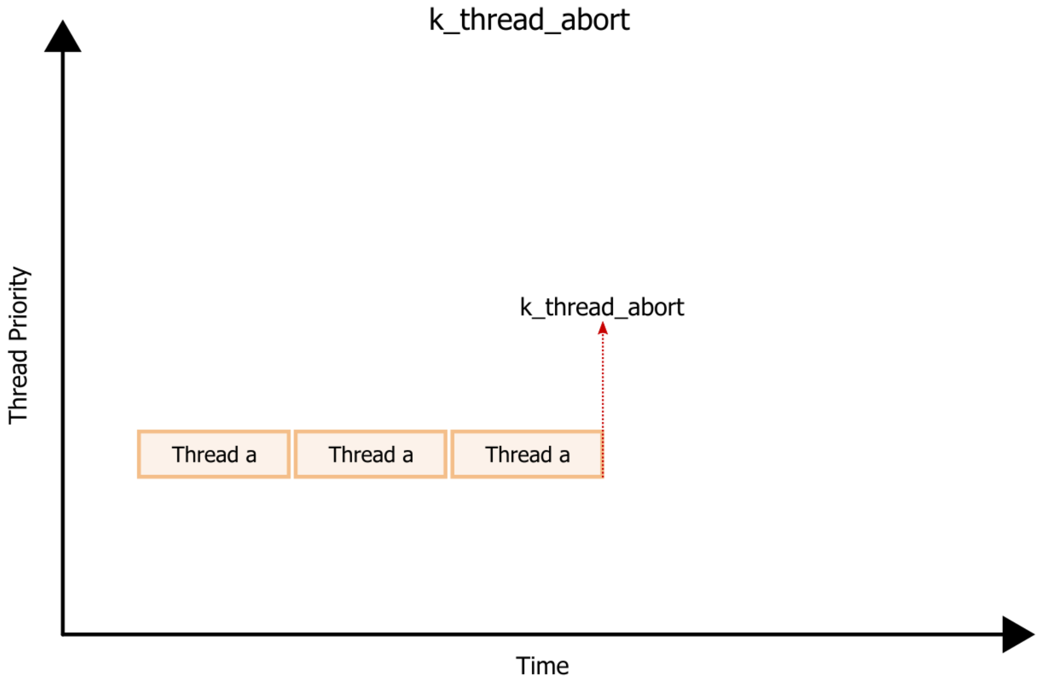
k_thread_start()

A thread must be created before it can be used.



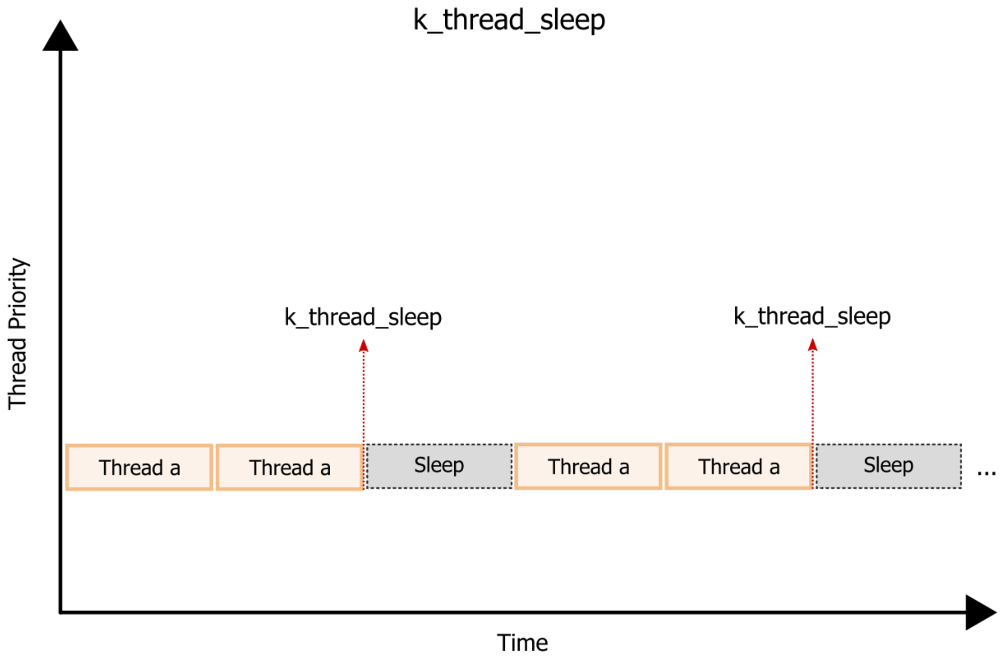
k_thread_abort()

Abort a thread. Thread is taken off all kernel queues.



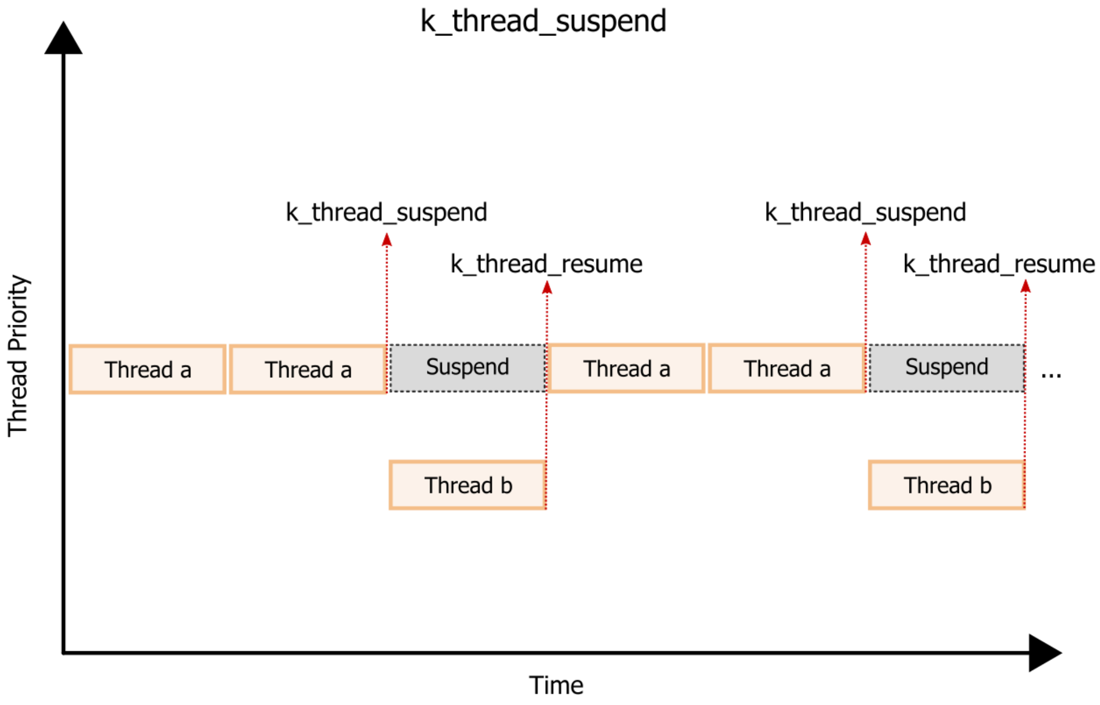
k_sleep()

A thread can prevent itself from executing for a specified amount of time. A sleeping thread becomes executable automatically once the time limit is reached.



k_thread_suspend()

Prevent a thread from executing for an indefinite period of time. Once suspended, use k_thread_resume() to re-start.

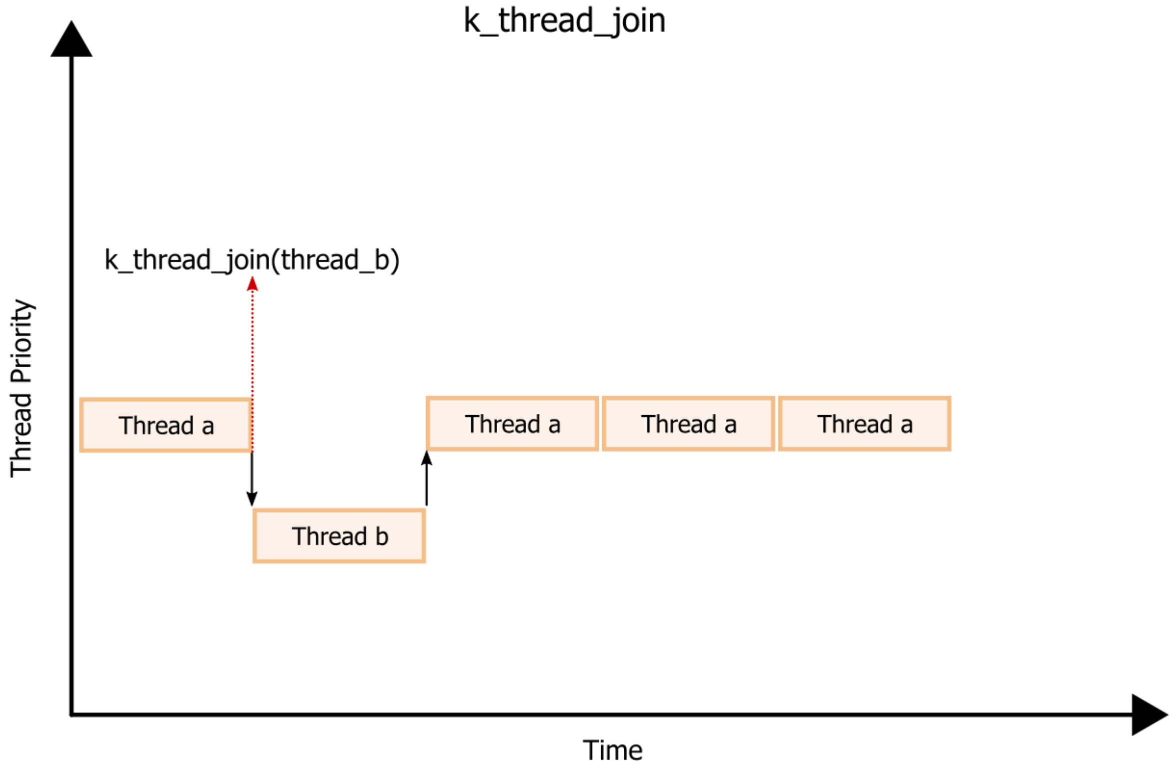


k_thread_join()

Sleep until a thread exits.

For example:

- thread_b is responsible for setting up a hardware interface
- thread_a is responsible for processing data from this interface
- As long as thread_b has not exited, thread_a can't start, so we'll use k_thread_join(thread_b, timeout) in this case.



Kconfig

Threads are always included in the system configuration, therefore no additional configs need to be set.

However optionally the following configuration options can be set:

Kconfig	Description
CONFIG_MAIN_THREAD_PRIORITY	Priority of initialization/main thread
CONFIG_MAIN_STACK_SIZE	Size of stack for initialization and main thread
CONFIG_IDLE_STACK_SIZE	Size of stack for idle thread
CONFIG_THREAD_CUSTOM_DATA	This option allows each thread to store 32 bits of custom data, which can be accessed using the k_thread_custom_data_xxx() APIs.
CONFIG_NUM_COOP_PRIORITIES	Number of cooperative priorities configured in the system

Kconfig	Description
CONFIG_NUM_PREEMPT_PRIORITIES	Number of preemptible priorities available in the system
CONFIG_TIMESLICING	This option enables time slicing between preemptible threads of equal priority.
CONFIG_TIMESLICE_SIZE	This option specifies the maximum amount of time a thread can execute before other threads of equal priority are given an opportunity to run. A time slice size of zero means "no limit" (i.e. an infinitely large time slice).
CONFIG_TIMESLICE_PRIORITY	This option specifies the thread priority level at which time slicing takes effect; threads having a higher priority than this ceiling are not subject to time slicing.
CONFIG_USERSPACE	When enabled, threads may be created or dropped down to user mode, which has significantly restricted permissions and must interact with the kernel via system calls.

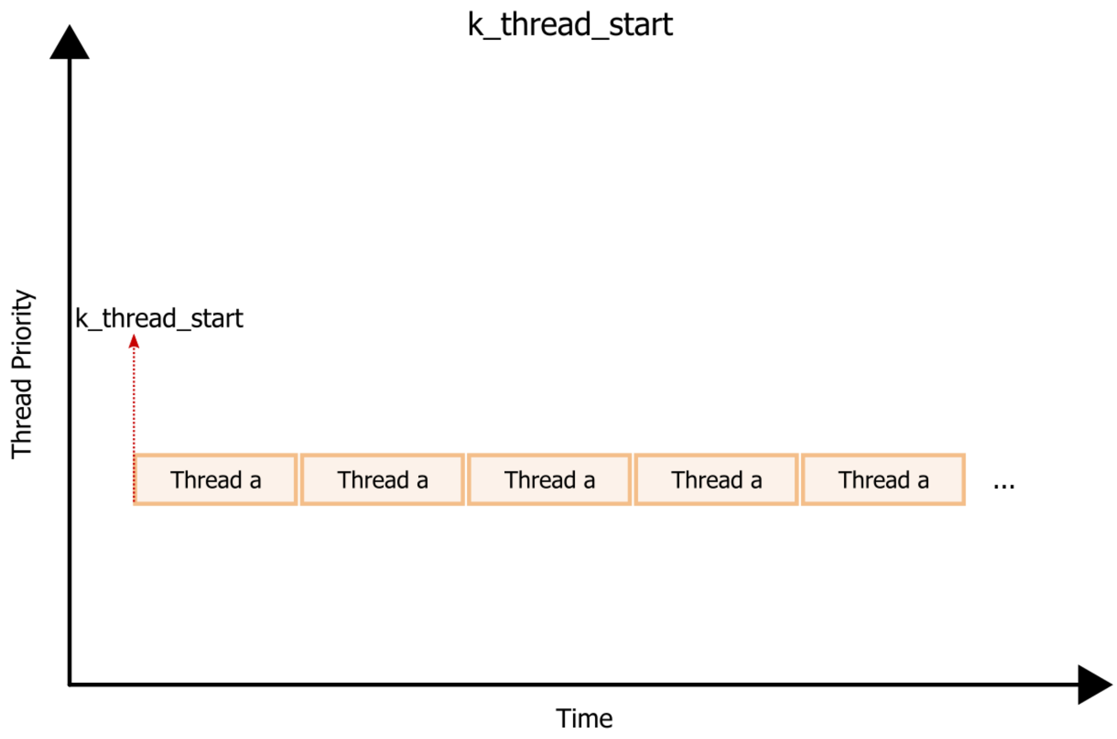
Exercises

For solving the exercises I recommend you create your own folder inside `zephyr-rtos-tutorial` which contains the following filetree.

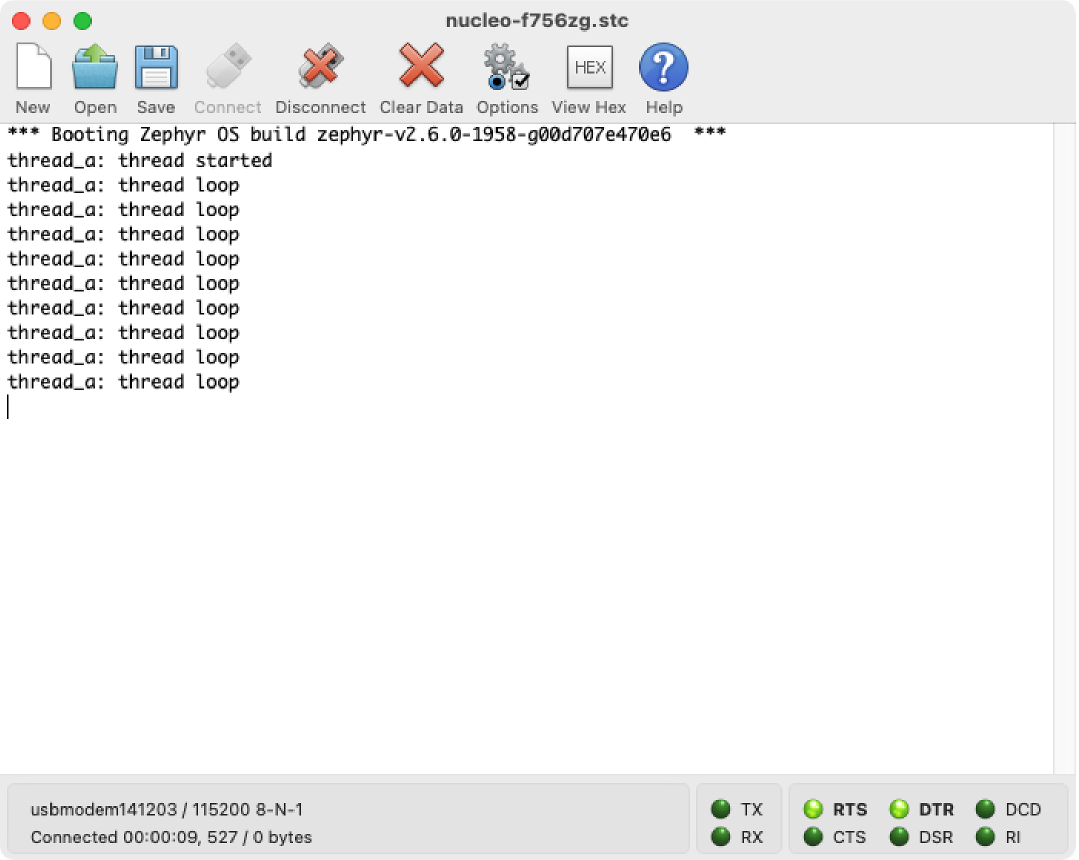
```
.
└── my-exercises
    ├── threads
    │   ├── thread-start
    │   ├── thread-start-define
    │   ├── thread-abort
    │   ├── thread-sleep
    │   ├── thread-suspend
    │   └── thread-join
    ├── gpio
    ├── scheduling
    ├── debugging
    ├── interrupts
    ├── timers
    ├── mutexes
    ├── semaphores
    ├── logging
    └── networking
```

thread creation: main

- Use k_thread_create() to create a thread
- Implement the following



- Output the following serial using printk()



```
*** Booting Zephyr OS build zephyr-v2.6.0-1958-g00d707e470e6  ***
thread_a: thread started
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
```

usbmodem141203 / 115200 8-N-1
Connected 00:00:09, 527 / 0 bytes

solution: `exercises/threads/thread-start`

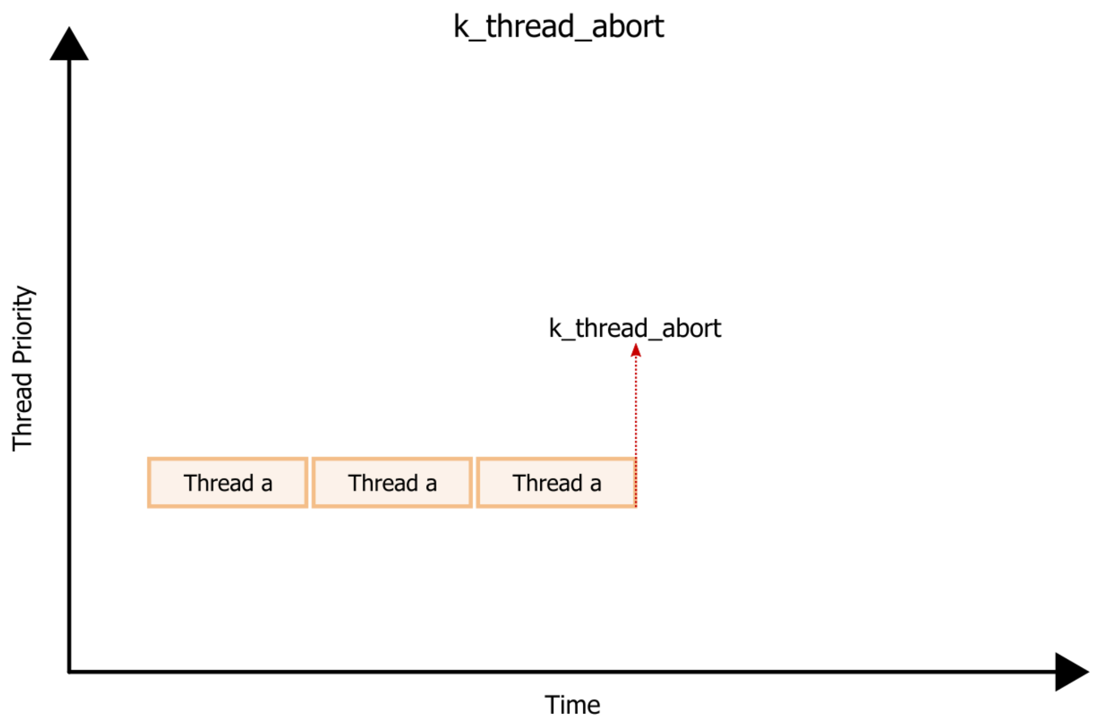
thread creation: define

- Same as previous, but this time using K_THREAD_DEFINE to create thread

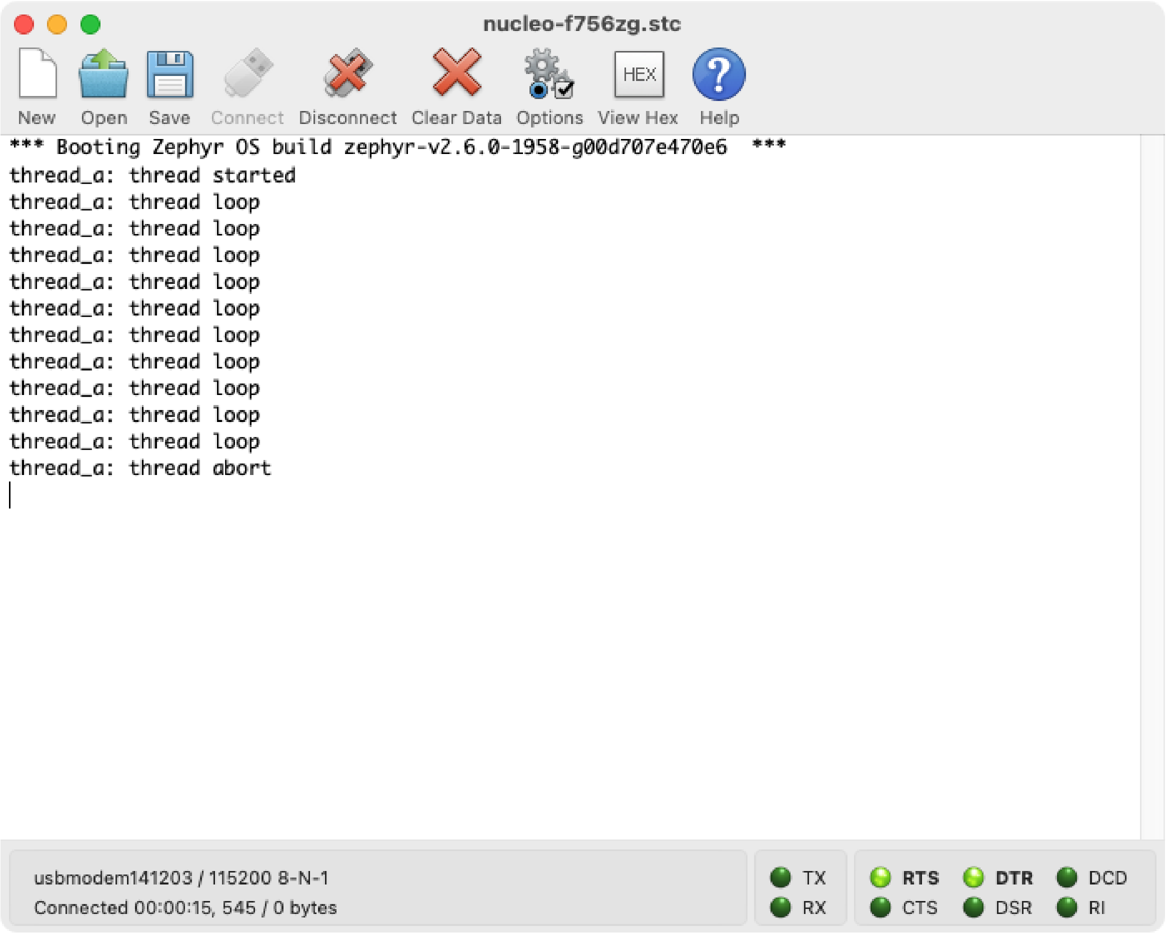
solution: `exercises/threads/thread-start-define`

thread abort

- Implement the following



- Output the following serial using printk()

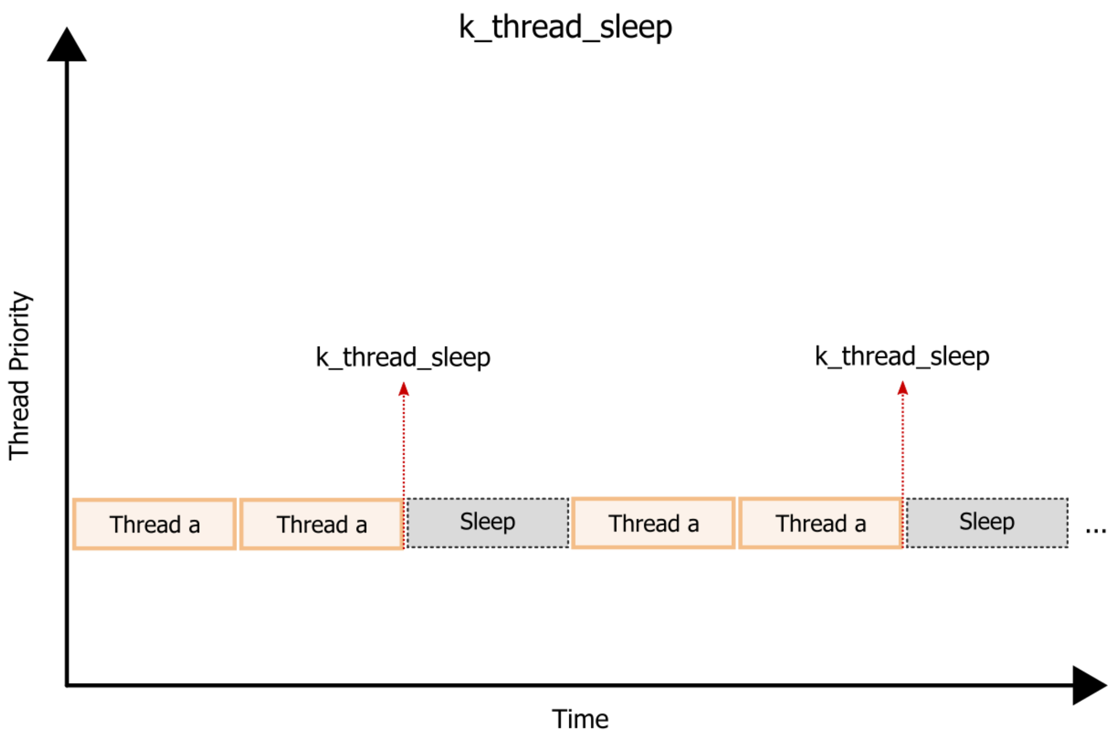


```
*** Booting Zephyr OS build zephyr-v2.6.0-1958-g00d707e470e6  ***
thread_a: thread started
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread abort
```

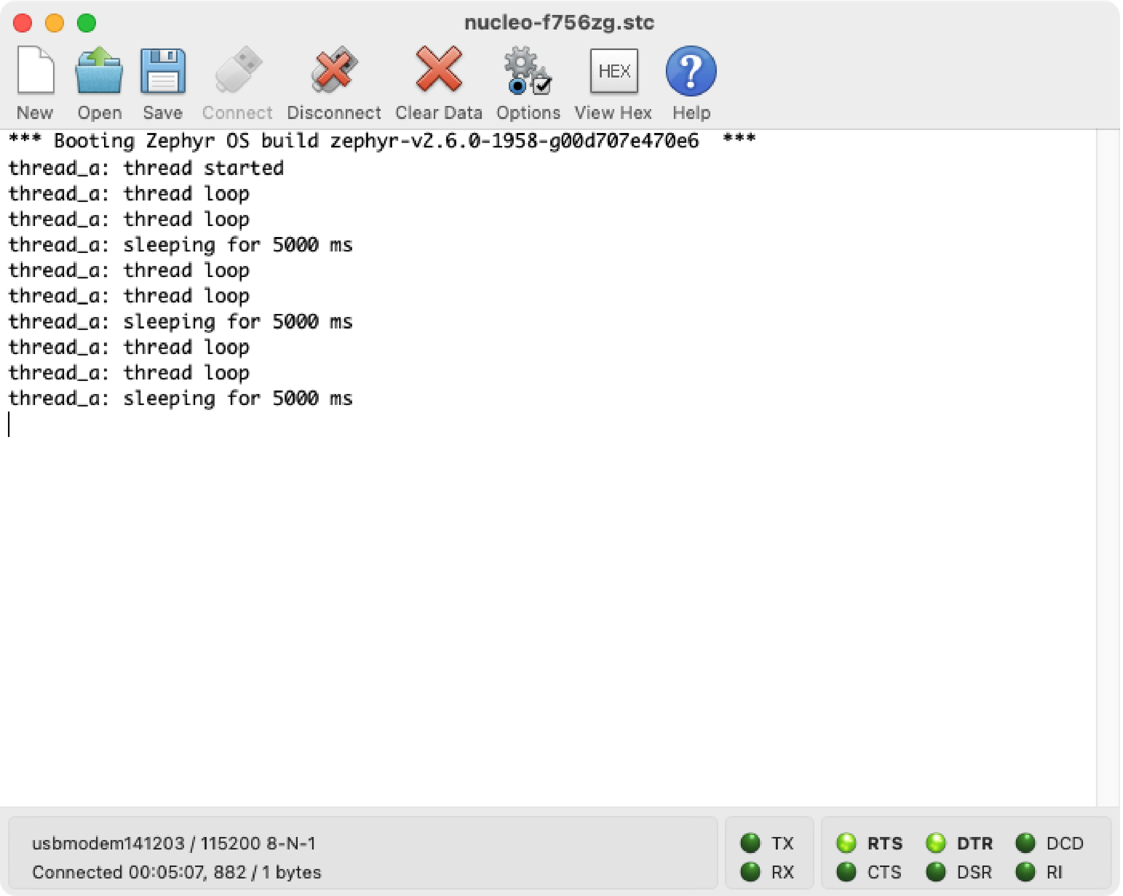
solution: `exercises/threads/thread-abort`

thread sleep

- Implement the following



- Output the following serial using printk()



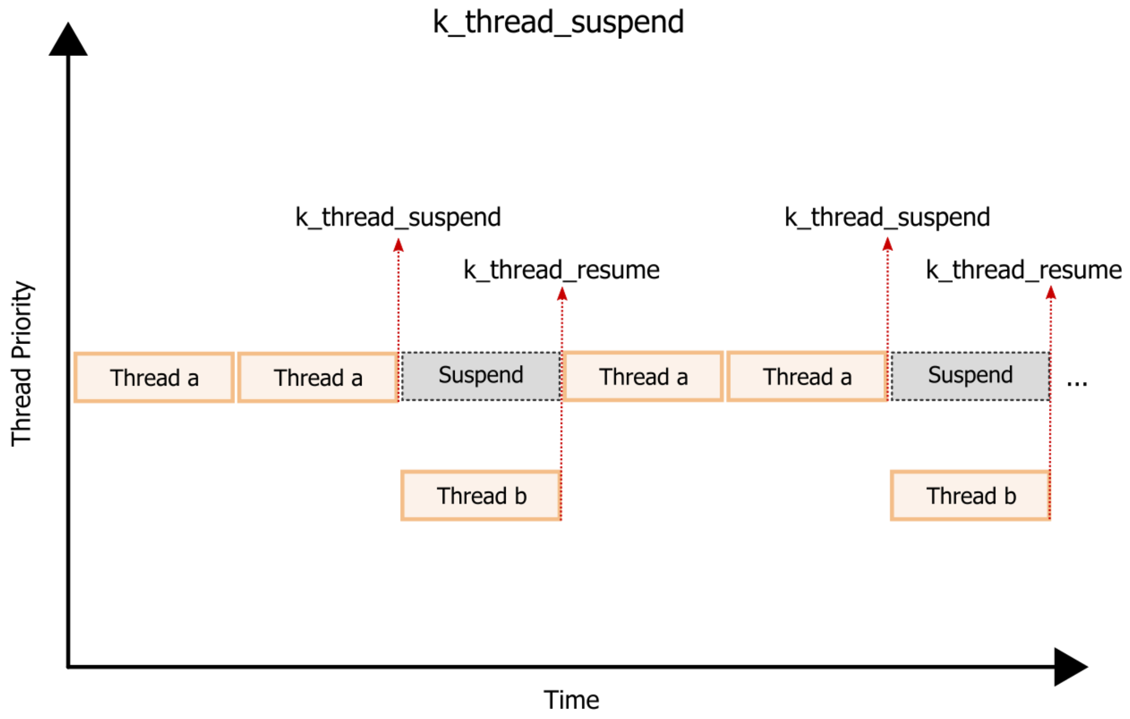
```
*** Booting Zephyr OS build zephyr-v2.6.0-1958-g00d707e470e6  ***
thread_a: thread started
thread_a: thread loop
thread_a: thread loop
thread_a: sleeping for 5000 ms
thread_a: thread loop
thread_a: thread loop
thread_a: sleeping for 5000 ms
thread_a: thread loop
thread_a: thread loop
thread_a: sleeping for 5000 ms
```

usbmodem141203 / 115200 8-N-1
Connected 00:05:07, 882 / 1 bytes

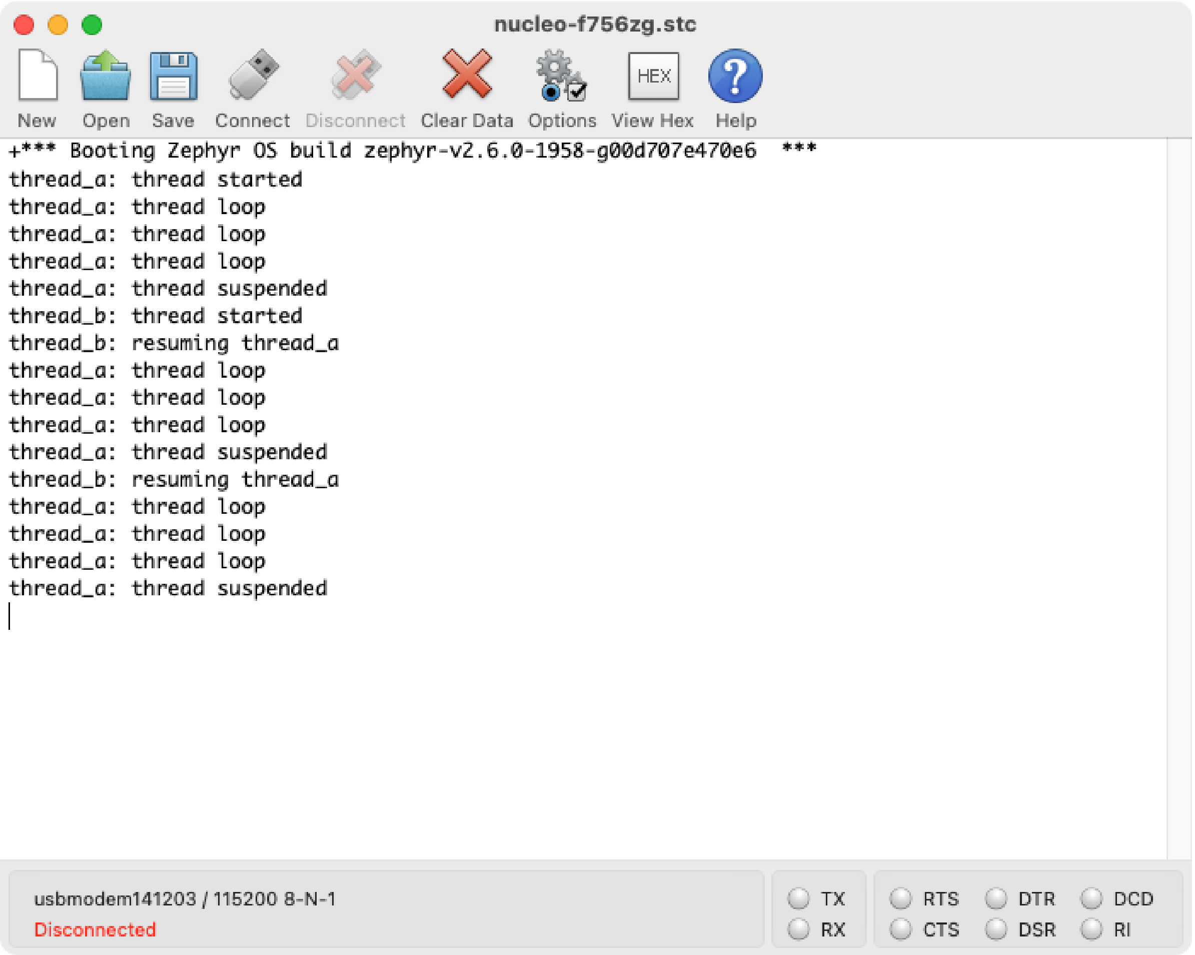
solution: `exercises/threads/thread-sleep`

thread suspend

- Implement the following



- Output the following serial using printk()



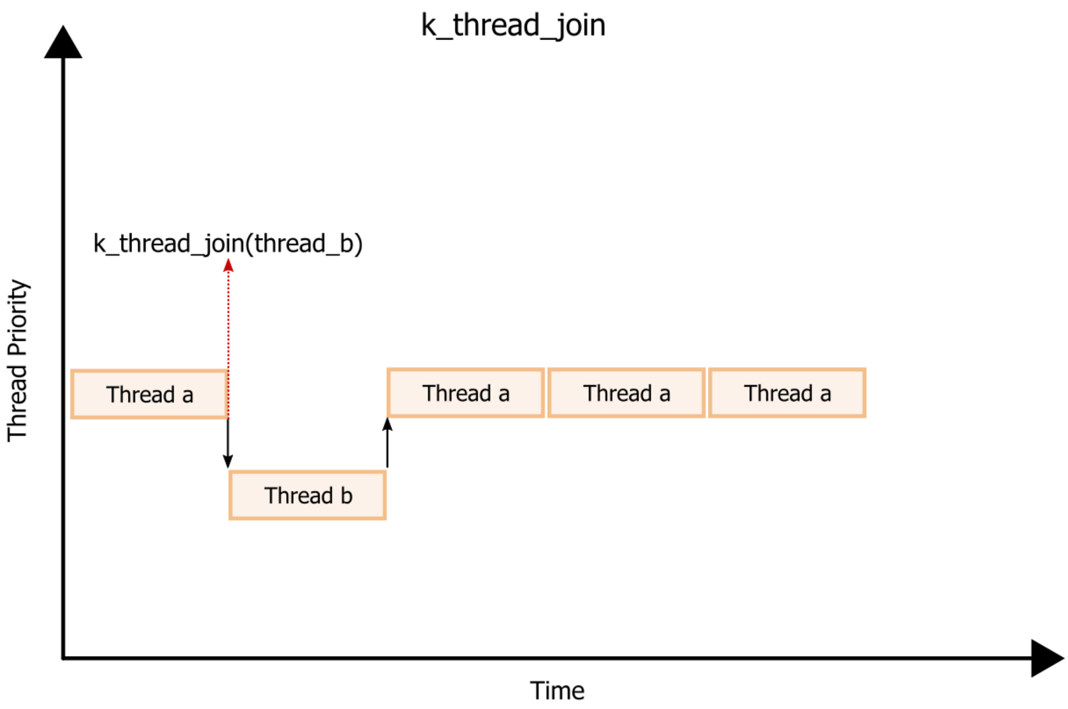
```
+*** Booting Zephyr OS build zephyr-v2.6.0-1958-g00d707e470e6  ***
thread_a: thread started
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread suspended
thread_b: thread started
thread_b: resuming thread_a
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread suspended
thread_b: resuming thread_a
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread suspended
```

usbmodem141203 / 115200 8-N-1
Disconnected

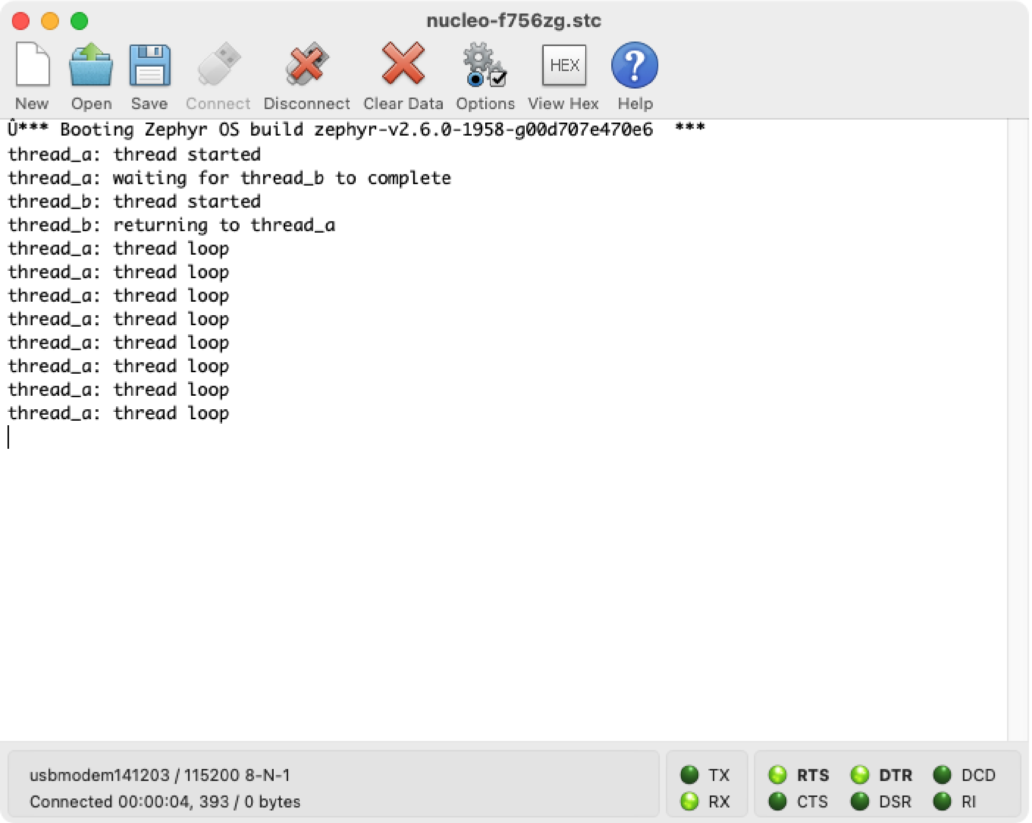
solution: `exercises/threads/thread-suspend`

thread join

- Implement the following



- Output the following serial using printk()



```
Û*** Booting Zephyr OS build zephyr-v2.6.0-1958-g00d707e470e6  ***
thread_a: thread started
thread_a: waiting for thread_b to complete
thread_b: thread started
thread_b: returning to thread_a
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
thread_a: thread loop
```

solution: `exercises/threads/thread-join`