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 sm_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 date 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 tim ., *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.