**Lesson 9: Timers**

- 9.1 Introduction
- 9.2 Commands
- 9.3 Kconfig
- 9.4 Exercise

**What is a timer?**

A timer is a kernel object that measures the passage of time using the kernel's system clock. When a timer's specified time limit is reached it can perform an application-defined action, or it can simply record the expiration and wait for the application to read its status.

Any number of timer can be defined (limited only by available RAM). Each timer is referenced by its memory address.

A timer has the following key priorities:

- A duration specifying the time interval before the timer expires for the first time. This is a k_timeout_t value that may be initialized via different units.
- A period specifying the time interval between all timer expirations after the first one, also a k_timeout_t. It must be non-negative. A period of K_NO_WAIT (i.e. zero) or K_FOREVER means that the times is a one shot timer that stops after a single expiration. (For example then, if a timer is started with a duration of 200 and a period of 75, it will first expire after 200ms and then every 75ms after that.)
- An expiry function that is executed each time the timer expires. The function is executed by the system clock interrupt handler. If no expiry function is required a NULL function can be specified.
- A stop function that is executed if the timer is stopped prematurely while running. The function is executed by the thread that stops the timer. If no stop function is required a NULL function can be specified.
- A status value that indicates how many times the timer has expired since the status value was last read.

A timer must be initialized before it can be used. This specifies its expiry function and stop function values, sets the timer's status to zero, and puts the timer into the stopped state.

A timer is started by specifying a duration and a period. The timer's status is reset to zero, then the timer enters the running state and begins counting down towards expiry.

Note that the timer's duration and period parameters specify minimum delays that will elapse. Because of internal system timer precision (and potentially runtime interactions like interrupt delay) it is possible that more time may have passed as measured by reads from the relevant system time APIs. But at least this much time is guaranteed to have elapsed.

When a running timer expires its status is incremented and the timer executes its expiry function, if one exists; if a thread is waiting on the timer, it is unblocked. If the timer's period is zero the timer enters the stopped state; otherwise the timer restarts with a new duration equal to its period.

A running timer can be stopped in mid-countdown, if desired. The timer's status is left unchanged, then the timer enters the stopped state and executes its stop function, if one exists. If a thread is waiting on the timer, it is unblocked. Attempting to stop a non-running timer is permitted, but has no effect on the timer since it is already stopped.

A running timer can be restarted in mid-countdown, if desired. The timer's status is reset to zero, then the timer begins counting down using the new duration and period values specified by the caller. If a thread is waiting on the timer, it continues waiting.

A timer's status can be read directly at any time to determine how many times the timer has expired since its status was last read. Reading a timer's status resets its value to zero. The amount of time remaining before a timer expires can also be read; a value of zero indicates that the timer has stopped.

A thread may read a timer's status indirectly by synchronizing with the timer. This blocks the thread until the timer's status is non-zero (indicating that it has expired at least once) or the timer is stopped; if the timer status is already non-zero or the timer is already stopped the thread continues with waiting. The synchronization operation returns the timer's status and resets it to zero.

Note: Only a single user should examine the status of any given timer, since reading the status (directly or indirectly) changes its value. Similarly, only a single thread at a time should synchronize with a given timer. ISRs are not permitted to synchronize with timers, since ISRs are not allowed to block.

**How to define a timer?**

A timer is defined using a variable of type k_timer. It must then be initialized by calling k_timer_init().

The following code defines and initializes a timer.

```
struct k_timer my_timer;
extern void my_expiry_function(struct k_timer *timer_id);

k_timer_init(&my_timer, my_expiry_function, NULL);
```

Alternatively, a timer can be defined and initialized at compile time by calling K_TIMER_DEFINE.

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

```
K_TIMER_DEFINE(my_timer, my_expiry_function, NULL);
```

The following code uses a timer to perform a non-trivial action on a periodic basis. Since the required work cannot be done at interrupt level, the timer's expiry function submits a work item to the system workqueue, whose thread performs the work.

```
void my_work_handler(struct k_work *work)
{
    /* do the processing that needs to be done periodically */
    ...
}

K_WORK_DEFINE(my_work, my_work_handler);

void my_timer_handler(struct k_timer *dummy)
{
    k_work_submit(&my_work);
}

K_TIMER_DEFINE(my_timer, my_timer_handler, NULL);

...

/* start periodic timer that expires once every second */
k_timer_start(&my_timer, K_SECONDS(1), K_SECONDS(1));
```

The following code reads a timer's status directly to determine if the timer has expired or not.

```
K_TIMER_DEFINE(my_status_timer, NULL, NULL);

...

/* start one shot timer that expires after 200 ms */
k_timer_start(&my_status_timer, K_MSEC(200), K_NO_WAIT);

/* do work */
...

/* check timer status */
if (k_timer_status_get(&my_status_timer) > 0) {
    /* timer has expired */
} else if (k_timer_remaining_get(&my_status_timer) == 0) {
    /* timer was stopped (by someone else) before expiring */
} else {
    /* timer is still running */
}
```

The following code performs timer status synchronization to allow a thread to do useful work while ensuring that a pair of protocol operations are seperated by the specified time interval.

```
K_TIMER_DEFINE(my_sync_timer, NULL, NULL);

...

/* do first protocol operation */
...

/* start one shot timer that expires after 500 ms */
```

```
k_timer_start(&my_sync_timer, K_MSEC(500), K_NO_WAIT);

/* do other work */
...

/* ensure timer has expired (waiting for expiry, if necessary) */
k_timer_status_sync(&my_sync_timer);

/* do second protocol operation */
...
```

**When to use a timer?**

Use a timer to initiate an asynchronous operation after a specified amount of time.

Use a timer to determine whether or not a specified amount of time has elapsed. In particular, timers should be used when higher precision and/or unit control is required than that afforded by the simpler k_sleep() and k_usleep() calls.

Use a timer to perform other work while carrying out operations involving time limits.

Note: If a thread needs to measure the time required to perform an operation it can read the system clock or the hardware clock directly, rather than using a timer.

| Command | Description |
|---|---|
| k_timer_init | Initialize a timer. This routine initializes a timer, prior to its first use. |
| k_timer_start | Start a timer. This routine starts a timer, and resets its status to zero. The timer begins counting down using the specified duration and period values. |
| k_timer_stop | Stop a timer. This routine stops a running timer prematurely. The timer's stop function, if one exists, is invoked by the caller. |
| k_timer_status_get | Read timer status. This routine reads the timer's status, which indicates the number of times it has expired since its status was last read. |
| k_timer_status_sync | Synchronize thread to timer expiration. This routine blocks the calling thread until the timer's status is non-zero (indicating that it has expired at least once since it was last examined) or the timer is stopped. If the timer status is already non-zero, or the timer is already stopped, the caller continues without waiting. |

| Command | Description |
| --- | --- |
| k_timer_expires_ticks | Get next expiration time of a timer, in system ticks. |
| k_timer_remaining_ticks | Get time remaining before a timer next expires, in system ticks. |
| k_timer_remaining_get | Get time remaining before a timer next expires. |