Chapter 8
Time Management

Revised: Feb. 21, 2011

8.1 INTRODUCTION

Time management is concerned with operating system facilities and services that concern the measurement of real time. These include keeping track of the time of day, measuring the time used by various activities, starting or stopping activities at specific times, etc. These services are considered in this chapter.

Time management provides an important but relatively simple illustration of the use of interrupts, described in the previous chapter. A timing device may be considered an example of an input-output (I/O) device, so time management raises some of the issues to be considered in I/O device management. However, it is a specialized and simple device. The discussion of time management in this chapter can thus be seen as an introduction to some of the issues to be discussed more thoroughly in the chapters that follow.

8.2 CLOCKS

A computing system that provides timing services requires a physical device that measures time. Such a device is called a **clock**. Clocks are essential to almost any operating system. The requirements for time-of-day reporting and timing of resource usage for accounting in large operating systems represent two obvious uses. We will consider these and other uses of clocks in this chapter.

Clocks may be viewed as I/O devices, but they are unusual in that they do not transfer any information outside the computer. Their purpose is to provide a means of measuring time intervals. Because clocks cause periodic interrupts and often must be reset, control of clocks can be provided by a device driver, even though clocks differ from the other I/O devices considered in later chapter.

A clock is a relatively simple hardware device that merely generates interrupts upon the expiration of a time interval. The software must do all the work; this software usually consists of a clock driver and timer management routines.

It is possible to have multiple virtual clocks or software clocks based on a single physical clock. Examples include time-of-day clocks and clocks for timing a process's current time quantum. Other such clocks are needed for accounting in large systems so that users can be billed appropriately. Each such clock is represented by a data item in main memory, which is updated in an appropriate way whenever a signal from the hardware clock is received. Software clocks may also be treated as timers that may be allocated to processes as a logical resource for the purpose of timing events.

Types of Clocks

All processors have a **master clock** that drives every cycle of hardware operation. In today's processors these clocks produce control signals ("ticks") at a rate of billions of times a second. This is much too fast to be used directly by software.

The software-accessible clocks found in today's processors are **programmable clocks**. These have a counter register, which is decremented at a fixed periodic rate (derived from the master clock) until its value is 0, when it generates an interrupt. Some programmable clocks can automatically refresh the counter with a holding register, thus avoiding the overhead of processing an interrupt. A programmable clock can have the frequency at which it interrupts the CPU set by the software. Typical frequencies are between 20 and 100 ticks per second.

Because the resolution of programmable clocks is relative coarse, modern processors also provide a **cycle counter** which provides a running count of the number of master clock cycles since the system was started. This requires a large number of bits, typically 64, to avoid overflow. This type of counter never causes an interrupt, but may be read directly when extremely accurate timing information is required.

Clock Drivers

A **driver** is a software module that controls the operation of an external device**. Clock drivers** control clocks. These drivers provide service routines to read and modify the value or behavior of both hardware clocks and logical clocks. Their principal responsibility, however, is to deal with interrupts that represent **clock ticks**; these occur when a programmable clock's counter has reached zero.. A variety of techniques can be used to maintain a time-of-day clock and other software clocks. Almost all of them rely on these clock ticks to update the software clocks.

Interrupt handlers for clocks are often relatively simple and provide a good illustration of the format of an interrupt handler. In its simplest form, a clock interrupt handler might adjust a (logical) time-of-day clock that is maintained in clock ticks (past midnight) and then check to see if this clock exceeds the 24-hour maximum. In this case, the date must be advanced and the time-of-day clock set back to zero. The logic of this simple clock interrupt handler is shown below:

-

```
Clock_Interrupt: save registers
  increment Time_Of_Day_Clock
  IF Time_Of_Day_Clock > 24 hours THEN

    set Time_of_Day_Clock to 0
    advance date

  ENDIF
  restore registers
  RETURN from interrupt
```

8.3 TIME-OF-DAY SERVICES

A primary responsibility of time management is to maintain a record of the current date and time, and to provide this information in response to requests. Various methods are used to record time and date information; these vary widely in precision and accuracy.

One method of maintaining the time of day is simply to record clock ticks past midnight in a single integer data word (note that this value requires more than 16 bits). When the number of clock ticks in the counter equals the number of clock ticks in a 24-hour day, the date is advanced and the tick counter is reset to zero. In this case, requests for the time of day must convert this counter to hours, minutes and seconds.

Rather than keep clock ticks only, it is possible to use the same scheme above with separate counters indicating the number of seconds past midnight, and the number of clock ticks in the current second. When the counter that tracks the number of clock ticks equals the number of clock ticks in a second, the clock interrupt handler must increment the second counter and set the clock tick counter to zero. Conversions for time of day will now convert seconds past midnight to hours, minutes, and seconds. When the second counter reaches the number of seconds in a 24-hour day, the date can be advanced and the second counter set back to zero.

A separate counter is required for the current date. Although date information could be separated internally into day, month, and year, it is most often recorded as a single integer representing the number of days since some base date. This base date is usually chosen to be relatively recent, so the date information is compact. For example, MS-DOS uses January 1, 1980, while UNIX uses January 1, 1970. The Macintosh OS is a little more expansive, choosing a base date of January 1, 1904.

Alternately, the date and time may be combined into a single value, giving the number of seconds since (midnight on) the selected base date. Note that operating systems using this approach are somewhat insensitive to the "Year 2000" problem. Problems occur only when the present date or time, as a distance from the base date or time, exceeds the capacity of the word size used to store it.

-

The operating system must provide a service for decoding the time and date information, and delivering it in the form of year, month, date, hour, minute, and second. The current date must be calculated from the cumulative day count, taking note of leap years and other complications. In most cases the day of the week is also computed. Some OSs offer this information in a variety of formats; others provide one basic form and rely on library routines for any necessary translation.

Because the global location of computers is unpredictable, provisions should be made for setting time zone information (including daylight savings time). This is especially complicated for large networks and timesharing systems, since users may be in different time zones from the main system (and each other)!

8.4 TIMER MANAGEMENT

Besides keeping track of the time of day, a time management module is often called upon to provide a variety of service related to measuring and monitoring time intervals. These services are grouped under the heading of **timer management**. Timer management includes a broad class of user services, such as:

- provide "watchdog" or monitoring timers for systems processes and expected events
- provide suspension of a process for a specified interval
- provide suspension of a process until a specific time of day
- provide for the startup of a process at a specific time of day
- provide for timing of expected events and asynchronous execution of procedures upon expiration of the interval (timeouts)
- provide for cancellation of previous timer requests

Many of these timer services go beyond a simple clock device driver. Complex multiprogramming operating systems can have hundreds of timer service requests pending at any instant in time. Such systems require the management of a timer queue, just as process management requires the management of process scheduling queues.

Timer Queues

While it is possible to have multiple physical clocks, it would never be possible to have enough clocks to assign a separate one to each timing request generated by an operating system and the processes in the system. Moreover, it may be difficult to keep multiple physical clocks perfectly synchronized. A much better solution is to establish some form of multiple **virtual clocks**.

One way to do this is to keep a linked list of elements containing the time in clock ticks (or perhaps microseconds) representing timer requests, associated with an action to be performed

-

when that time occurs. Such a list, ordered by increasing time of request, is called a **timer queue**. Each element of the timer queue is called a **timer queue element**. Figure 8-1 illustrates a timer queue with timer queue elements containing these time-of-day values.

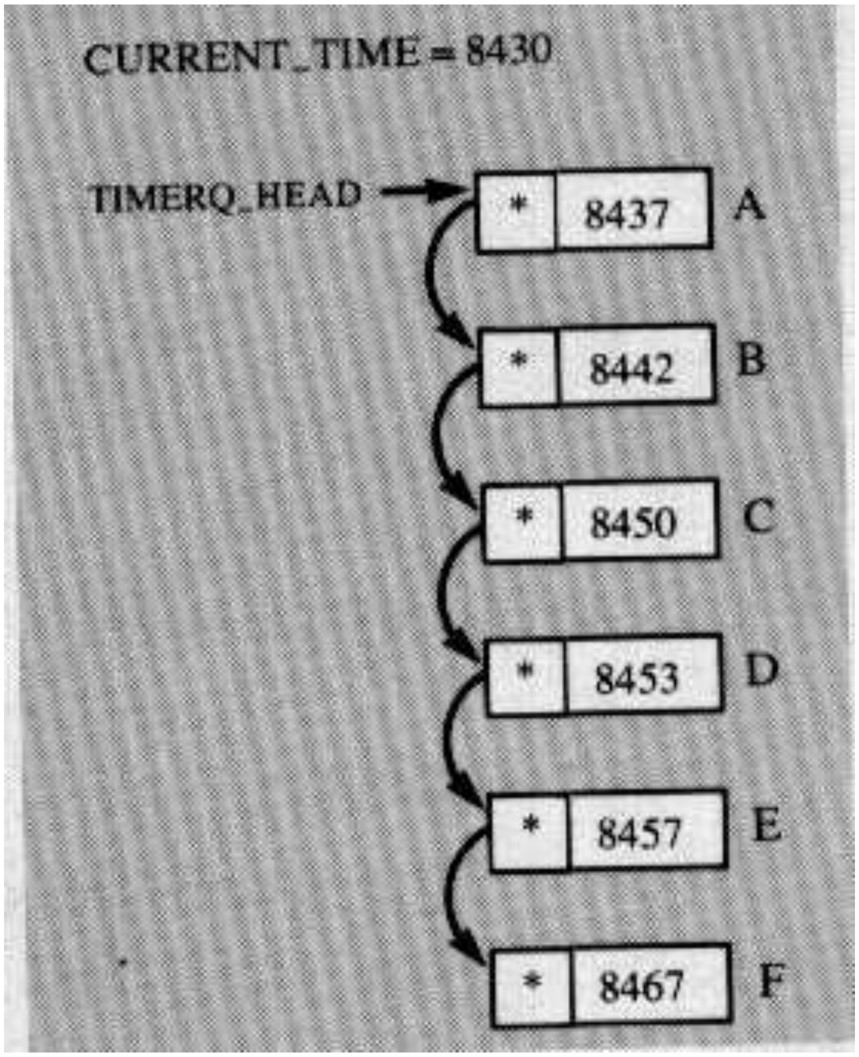

Figure 8-1: A Timer Queue using Time-of-Day Units

Every time the real time is updated (by the clock interrupt handler), the timer queue should be checked to see if the time specified for the next activity has arrived. If so, the appropriate actions specified by the timer queue element must be taken by the operating system. These actions include such things as resuming a suspended process, removing CPU control from a process, or triggering the execution of a timeout routine. A second-level interrupt handler could be scheduled to search the timer queue while still permitting the system time-of-day clock to be updated by the clock interrupt handler. A possible algorithm for checking the timer queue is:

```
Timer_Search: Set P to Top_TQE
DO WHILE Time_Of_Day_Clock
        > P->TQE_TOD
    schedule action of P->TQE
    set P to next TQE
ENDWHILE
set Top_TQE pointer to
    TQE pointed to by P
RETURN
```

The interrupt handler overhead required for processing timer queue elements is potentially high. This can be reduced by storing clock intervals, or time quanta, between successive timer queue element requests. In this case, a particular timer request represented by a timer queue element is for a total time interval, equal to the sum of the quanta in this element and all elements that precede it in the list. For example, if the current time in clock ticks is 8430, as shown in Figure 8-2, the timer queue element D is for a timer request that will occur after 23 clock ticks (the sum of 7+5+8+3, i.e., clock ticks in elements A through D) or at real-time 8430+23=8453, the same as it was for element D in Figure 8-1.

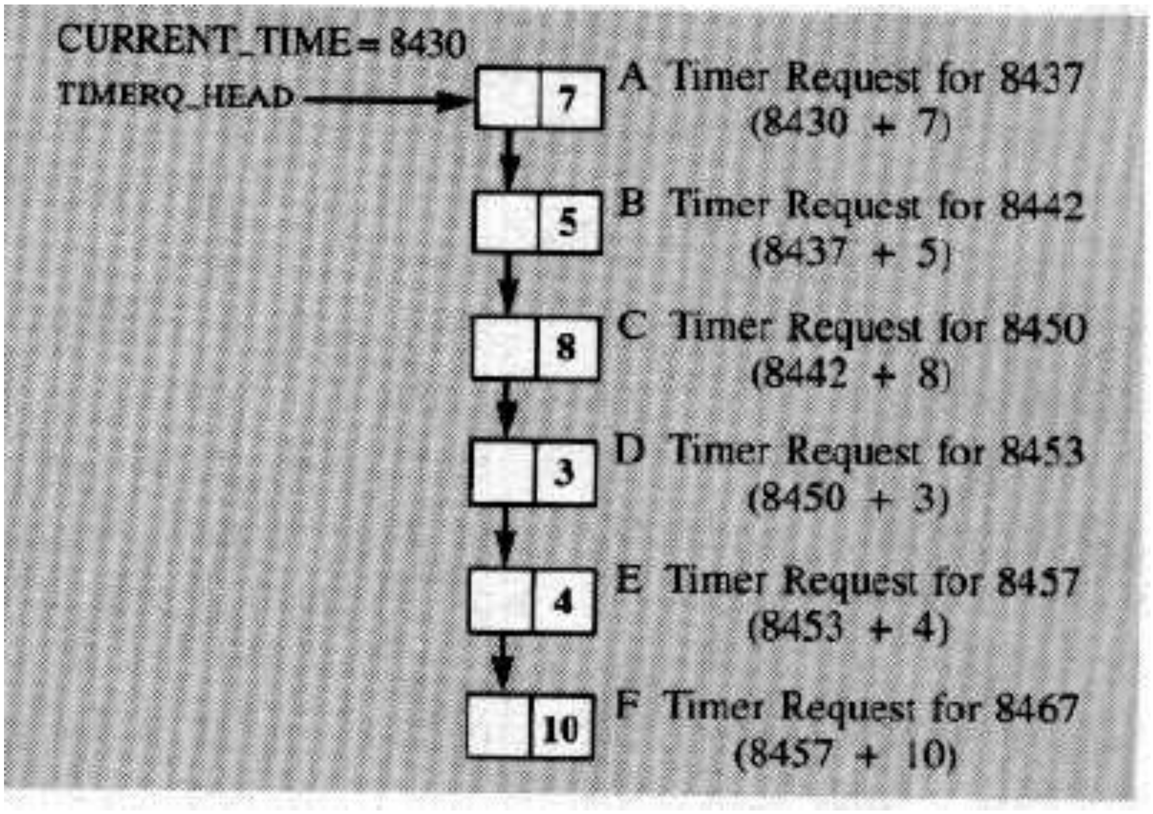
CURRENT_TIME = 8430
TIMERQ_HEAD
7
A Timer Request for 8437 (8430 + 7)
5
B Timer Request for 8442 (8437 + 5)
8
C Timer Request for 8450 (8442 + 8)
3
D Timer Request for 8453 (8450 + 3)
4
E Timer Request for 8457 (8453 + 4)
10
F Timer Request for 8467 (8457 + 10)

Figure 8-2: A Timer Queue using the Sum of Time Quanta

In addition to pointers to the previous and next timer queue element and the time quantum, contents of a timer queue element may include:

- PCB of requesting process
- address of procedure to call
- type of request (user or operating system)

Figure 8-3 illustrates a possible structure for a timer queue element.

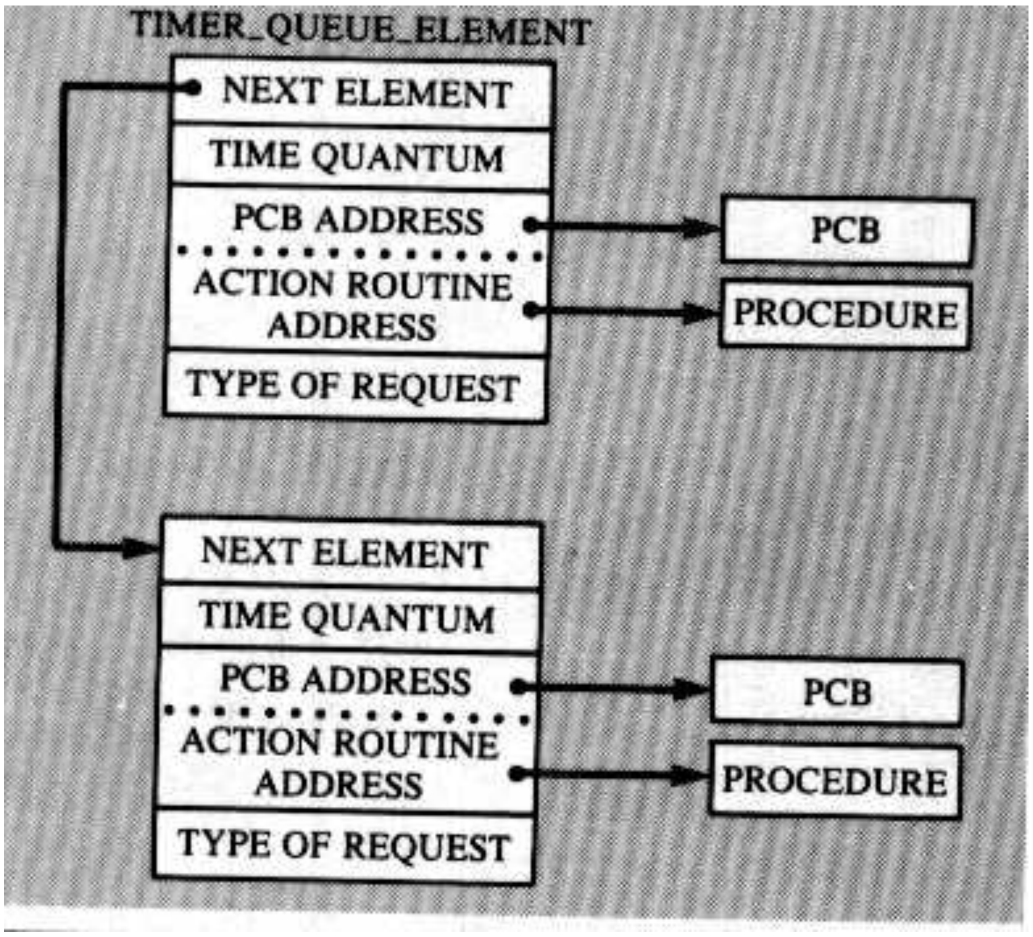

Figure 8-3: Structure of a Timer Queue Element

8.5 TIMING SERVICES

A variety of services may be provided at the program interface by systems that support time management. Common examples are outlined in this section.

Maintaining the Time of Day. As has been discussed, time-of-day maintenance requires constant update of the time-of-day clock and the ability to convert from clock ticks to hours,

-

minutes and seconds. It also requires that the date be advanced when midnight occurs. This could be done by simply having a system timer request in the timer queue that expires at midnight, invoking a portion of the clock routine that appropriately updates the time-of-day clock and advances the date.

Allocating Private Timers. Private, logical timers may be structured as logical resources to be allocated to individual processes. This type of mechanism is provided by the Berkeley versions of Unix. Typical operations supported for private timers would include allocate and free; read and set values; and start and stop running. In addition, a procedure may be specified to be executed when the timer reaches a particular value such as zero.

Monitoring Systems Processes. Operating systems often require a timer request to monitor part of their own operation. For example, if an input device is started for a read, and it is possible under certain conditions for there to be no response, it is necessary for the operating system to request a timeout interval be set, so that when it expires, the appropriate error action can be invoked in the device driver. If data communications is part of the operating systems software, timeouts for receiving data are another example of the need for alerting the operating system if expected events fail to occur.

Process Suspension for a Specific Time Interval. Suspension of a user or system process for a specific time interval, by means of a sleep or suspend system call, is often required. This is the type of request that may be used, for example, to process an Ada DELAY statement. To handle such a request, the OS must:

- suspend the process by changing its status in the PCB
- resume the process by changing its status in the PCB upon the expiration of the requested time interval

Process Suspension until a Specific Time of Day. Suspension of a process until a specific time of day is similar to suspension for a specified interval of time, except that the user provides the real time as the parameter. The action taken would only differ by how the timer queue element is created.

Startup of a Process at a Specific Time of Day. Many operating systems, particularly real-time systems, provide the facility to start a process in the future at a specific time of day. One possibility for implementation is to create the process when the request is made, but suspend it until the specific time of day; however, this could tie up valuable memory resources, and might not be the most effective method of implementation. Another technique would be to have a small startup process for each process to be started in the future, which is suspended until the startup time. Figure 8-4 illustrates this technique.

Timing of Events and Invoking Timeout Procedures. User and system processes often require specific actions to be taken upon the expiration of a timer request. This timer service request would have to specify the timer interval or time of day and the procedure (action routine) to be called upon the signaling of this timeout.

-

Cancellation of Pending Timer Requests. It is always possible that a particular timer service request will be canceled. Normally, an application process would only be permitted to cancel a timer service request that it issued. Since each timer queue element will contain a pointer to the PCB of the requesting process, this can be easily verified. The appropriate timer queue element must be removed from the timer queue and the timer queue entry following the one to be removed must be updated by adding the time quantum in the deleted element to the next element.

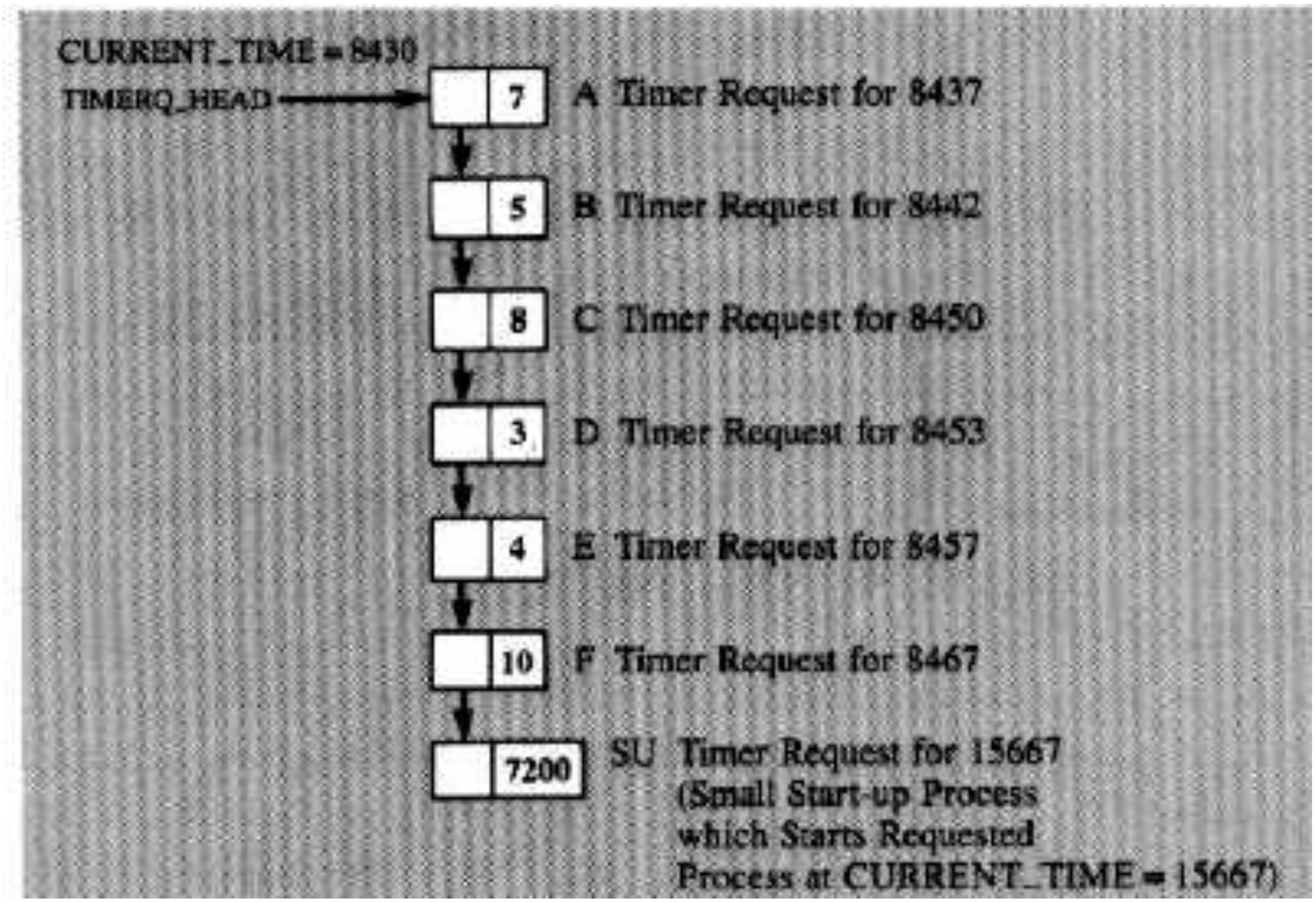

Figure 8-4: Starting a Process at a Specific Time of Day

FOR FURTHER READING

Timing mechanisms are not extensively discussed in most general OS texts. However, a useful discussion of clock and timer management is contained in Tanenbaum [1987, Sec. 3.7]. The topic is treated briefly in Silberscatz et al [2002, Sec. 13.3.3]

Timing services in VMS are described by Sewell [1992]. UNIX timing services are covered by Robbins & Robbins [2003]. Linux timing issues are covered by Rubini & Corbet [2001].

-

REVIEW QUESTIONS

1. Outline the types of timer services a programmer would expect to find in a multiprogramming operating system.
2. How is a timer queue element used? Illustrate.
3. What effect would the masking of timer interrupts by other interrupt handlers have on device and timer management?
4. If timer management uses a programmable clock, and a timer request for a time prior to the expiration of the current interval in the physical clock is made by a process, how can it be serviced (since the clock already contains a value greater than the one necessary for this request)?

ASSIGNMENTS

1. Under what conditions would a program cancel a pending timer request to invoke a specific procedure upon the expiration of the requested timer interval?

PROJECT

This section describes a three-part programming project to implement a set of timing mechanisms for a simple multiprogramming operating system. These mechanisms are optional components of the multiprogramming executive (MPX) project .

a. Implement a clock interrupt handler that maintains the number of clock ticks since midnight. The time-of-day clock that maintains clock ticks should be called CLOCK.

b. Using the clock interrupt handler developed in Part a., implement a set of procedures within the MPX scheduler/dispatcher which may be called by test processes to make use of clock services. The operation of these procedures is as follows:

   1. `Start_Clock:` Start the operation of CLOCK, so that subsequent interrupts by the hardware clock will affect its value.
   2. `Stop_Clock:` Stop operation of CLOCK, so that further interrupts will have no effect on its value.
   3. `Set_Clock(val):` Set the value of CLOCK to val. The value is expressed in clock ticks past midnight. Note that this does not affect whether the clock is started or stopped.

-

4 `Read_Clock(val):` Read the value of CLOCK into the variable val.

c. An operating system that requires users to think in clock ticks is not particularly user friendly. Refine the clock support to provide additional procedures and data structures which:

1. Set the present date
2. Read the date
3. Set the clock in hours, minutes, seconds
4. Read the clock in hours, minutes, seconds

-