THE SYSTEM CLOCK

This is a very short chapter. There are several reasons for the brevity, but perhaps the most significant is that the clock is not a part of the operating system at all. We include it here because we have already mentioned the clock twice, and because we shall want to say more about it in the next chapter. The clock is an essential component of a system which must meet any form of time constraint, but if it is to work properly it must be quite independent of the rest of the system – another, this time symbolic, reason for the separate chapter.

Most processors with any pretensions to greatness incorporate a system clock. This is an electronic circuit which counts signals from the processor clock, an independent oscillator, or some other reliable source, and, among other things, causes a processor interrupt every now and then. Its most important feature is that it keeps running no matter what the processor is doing. Indeed, it is not unreasonable to think of it as a second processor, with the unusual characteristic of not being able to execute any process except measuring the elapsed time. Its importance comes from the principle we have emphasised from time to time right from the beginning : that a processor, once running, can either execute code or break down. If it's broken down, we've lost control anyway; if it's executing code, it will go on executing its current code stream until it has to stop, which, in a (deliberate or accidental) tight loop, could be for ever. If we want to ensure that this won't happen, we must provide an independent entity in the system which can take action to divert the processor to some other code from time to time.

This is the function of the clock, and it is (necessarily ?) implemented by providing a clock interrupt. This is emitted by the clock from time to time, and, in the usual way of interrupts, redirects the processor to some interrupt-handling code belonging to the system. This gives the system an opportunity to decide what to do next; if all is well, it can return to the interrupted code and carry on, or there might be reasons for suspending that activity for a while and attending to some other code. It therefore solves the problem mentioned in the *DISPATCHERS* chapter of ejecting a process from the processor; it gives an opportunity to attend to any tasks which must be performed periodically; and – particularly important for process management – it is the basis of a method for giving different processes a fair share of the processor. When a process is dispatched, it can now expect only a limited clear run on the processor; this is commonly called the process's *time-slice*.

There is only one more thing to say about the clock. We have said that the clock emits an interrupt "from time to time"; what determines the time ? The answer is that either the time interval between interrupts is fixed by the hardware, or that the interval can be set from the processor. In practice it makes little difference; if you want a fixed interrupt frequency with a variable clock, you don't vary it, while if you want a variable interrupt frequency with a fixed clock you write your interrupt-handling software to count interrupts before passing them on to the system. That's a little more overhead from the system point of view, so the variable clock is to be preferred in principle, but the difference is small.

COMPARE :

Silberschatz and Galvin[INT4] : Section 2.5.4.
