ACTUALLY RUNNING PROGRAMMES

At last, all is prepared. We can tell our system what to do, it can provide memory space and file services, and, all being well, can do so safely. We are ready to run a programme.

And that is a rather curious statement, because in order to attend to the keyboard, screen, memory, disc, and security the computer must necessarily run a programme. So what's new ?

What's new is that we are now ready to run *any* programme, not just the operating system. We can at last do some real work for someone who wants it, and thereby discharge our basic duty as a computer. Indeed, in the discussions which follow it is often convenient to practise a little sleight of mind and forget that the operating system is a programme. That's because the operating system kernel, if not its other parts, is not subject to the same sorts of discipline as are "real" programmes; in effect, they are part of "the system". (Proof : The dispatcher is the part of the system which decides which process should be run on a vacant processor - see the chapter *DISPATCHERS.* If the dispatcher were a process like any other, how could it ever be executed ?) One of the consequences of this special nature has already come to our notice. We saw back in Chapter 1 that it was necessary to invent a *system call* to permit ordinary programmes to communicate safely with the operating system.

The central focus of this section is therefore what happens when we do some real work for someone. We shall have to take account of the real nature of the operating system procedures from time to time – most obviously, when they require to use a processor – but frequently they don't interfere with the argument, and we can just assume that they're there, working away quietly.

What happens, then, when a programme runs ? At the simplest level, the answer is obvious, and we gave it earlier : the programme is presented to an interpreter, which can understand its code and follow the instructions which it contains. That's a general (and true) statement - but now we're concerned with making things work on a computer, so we must direct our attention first of all to the interpreter provided - the processor. In these terms, the answer becomes : a processor's programme counter contains the address of some code in memory, and the processor is left to run. So far as we know, this description applies to all current practical computers which need operating systems. It applies, with minor bends, to other sorts of computer architecture which are or have been research topics from time to time, such as dataflow and reduction machines; it might not be flexible enough to encompass neural networks, but it will be some time before we need operating systems for them. We shall therefore accept the description henceforth.

In this section, we explore how to make this real work happen as we want it to happen. We have already said quite a lot about that in earlier sections. Some examples : we might want to use many processors; we might want to run many activities "at once" in multiprogramming; sometimes activities have to wait for certain resources to become available before they can continue. These are just some of the complications which we want to support with our simple model of a running programme. How can we do it ? Read on ....