PROCESSES

We need processes, as we said earlier, to talk about the behaviour of computers, and particularly to talk about the behaviour with precision. If you just want to describe what computers do in general terms, you can almost always get away with descriptions in terms of running programmes. Processes become necessary when you wish to give each activity a name so that you can analyse the progress of each, investigate their interactions, and so on. It is then important to have a way of identifying a continuing activity which might involve several programmes, or which might share the same programme with other similar activities, which might have periods of inaction between its periods of activity, and which might even move from one computer to another as it proceeds, but nevertheless retains its identity throughout.

Processes are abstract entities. We shall later use the word in a much more concrete sense (because that's the current practice), but for the moment we are still designing.

AN ANALOGY ?

If you wish to bake a cake, you will require data (flour, eggs, etc. – recall that data means "things which are given"), hardware (baking tins, oven, etc.), instructions (a recipe), and a processor (more usually called a baker in this context – probably you, unless you have affluence or influence), and you hope that you will eventually acquire a result (the cake). The analogy with the description above is quite good. But you won't get a cake without some activity : the baker follows the instructions, which determine how the materials and hardware are used, and should eventually result in the appearance of the product. The activity is the process.

WHAT PROCESSES DO.

Processes start and stop. They don't simply materialise from nothing; they are not primitive entities, and have to be built up. At the very least, they must involve a set of instructions (the programme) and some means of executing the instructions (a processor), and these two must be brought together by some agency. (Someone has to give the recipe to the baker; the recipe might then specify the other details – materials, hardware, etc. – and a sufficiently capable baker can go and get those, but the recipe must be provided.) We've also seen that the programme, when it runs, requires areas of storage which we called segments; the segment table which we invented finds a natural home as part of the process structure. (Perhaps the baker must reserve space in the oven ?)

What is it that brings together the programme and the processor ? It's an activity of some sort – so if we take the view that all activities in our systems are carried out by processes, one process must be started by another. In our abstract discussions at least, we shall indeed suppose that processes do all the work, so we necessarily conclude that a process must be built into each system from the start if it is ever to do anything at all.

A process might pause; the baker might find that there are no eggs, and suspend the cake process until some are available. At this point, the baker will doubtless find other things to do (which is to say, start executing another programme); what has happened to the cake process ? It is not enough just to sever the link between processor and programme, for to do so might lose information about the process (the baker might forget which of the programme instructions to execute next) or other structures built up by the process (perhaps there is a bowl with some flour in it awaiting attention). It is clearly necessary to have some means to record this sort of information about a process – so the process acquires its own data structure to link its parts together.

Processes are likely to communicate with each other; whoever brings the eggs might tell the original baker – or, in principle, any baker who can make sense of the cake process's data structure – that eggs are now available, and the cake process might then be restarted. Don't worry too much about which processes are involved in this transaction; one might guess that the communication is between the operating systems of the egg deliverer and the baker, but the principle is the same anyway.

Processes can form hierarchies; the baker, continuing with the cake process, might use a machine to mix the ingredients. This machine is another processor (indeed, it might well be called a food processor), with its own very simple communications (a switch) and programme (a motor). While the machine is doing the mixing, the baker might be preparing the next step of the cake process – so at this point the original process has split into two subprocesses, which are being executed by quite different processors.

And so on. These examples all have counterparts in computer systems, and by no means exhaust the possibilities. In each case, though, there is a programme and a processor, and there is some information about the process (admittedly vestigial in the case of the food processor). Throughout the whole story, though, there is a continuing thread of activity, and it is clear how the different steps are related to each other through the cake process. It is important that this is still true if we put the cake process in the context of the whole activity of the bakery, in which many other processes will be in action at the same time, with many processors, human and mechanical, also involved. The notion of the process serves to distinguish the particular set of activities connected with the cake, and to keep them separate from other processes executed for other purposes.

We have rather laboured this analogy because we think it's very important to grasp the idea of a process, and to understand both its relationship to programmes, processors, and so on, and the variety of activities in which it can take part – execute, wait, split into two, communicate, etc. It's also useful to bear in mind that processes are not only to be found in computers.

HOW CAN PROCESSES BE MANAGED ?

The processes are the means by which the useful computational work gets done; it is the task of the operating system to run the computer in an effective manner. The operating system must therefore control the processes in some way, and – as we saw earlier in this chapter – it will require structures through which processes can be identified, and functions which will perform such operations on processes as are needed.

The operating system must therefore be provided with some means of identifying and manipulating the processes; this argument is quite similar to that which led us to invent the file table, though in this case we are thinking of the convenience more of the operating system than of the person using the system. That suggests that we shall have a process table, in which all the processes are somehow represented, and through which the operating system can exercise its control.

We are conscious that in leading the discussion towards process tables we might be influenced by our knowledge of current practice, in which some form of process table is almost universal. Such prejudice is not what we want in our attempt at disinterested top-down analysis, but we offer two not-quite-arguments which support the suggestion. First, we point out that we have not imposed any special structure on the table; it might be a compact table, or a linked list, or a tree, or a circular list, or whatever; a table, in the sense we intend, is a representation of each of the entities (in this case, processes) tabulated, and some defined means for finding them when required. Second, we cannot think of another way to do it which seems likely to be of general application; given that we don't know how many processes we might want, and that the operating system must be able to find all of them if it is to perform its management function satisfactorily, there is little alternative. Let us know if you can think of one.

Assuming the necessity of a table, then, we can next ask what will be in the table. Many details are likely to depend on implementation requirements, and we shall defer these until later, but there are a few items which seem unavoidable. These are the data which tie the process together. We have seen that a process is associated with a programme, and some storage for computation; it would not be surprising to find some sort of link from the process's representation in the process table to the programme and storage concerned. An alternative is to label the programme and storage with the identity of the process, and search for these labels (either in the complete store or in other tables for programmes and storage) when required, but the direct link is almost certainly preferable, as we have already seen that several processes could share a programme, and that different programmes might be associated with a process during its life.

We have also seen that a process can engage in various sorts of activity; as we pointed out earlier, it will be important to record what each process is doing so that the operating system can deal with it accordingly. This is more information for the process structure. Clearly, we can identify several properties of processes which we wish to tabulate, so we shall expect our operating system to incorporate some sort of process table. This can join the file table and the user table as another important structure underlying the operation of the system as a whole. We shall discover more tables as we continue our studies – simply because they are very effective means of presenting information about a lot of entities of much the same type in an orderly and accessible manner.
