13.9 Putting A Process To Sleep

An application does not call *insertd*, nor does the application access the sleep queue directly. Instead, an application invokes system call *sleep* or *sleepms* to request a delay. The only difference between the two functions is the granularity of their arguments. An argument to *sleepms* specifies a delay in milliseconds, the smallest granularity delay that is possible when a clock interrupts every millisecond. An argument to *sleep* specifies a delay in seconds, which is easier to use in some cases. For example, a delay visible to a human is often expressed in seconds rather than milliseconds.

To avoid duplicating code, function *sleep* multiplies its argument by 1000 and invokes *sleepms*. The only interesting aspect of *sleep* is a check on its argument size: to avoid integer overflow, *sleep* limits the delay to a value that can be represented as a 32-bit integer. If the caller specifies a larger value, *sleep* returns *SYSERR*.

On a 32-bit processor, measuring delay in milliseconds provides an adequate range of delay for most applications. A 32-bit integer accommodates delays over 596 hours (24.8 days). Delays longer than 24.8 days can be managed by having a process repeatedly sleep for many days, awaken, check the time, and sleep again. On embedded systems that use 16-bit integers, however, millisecond delays mean that a caller can only express a delay of thirty-two seconds. Such systems seldom have much memory or processing power, so using a process to manage longer delays may not be feasible. Therefore, an operating system designed for a slow, 16-bit processor may choose a larger granularity for clock interrupts (e.g., tenths of seconds instead of milliseconds). If the clock generates interrupts every tenth of a second, a sleep function must be changed to measure delays in tenths of seconds.

The choice of delay granularity may also be limited by the speed of the processor. Handling clock interrupts can take a surprising amount of time because they never stop, even when no processes are sleeping. If a clock interrupts too fast, a processor will spend most of its time handling clock interrupts. Fortunately, processors have become extremely fast. As processor speeds increased, it became possible to increase the rate of clock interrupts, allowing the granularity of delays to decrease. Thus, the fastest processors allow microsecond delays.

Consider the state of a sleeping process. We said that to delay a process, *sleepms* inserts the process into the delta list of sleeping processes. When it has been moved to the list of sleeping processes, the process is no longer *ready* or *current*. In what state should it be placed? Sleeping differs from suspension, waiting to receive a message, or waiting for a semaphore. Thus, because none of the existing states suffices, a new process state must be added to the design. We call the new state *sleeping*, and denote it with symbolic constant *PR_SLEEP*. Figure 13.2 illustrates state transitions that include the sleeping state.

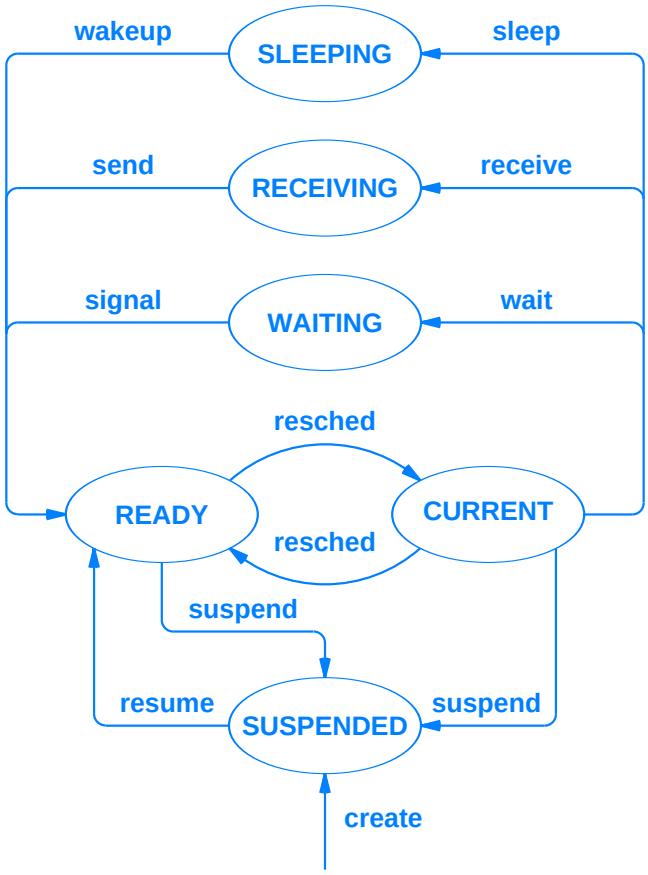

Figure 13.2 State transitions including the *sleeping* state.

The implementation of *sleepms*, shown below in file *sleep.c*, includes a special case: if a process specified a delay of zero, *sleepms* does not delay the process, but calls *resched* immediately. Otherwise, *sleepms* uses *insertd* to insert the current process in the delta list of sleeping processes, changes the state to *sleeping*, and calls *resched* to allow other processes to execute.

```
/* sleep.c - sleep sleepms */

#include <xinu.h>

#define MAXSECONDS      2147483         /* Max seconds per 32-bit msec  */
```

```
/*------------------------------------------------------------------------
 *  sleep  -  Delay the calling process n seconds
 *------------------------------------------------------------------------
 */
syscall sleep(
         int32 delay           /* Time to delay in seconds     */
        )
{
        if ( (delay < 0) || (delay > MAXSECONDS) ) {
                return SYSERR;
        }
        sleepms(1000*delay);
        return OK;
}

/*------------------------------------------------------------------------
 *  sleepms  -  Delay the calling process n milliseconds
 *------------------------------------------------------------------------
 */
syscall sleepms(
         int32 delay                   /* Time to delay in msec.       */
        )
{
        intmask mask;                  /* Saved interrupt mask         */

        if (delay < 0) {
                return SYSERR;
        }

        if (delay == 0) {
                yield();
                return OK;
        }

        /* Delay calling process */

        mask = disable();
        if (insertd(currpid, sleepq, delay) == SYSERR) {
                restore(mask);
                return SYSERR;
        }

        proctab[currpid].prstate = PR_SLEEP;
        resched();
        restore(mask);
        return OK;
}
```