PROGRAMMES

What's a programme ? If you don't know, you must have missed quite a few interesting points in these notes, because we have not felt constrained to avoid mentioning programmes even though we haven't defined them.

Now, though, we have a new viewpoint – a programme is how we describe the behaviour we want in a process. In one way or another, which is not our concern, it is a collection of instructions for some processor, and the correct processor must be used to execute the instructions. Thus, a machine language programme requires the hardware processor, a code file for a virtual machine requires a software "machine" (such as the Java virtual machine), a command file requires a command line interpreter, and so on. All these are programmes; the differences don't matter, provided that the programme can be understood and executed.

Notice that the process structure is recursive; the processor which runs a process for a Java script is the Java Virtual Machine, but that is itself implemented by a process which runs machine language on the hardware processor. This isn't quite the same as the conventional view of processes in an operating system; that's because we haven't got there yet. These processes are design abstractions, and we begin to discuss implementation in the HOW TO DO IT section.

Physically, a programme is a collection of instructions, and, like any other collection of information, it is most commonly stored as a file. We already know about files; the only remarkable fact about a programme file is that in some circumstances it can be executed. Programmes can also be provided in ROM – most modern computers contain a significant amount of ROM code, often used to start up the system or to implement frequently used functions. Yet another source for a programme is a data stream; the instructions you enter at a keyboard during a computer session can be regarded as a programme of this sort.

Why do we care what a programme is ? A common attitude is that the operating system's job so far as a programme is concerned is to set it running, and what it does thereafter is its own business. We argue that this at best an oversimplification, and at worst could lead to less than optimal service from the operating system. It is certainly true that as long as we wish to build a general-purpose operating system there should be no constraint on what the programmes do; but the operating system is already deeply involved in coping with the requirements of running programmes, and – as we know from the example of the batch systems – the more we know about the properties and behaviour of these requirements, the more likely we are to be able to build systems which efficiently supply the services needed.

More precisely, we want to know the right sorts of properties and behaviour. In one sense, we already know all about the programme, because we have the code, and all the behaviour follows from that (and from the data, but that doesn't change the argument). In practice, that's no use, because we haven't time to work out the behaviour in time to use it while the programme is running. What we need is realistic information about patterns of behaviour which the programme is likely to exhibit as it runs. We might gain some information of that sort by examining the sorts of demand for services which programmes make.

WHAT DO PROGRAMMES WANT ?

Programmes make many sorts of demand on an operating system. In fact, that's a truism, for a major reason for the existence of an operating system, and pretty well the sole reason for the invention of operating systems, was to provide services to programmes. Many of the demands are made by the programme on behalf of something else, such as a file or a terminal display; we usually deal with such a demand under the heading of the something else in question. Here, we are interested more in the demands made by the programme on its own behalf, and we think that there are just two of these : the demand for a processor to carry out the instructions, and the demand for storage space for use in the computation. (Let us know if you find any more !)

The first is obvious; nothing will happen unless the instructions are read by a processor of some sort. In principle, bearing in mind our earlier remarks about the possible variety of processors, a collection of instructions could record the identity of its processor, so that the system could select one of the correct type. It is more common, though less general, to regard only the hardware as the real processor, and to treat everything else in a different way, but in principle different sorts of processor could be implemented. This monopoly is beginning to slip a little as operating systems broaden their area of control from a single processor (or a few essentially identical processors) to a distributed system in which there might be significantly different processor types. If there are advantages to be gained by selecting a specific processor type for a programme, or perhaps by moving it from one processor to another, we would like our operating system to be able to exploit these possibilities. For the moment, though, such systems are uncommon.

The second requirement of a programme is much more common and much better understood; this is the use of storage space during the computation. Different sorts of computation exhibit different sorts of demand for storage, partly because of the data structures which they use and partly because of the programme structures imposed by the algorithms chosen. The programming language used affects both these factors, and to a great extent control the *memory model* used in the computation. This in turn gives a strong indication of how a programme will use the available storage. If we can identify common patterns of storage requirements, we can construct our operating system to cater efficiently for these patterns, thereby contributing to the smooth running of the system as a whole.

Observe that this new interest in the use of storage is not quite the same as that which earlier caused us to investigate the notion of files. Then, we were interested simply in providing some means of storing information, perhaps for considerable lengths of time, and our aim was to ensure that this requirement was satisfied; now, we are more interested in knowing how stored information is used by programmes, perhaps only for short time periods, and our aim is to identify facilities which might assist programmes to operate more efficiently. The two topics are obviously related, but the emphasis is different.