MAKING LIFE EASIER

Once the basic machinery of the operating system does everything we want it to, we find new things to want. Are we such pessimists as to believe that it would never do to be satisfied ? We can't answer that general question, but in the context of operating systems we'd rather view the continuing demand for additional facilities as a manifestation of the constant push towards increased efficiency which we followed in the *INTRODUCTION* section.

Be that as it may, it is certainly true that many features of modern operating systems contribute little or nothing to the set of operations which can be carried out by the system (its "functionality", if you must), but rather provide different ways to achieve certain ends which in some circumstances can make it easier to get work done.

We have no intention of describing all such features : there are far too many, and the boundaries of the set are not at all well defined – one system's special feature is another's standard component. Instead, we draw attention to the phenomenon by describing a few techniques which strike us as interesting.

COMMAND FILES.

We've already commented on command files, and do so at greater length later, but we include them here because they are a good example of the phenomenon. They stand in the same relation to operating systems as programmes do to computers. A computer which can't run programmes is a calculator; each instruction must be entered individually. The ability to execute programmes doesn't increase the repertoire of data processing operations of the processor, but it makes a great difference to its utility !

SESSION LOGS.

Session logs also appear elsewhere in these pages. They share with command files a deceptive simplicity – all they do is automate the obvious, and laborious, manual process of writing down all that transpires in a terminal session, so that a permanent record of events is available in case it's needed. The automatic log saves a lot of work, and also ensures that nothing is missed in fast sequences. That's particularly valuable if it can capture the last two lines of system output displayed before the system crash blanks the screen !

The "deceptive" part of the simplicity is connected with the multiple functions of the screen. The (comparatively) orderly display of material on the screen is the consequences of operations carried out by several entities in the system; it might include material echoed from the keyboard input, material from the system, and material from other executing programmes. In a simple system, there is little need to coordinate the activities of these components, and each can be allowed to use the screen independently of the others. If a log is required, though, the separate streams of information must be collected, so once again additional operating system machinery is required.

FILE POINTERS.

We include file pointers in our list to give an example of a feature not directly connected with input or output. A file pointer is a reference to a file (or other similar entity) which might be more complex than a conventional file directory entry. They are typically used to provide different ways of naming files, so the same file can appear in different system file directories if it's convenient. The Unix system has provided file *links* for this purpose for a long time; more recently, the Macintosh system has introduced file *aliases*.

The complication in this case arises because a file directory entry usually contains information about the file which must be kept consistent with the real state of the file as it is stored on the disc. Simply duplicating the directory information wouldn't work, as the consequences of manipulations on the file using one of its pointers wouldn't appear in the directory entries for the other pointers. It would also be possible to use two pointers to the same file simultaneously without knowing, which could cause all manner of difficulties.

Some coordination is therefore necessary. Just what is required depends on what you want to do with the pointers. For example, if you want each to be indistinguishable from a conventional file directory entry, you can't choose one of them (such as the first to be made) as the master item, for then you couldn't just delete the first instance without affecting the others. We'll discuss this question further when we deal with the file system in *DISC FILE SYSTEMS*.

COMMENT.

The position of many of these features with respect to the operating system is by no means clear. Historically, many first appeared as quite independent utilities; later, they might have been absorbed into the system; later still, the system shrank to a microkernel and the distinction between utilities and system components all but vanished. With the disappearance of monolithic operating systems and the greatly improved programme interfaces, it is probably easier now than ever before to integrate new components into an operating system. Whether or not you consider them to be "part of" the operating system is a matter of definition; we take the view (consistent with the dustbin model we introduced in the *INTRODUCTION* section) that anything helping you to run the programme which really does the useful work is operating system.

Which is all very well, but it is also *more* operating system. Adding features ad libitum is very unlikely to simplify the system model, and the features which you add to your basic system might be sufficient to make it quite different from mine, even though they are ostensibly the same. It is also possible (and by no means unknown in practice) that the various software items which implement the new features might be incompatible, perhaps leading to unexpected behaviour or breakdown of the system. At the moment, that's the price we pay for the convenience of the extensions; we know of no reliable way to guarantee that combinations of additional items of software will operate harmoniously together in the context of a system, and without something of the sort the element of risk will remain.

And that is exactly why we think it important to introduce these superficially trivial items at this stage in the design. We are dealing with the specification of system requirements; the common feature of the three examples which we have given is that apparently modest extensions to system capabilities can have far-reaching implications for the machinery which must be provided in the underlying system. If we don't take these into account at an early stage of design, it can be very difficult to tack them on later in a consistent and safe way.