3.3.5 SM\_V

NAME

sm\_v – “Release Semaphore”

SYNOPSIS

#include <semaphore.h>

uint sm\_v ( smid )

uint smid; /\* semaphore id as returned by srn\_create or sm\_ident \*/

DESCRIPTION

The current semaphore count of the semaphore identified in the smid field is incremented by one.

If the count is zero or negative, the first task in the waiting list is removed from the list and is made ready to await execution. If the task is of higher priority than the running task, it will cause a preempt.

RETURN VALUE

If sm\_v succeeded, then 0 is returned.

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

ERROR CONDITIONS

Invalid smid.

ISR cannot reference remote node.

NOTES

Can be called from within an ISR, except when the semaphore was not created on the local node.

May cause a preempt if a task waiting on the semaphore has a higher priority than the running task, and the preempt mode is in effect. A preempt will not occur if the task waiting exists on a remote processor in a multiprocessor configuration.

3.4 Time Management

The executive time manager supports two concepts of time: calendar time and elapsed time. These functions depend on periodic timer interrupts, and will not work without timer hardware.

The tm\_set directive allows a task to inform the time manager of the current date and time ( e.g., March 21, 1985; 12:04 ). The tm\_get directive allows a task to request the current date and time from the time manager ( e.g., March 27, 1986; 09:24 ).

The tm\_wkafter directive allows a task to remove itself from the running state and enter into a wait state for a specified number of ticks. After the elapsed time expires, the task is made ready.

The tm\_wkwhen directive allows a task to remove itself from the running state and enter into a wait state until a specific date and time is reached. When the date and time is reached, the task is made ready.

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.

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.

The tm\_cancel directive allows a task to cancel a timer event scheduled by the tm\_evafter or tm\_evwhen directives.

The tm\_tick directive allows a task or an interrupt service routine to inform the system of the occurrence of a system clock tick. This information is used to maintain correct calendar time, execute timeslicing, and decrement ticks from tasks which are currently being delayed or timing out.

Tick and timeslice are configuration parameters. A tick is defined to be some integral number of milliseconds. A timeslice is defined to be some integral number of ticks.

The directives provided by the time manager are:

|  |  |
| --- | --- |
| Directive | Function |
| tm\_set | Set date and time |
| tm\_get | Get date and time |
| tm\_wkafter | Wake after interval |
| tm\_wkwhen | Wake when date and time |
| tm\_evafter | Send event after interval |
| tm\_evwhen | Send event when data and time |
| tm\_cancel | Cancel timer event |
| tm\_tick | Announce tick |

3.4.1 Timebuf Structure

The time and date buffer structure is defined as follows:

struct time\_ds {

 struct t\_date date; /\* date \*/

 struct t\_time time; /\* time \*/

 uint ticks; /\* current elapsed ticks between seconds \*/

 };

Date is defined as follows:

 struct t\_date {

 short year; /\* year, A.D. \*/

 char month; /\* month, 1->12 \*/

 char day; /\* day, 1-> 31 \*/

 };

Time is defined as follows:

 struct t\_time {

 short hour; /\* hour, 0-> 23 \*/

 char minute; /\* minute, 0-> 59 \*/

 char second; /\* second, 0-> 59 \*/

 };

3.4.2 TM\_SET

NAME

tm\_set – “Set System Time and Date”

SYNOPSIS

#include <time.h >

uint tm\_set ( timebuf )

struct time\_ds \*timebuf; /\* pointer to time and date structure \*/

DESCRIPTION

The tm\_set directive sets or resets the date and time of all nodes within the system. The parameters within the time and date structure are validated, and an error will be returned if they are out of range.

After this call is successfully completed, the system maintains the date and time based upon the frequency of system clock ticks. The current date and time may be obtained by using the tm\_get directive.

RETURN VALUE

If tm\_set successfully set the date and time, then 0 is returned.

If the date and time were not successfully set, an error code is returned.

ERROR CONDITIONS

Date input parameter error.

Time input parameter error.

Ticks input parameter error.

NOTES

Callable from ISR.

May cause a preempt if setting the time causes a task on the timeout list to become ready, and that task has a higher priority than the running task, and the preempt mode is in effect.

3.4.3 TM\_GET

NAME

tm\_get – “Get System Time and Date”

SYNOPSIS

#include <time.h >

uint tm\_get ( timebuf )

struct time\_ds \*timebuf; /\* pointer to time and date structure \*/

DESCRIPTION

The requester is allowed to get the current date and time as maintained by the system. If the date and time have not been set via the tm\_set directive, then an error is returned, and the buffer contents will be meaningless.

RETURN VALUE

If tm\_get successfully got the date and time, timebuf will be filled in, and 0 is returned.

If the date and time have not been set, an error code is returned.

ERROR CONDITIONS

Date and time have not been set.

NOTES

Callable from ISR.

Will not cause a preempt.

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.

3.4.6 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 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 form 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.