13.8 Delta List Implementation

An overarching goal of operating systems designers arises from the desire to achieve maximal functionality with minimal mechanism. Designs that provide powerful functionality with minimal overhead are valued. To achieve such goals, designers look for ways to create underlying mechanisms that accommodate multiple functions. In the case of delta lists, we will see that it is possible to use the basic list data structure covered in Chapter 4. That is, the delta list of delayed processes will reside in the *queuetab* structure, just like other lists of processes.

Conceptually, the processing required for a delta list is straightforward. Global variable *sleepq* contains the queue ID of the delta list for sleeping processes. On each clock tick, the clock interrupt handler examines the queue of sleeping processes, and decrements the key on the first item if the queue is nonempty. If the key reaches zero, the delay has expired and the process must be awakened. To awaken a process, the clock handler calls function *wakeup*.

Functions to manipulate a delta list seem straightforward, but the implementation can be tricky. Therefore, a programmer must pay close attention to details. Function *insertd* takes three arguments: a process ID, *pid*, a queue ID, *q*, and a delay given by argument *key*. *Insertd* finds the location on the delta list where the new process should be inserted and links the process into the list. In the code, variable *next* scans the delta list searching for the place to insert the new process. File *insertd.c* contains the code.

Observe that the initial value of argument *key* specifies a delay relative to the current time. Thus, argument *key* can be compared to the key in the first item on the delta list. However, successive keys in the delta list specify delays relative to their predecessor. Thus, the key in successive nodes on the list cannot be compared directly to the value of argument *key*. To keep the delays comparable, *insertd* subtracts the relative delays from *key* as the search proceeds, maintaining the following invariant:

> *At any time during the search, both key and queuetab[next].qkey specify a delay relative to the time at which the predecessor of "next" awakens.*

Although *insertd* checks for the tail of the list explicitly during the search, the test could be removed without affecting the execution. To understand why, recall that the key value in the tail of a list is assumed to be greater than any key being inserted. As long as the assertion holds, the loop will terminate once the tail has been reached. Because *insertd* does not check its argument, keeping the test provides a safety check.

After it has identified a location on the list where the relative delay of the item being inserted is smaller than the relative delay of an item on the list, *insertd* links the new item into the list. *Insertd* must also subtract the extra delay that the new item introduces from the delay of the rest of the list. To do so, *insertd* decrements the key in the next item on the list by the key value being inserted. The subtraction is guaranteed