Clkhandler begins by decrementing local variable *count1000*, which counts from 1000 down to 0. When *count1000* reaches zero, one second (i.e., 1000 ms.) has elapsed, so *clkhandler* increments global variable *clktime*, which stores the time in seconds since the system booted. *Clktime* is used to provide the date (e.g., it is used by the Xinu shell command *date*).

Once it has handled incrementing global counters, *clkhandler* performs two tasks related to processes: sleeping processes and time slicing. To manage sleeping processes, *clkhandler* decrements the time remaining on the first process in *sleepq* (provided *sleepq* is nonempty). If the remaining delay reaches zero, *clkhandler* calls *wakeup*, which removes all processes from the sleep queue that have a zero delay. As we have seen, *wakeup* makes the processes *ready*. Finally, *clkhandler* decrements the preemption counter, calling *resched* if the preemption counter reaches zero.

13.13 Clock Initialization

Clock initialization can be divided into two conceptual parts: initialization related to the operating system and initialization related to the underlying clock hardware. In terms of the operating system, the clock initialization code performs three steps. First, it allocates a queue to hold the delta list of sleeping processes, and stores the queue ID in global variable *sleepq*. Second, it initializes the preemption counter, *preempt* to *QUANTUM*. Third, the code initializes global variable *clktime*, which gives the seconds since the system booted, to zero.

In terms of clock hardware initialization, the details vary widely among platforms. On the simplest systems, the clock hardware is completely preconfigured — both the interrupt vector and clock rate are hardwired. On most systems, the hardware is parameterized, which means the operating system can control the rate at which interrupts are generated. The operating system, may also be able to assign an interrupt vector.

Both the Galileo and BeagleBone Black platforms have configurable clock hardware. On the Galileo, the operating system must assign an interrupt vector, set the mode of the clock, and set the exact clock rate. Our code calls *set_evec* to configure *clkdisp* to be the function that receives interrupts. It then uses an *outb* call to configure the clock as a 16-bit timer. Finally, our code uses two calls of *outb* to configure the 16-bit rate register to 1193, which will cause the clock to interrupt once per millisecond. The clock initialization code for the Galileo can be found in file *clkinit.c*.