WHITHER MONITOR SYSTEMS ?

WHAT'S WRONG WITH A MONITOR SYSTEM ?

- which is to say, what do you need to make it more *efficient* ? (Notice for future reference that that isn't the question – or, at least, not the *only* question – we'd ask today.)

It depends on what you want to do. For many purposes, a monitor system is quite satisfactory : it works very nicely if you have a computer used by only one person who is usually there to sort out any difficulties, and the idea lived on for quite some time in many microcomputer systems of the CP/M and MS-DOS family. It is only quite recently that Windows and Macintosh systems have become significantly more advanced. Perhaps the main difference between the older microcomputer systems and monitor systems is that the microcomputers almost always take their input directly from the keyboard (and, if appropriate, mouse). This is really an accident of development, though; there's no reason why they couldn't work equally well from a predefined programme written in a job-control language, and the use of "batch files" (<*name*>.BAT in MS-DOS) demonstrates this. Alternatively, you might think of the mixture of system instructions and input to programmes which is communicated through the keyboard as a direct analogue of the monitor system's deck of cards.

Inadequacies begin to show up when any sort of complication creeps in. Monitor systems are not very good at handling more than one job at once; they are not very good at some aspects of sharing a computer between several people even if the people all only want to do one thing at once; they are not very good at handling difficulties without external assistance; they are not very good at maintaining their own integrity. Here are some areas, listed in no particular order, in which monitor systems are lacking.

WHAT'S WRONG WITH A MONITOR SYSTEM.

If the system is to run jobs for as much of the time as possible, we must speed up the transition from job to job. We can apply our universal overlapping operator to permit the operator to mount the next job's discs, tapes, etc. while the current job is running. To do that, though, there must somehow be room in the computer system for at least two jobs at once. We must also make sure that the system has a continuous supply of jobs – which almost certainly means that we must be able to accept jobs from many people. This idea brings with it a whole new set of demands for space. For example, we might hope to have enough secondary memory to hold everybody's files all the time – or, at least, a big enough sample to do useful work. And we'll also need space for information about the people authorised to use the system, and perhaps accounts – from which it follows that –

We need SECURITY.

On a simple system, the running programme has complete control, and can do anything at all within the computer's repertoire – and that includes reading, and changing, the accounting files, reading other people's files, getting at parts of memory which belong to other people's programmes, or to the monitor system, and other such unacceptable activities. There must be effective ways of preventing such behaviour. As an example –

We need MEMORY MANAGEMENT.

There's nothing to stop the running programme using memory belonging to the monitor system. That could be deliberate, to divert the system to some nefarious purpose or to change your accounts while the tables are in memory, or accidental, typically by using an invalid array index. Either way, we want to stop it.

We need STANDARDS.

A lot of software has to coexist in the system. If we are to gain the best advantage from the software available, the different packages and utilities must conform to

accepted standards. Examples are file formats (compilers which produce compatible code files can be used together), reserved memory locations (where to branch at the end of a programme, where to find a copy of the instruction which the monitor is obeying) – and recall that a few desirable conventions emerged from the earlier comments on the sample job.

We need SOMETHING ELSE TO DO.

What happens while the system is waiting for the operator to load a tape ? or while a long file is being printed ? So far as the processor is concerned, nothing and very little, respectively. If we are to make profitable use of the processor idle time, we have to provide it with something else to do.

We need CONTROL OF DEVICES.

There's nothing to stop anyone from replacing the system printer routine with some other version which might not always work, nor from sending requests to the card reader which will disable the machinery. (Possible on some models if you requested that all 960 holes on a card be punched !) That doesn't matter, much, so long as no one else is affected, but it becomes serious on a shared machine.

We need a HIGHER LEVEL LANGUAGE for job control.

Simple instructions to run programmes aren't enough to describe complicated jobs – and, particularly, a simple language might make it impossible to define satisfactory ways of coping with possible errors. At the least, we need to be able to include conditions – "IF file X is on the disc then …"; "IF that programme finished executing normally then …". Generally, we want a computer language for running a job; it should be able to talk about jobs, programmes, files, execution, etc.

WHAT CAN BE DONE ABOUT IT ?

To get MORE SPACE –

we have to buy it. But at the time these developments were taking place, this was rapidly becoming less of a problem. Hardware was getting cheaper rather quickly.

To get SECURITY –

we can no longer permit a strange programme to exercise complete control over the computer. Obviously, some programme has to control the computer, but we must make sure that it's in some sense authorised. Sensitive operations of any sort must be controlled by machine instructions which can only be executed by specially privileged programmes – such as the system programmes, which we believe to be trustworthy and safe. (Ha !)

To get STANDARDS –

we have to provide full documentation of any which we hope people will observe, and make the consequences of observing them tempting enough to encourage people to do so. We can't force people to do things the way we want them to – but maybe we can bribe them.

To get SOMETHING ELSE TO DO –

we have to give it a programme to run (because all that a processor can do is to run a programme); it's already running your programme; so we have to let it run another at the same time. Which is to say that we have to find ways to run more than one programme at once : we have to invent multiprogramming. And if the other programme isn't yours, that reinforces the case for memory management.

To get CONTROL OF DEVICES –

we have to make sure that all input and output is performed using our own system software. This is a bit like the security question, and can be handled similarly : make the actual input and output operations illegal for an ordinary programme, but supply system procedures which can be used to do the necessary jobs.

To get a HIGHER LEVEL LANGUAGE for job control –

we have to design and implement it. That's harder than it sounds, though, as we want information that isn't obviously readily accessible. How do we know whether or not a programme has finished normally ? What is the most effective way of using information about, say, the presence or absence of a file ?

QUESTIONS.

Are there any other items you would wish to add to the list ?

Would you expect development by smooth evolution, or is a sudden change essential ?
