WORKING NOTE AC113

Alan Creak
25 August 1997

INTRODUCING PROCESSES IN A COURSE ON OPERATING SYSTEMS

This discussion was recorded on or before 8th June 1996, and it went to Robert Sheehan by electronic mail on 10th June. I'd expected that long before now it would have been incorporated into some other document, but that hasn't happened. I'm therefore preserving it in semi-permanent form, with a few minor amendments to make it reasonably self-contained, before I lose it.

AN AWKWARD TRANSITION.

I have for a long time been unhappy about the tangle we get into in our operating systems course when we go from design to implementation. Suddenly we have to talk about memory and processors and processes all at once, and it's difficult to sort it out into a satisfactory order. We have explained, or perhaps attempted to excuse, this in the course by vague comments about getting very close to the real hardware in the design process, and therefore inevitably mixing the two. There is some substance in that claim, but not much, and the position has been unsatisfactory.

In 1995 I thought that I might be approaching a way to resolve the problem. Some glimmerings of routes through the morass became evident, mainly through a determination to follow the path of careful design to the limit before embarking on a discussion of implementation. I have now convinced myself that the problem is indeed no more, and this note is a record of my solution. I have tried to present it as an argument in a series of steps; I am not sure how successful I have been, but most of the points of interest should be detectable in there somewhere given a little goodwill. (I remark now – in 1997 – that I find myself less impressed by this development of the argument than I must have been in 1996. I have preserved the details for historical accuracy; and I still agree with the eventual conclusion.)

1 : The aim is (always) "to get work done as instructed". We identify that near the beginning of the course as the essential task of computers, and therefore of operating systems.

2 : Therefore, we must have some work to do, and some action must be taken by something while doing it.

3 : Work is defined as following instructions, which are expressed in some way and must be represented in the computer in some form which it can use. There might also be data and results, which, though their functions are different, must also have representations of some sort in the machine.

4 : But work does not always require data and results, at least in the sense of information recorded in files. Work might be behaviour. Consider a computer set up as a teaching machine; there is nothing directly corresponding to the traditional idea of "results". In this case, the "result" is the way in which the machine interacts with the pupil. There are no data in the traditional sense of the word, where it implies the special information you want to give to the computer for this run as opposed to the programme which is always the same; instead, the special information comes through the interface from the pupil, and the programme must deal with it as it arrives. (If you are not enthusiastic about teaching machines, think about games.)

5 : And even if we do want results, the computer must exhibit some behaviour; it must compute.

THEREFORE :

6 : To talk about what computers do, we must develop ways of talking about how we can deal with information in the computer, and how the computers behave.

7 : To deal with information, we invent *files* (for information that's standing still) and *streams* (for information in motion).

8 : To talk about the behaviour of computers, we invent *processes*. We have to be careful how we define processes, because there are many things which they are not. For example, they are not programmes; they are not results; they are not computers in action. A process is an abstraction of the notion of "doing work as instructed" which we have emphasised throughout. It requires instructions; it requires some agent capable of doing the work; it might require certain information, and it should produce work done, which might be embodied in results – but it is none of these. It is an activity – the performance of the instructions by the agent.

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.

IMPLICATIONS.

The significant change is that we now have a rather clear boundary between design and implementation. Once we have designed the files and the processes, the job is done. What's left is implementation.

In terms of the course notes, the stuff on "pure" processes – that is, without implementation – comes into the design section. Here we can discuss what we want processes to do – obvious simple things, interact in various ways, split into threads, etc.

All the material about memory and the processor follows. Memory is an implementation device to make the file data more readily accessible – ideally it should be a file cache. Why isn't it ? The processor is an implementation device to make the processes work. Memory management and process management are necessary to build a practical system.

After that, we're more or less back to the current course structure.

COMMENTS.

Perhaps one could see the change as something of a move back towards the traditional model : the treatment of processes has moved forward a bit, and is now closer to the beginning of the course. But the difference is that, having worked through the design operation, we know rather well just what the processes are, and how we want them to behave, *before* we start to discuss their implementation.