3.4.4 TM_WKAFTER

NAME

tm_wkafter -- "Wake After Interval"

SYNOPSIS

```
#include <time.h>
uint tm_wkafter ( ticks )

        uint ticks;     /* number of ticks to wait */
```

DESCRIPTION

The executive stops the execution of the requesting task until the specified number of system clock ticks have occurred. Execution resumes at the location following the *tm_wkafter* directive.

If the system clock frequency is 100 ticks per second, and the requester wants to wait for 2 seconds, then the input parameter will be 100*2, or 200 ticks.

The relative scheduling priority of the task will influence when the task actually gets to run again. A manual round-robin may be performed by executing *tm_wkafter(0)*. This causes the requesting task to yield the processor to other tasks at the same priority, if any exist.

The number of ticks remaining until the task is awakened will not be modified by the executive if the system date and time are reset via the *tm_set* directive.

The maximum duration is 2**32 - 1 ticks.

RETURN VALUE

Tm_wkafter always succeeds and returns 0.

ERROR CONDITIONS

None.

NOTES

Not callable from ISR.

The requesting task will be blocked until the interval is expired.

3.4.5 TM_WKWHEN

NAME

#include <time.h>
tm_wkwhen -- "Wake When Date and Time"

SYNOPSIS

```
#include <time.h>
uint tm_wkwhen ( timebuf )

        struct time_ds *timebuf;    /* pointer to time and date structure */
```

DESCRIPTION

The executive stops execution of the requesting task until the specified date and time is reached. Execution resumes at the location following the *tm_wkwhen* directive.

If the system date and time are reset via the *tm_set* directive, the requested date and time when the task will be awakened will be modified by the executive. Therefore, if the date and time are reset *ahead* of the requested time, the task may be awakened *late*.

The relative scheduling priority of the task will influence when the task actually gets to run again.

The current elapsed ticks in the *ticks* field within the timebuf structure are ignored.

RETURN VALUE

If *tm_wkwhen* is successful, then 0 is returned.

If the date and time are invalid, an error code is returned.

ERROR CONDITIONS

Date and time have not been set.

Date input parameter error.

Time input parameter error.

NOTES

Not callable from ISR.

The requesting task will be blocked until the date and time is reached.

3.4.8 TM_EVAFTER

NAME

tm_evafter -- "Send Event After Interval"

SYNOPSIS

```
#include <time.h>
uint tm_evafter ( ticks, event, &tmid )

        uint ticks;     /* number of ticks until event */
        uint event;     /* event condition */
        uint tmid;      /* timer id - returned by this call */
```

DESCRIPTION

The *tm_evafter* directive allows a task to receive a timer event after the specified number of system clock ticks have occurred. The requesting task is not blocked by this call. To receive the event, the *ev_receive* directive must be used.

If the system clock frequency is 100 ticks per second, and the requester wants to receive an event after 2 seconds, then the input parameter will be 100*2, or 200 ticks.

The number of ticks remaining until the timer event is sent will not be modified by the executive if the system date and time are reset via the *tm_set* directive.

The maximum duration is 2**32 - 1 ticks.

RETURN VALUE

Tm_evafter always succeeds, the *tmid* is filled in, and 0 is returned.

ERROR CONDITIONS

Too many timers.

NOTES

Not callable from ISR.

Will not cause a preempt.

The requesting task will not be blocked.

3.4.7 TM_EVWHEN

NAME

tm_evwhen -- "Send Event When Date and Time"

SYNOPSIS

```
#include <time.h>
uint tm_evwhen ( timebuf, event, &tmid )

        struct time_ds *timebuf;    /* pointer to time and date structure */
        uint event;                 /* event condition */
        uint tmid;                  /* timer id - returned by this call */
```

DESCRIPTION

The *tm_evwhen* directive allows a task to receive a timer event when the specified date and time is reached. The requesting task is not blocked by this call. To receive the event, the *ev_receive* directive must be used.

If the system date and time are reset via the *tm_set* directive, the requested date and time of the timer event will be modified by the executive. Therefore, if the date and time are reset *ahead* of the requested time, the task may receive the timer event *late*.

The current elapsed ticks in the *ticks* field within the timebuf structure are ignored.

RETURN VALUE

If *tm_evwhen* is successful, the *tmid* is filled in, and 0 is returned.

If the date and time are invalid, an error code is returned.

ERROR CONDITIONS

Too many timers.

Date and time have not been set.

Date input parameter error.

Time input parameter error.

NOTES

Not callable from ISR.

Will not cause a preempt.

The requesting task will not be blocked.

3.4.8 TM_CANCEL

NAME

tm_cancel -- "Cancel Timer Event"

SYNOPSIS

```
#include <time.h>
uint tm_cancel ( tmid )

        uint tmid;    /* timer id - as returned from tm_evafter or tm_evwhen */
```

DESCRIPTION

The *tm_cancel* directive allows a task to cancel the timer event identified by the *tmid*. The timer event may have been scheduled by the *tm_evafter* or *tm_evwhen* directives.

RETURN VALUE

If *tm_cancel* successfully canceled the timer event, then 0 is returned.

If the call was not successful, an error code is returned.

ERROR CONDITIONS

Invalid *tmid*.

Timer event not set.

NOTES

Not callable from ISR.

Will not cause a preempt.

The *timer event not set* error may occur if the specified *tmid* has expired. The caller may need to clear the event condition associated with the *tmid*.