CONFIGURING THE OPERATING SYSTEM.

The job of bending the general purpose operating system to suit a particular choice of hardware and management decisions is also called configuration. Many decisions must be made when starting a system : which devices are available, how to share the disc space between various functions, what characteristics should be set for scheduler queues at various levels, and so on. This can be a very considerable task for a large computer system. The system might be able to work out some of this information for itself; for example, if it could find out what devices were attached to it (not many systems can), it could select appropriate device handling programmes. But there is at present no way for the system to work out how the manager wants to allocate the disc space.

The system must therefore acquire information from somewhere else in order to set itself up in the desired way. In early systems, this was in the main a manual operation which was the responsibility of the system operators, and had to be done every time the system was started. Typically, the system would ask questions from time to time, to which the operator would respond with some of the required information, and all being well the job would be completed satisfactorily. (As the time taken could extend to hours, and operators are human, the job was not always either completed or satisfactorily.) More commonly nowadays, the system has a *configuration file* in which the required information is kept, and which can be edited if you want to start the system in some other way. We shall have more to say about using the configuration file in the chapter *STARTING AND STOPPING.*

Configuration from a file is comparatively easy if the system is based on tables of most things with administering procedures which can perform sensible changes to the table and the corresponding system component. For example, if the system uses a device table, and has procedures to add devices to the table, the device configuration becomes simply a matter of adding the available devices to the table one by one. The same design also facilitates *reconfiguration* : you can add or remove devices as the system is running. In older systems, such flexibility was often impossible. Devices were handled individually, and the only way to attach a new device was to stop the system and restart it with a new configuration file.

CONFIGURING FOR INDIVIDUALS.

Starting up a big system is one thing, but the idea of configuration can be extended to include starting practically anything that can have more than one way of working. In most such cases, though, there's a difference in emphasis : a large system has to be configured from necessity, so that it can do its job properly; but in many of the other instances it's more a matter of setting something up to suit someone's personal whim or fancy. This is made very clear in the name sometimes given to files which can be used to configure individual programmes – they're called *preferences* files. Such files are fairly obviously nothing to do with the operating system, but in between the big system configuration files and the preferences files for programmes there are other levels at which some sort of configuration makes sense, and some of these are of interest to the operating system. We'll illustrate our comments with examples of approaches to this sort of configuration in one large (well, shared) system, and in microcomputers.

Login files.

Most shared, or sharable, systems provide some means whereby you can determine just how your terminal is to be set up when you log in. This is usually (always ? – we can't think of any other plausible way to do the job) in the form of a command file which is automatically executed as part of the logging-in sequence. In the files you can put any instruction you like, but the common ones are of two sorts :

- hardware configuration instructions, such as terminal type and speed; and
- personal preferences, such as search path, screen configuration, aliases, and so on.

Just what's there depends on the system. If it only drives a fixed set of terminals, it might be sensible to build some or all of the terminal characteristics into the system configuration file; if it keeps its own record of, say, your search path as part of its userdata file, then you don't need it in your login file. Generally, though, the login file behaves as a part of the userdata system which is accessible to the operating system but is under your control.

Configuring microcomputers.

Microcomputer configuration is in principle much the same as the corresponding process on larger systems, but adapted to the rather different conditions under which microcomputers commonly run. The major differences are consequences of the assumption that a microcomputer is a fairly isolated system that will only be used by one person. Because of the assumption, there is little distinction between starting the system and setting it up for an individual, and the two sorts of operation may be combined. It is also fairly likely that there will be less system configuration – disc layout, etc. – to be done, and less concern for security issues. (That the assumption is frequently wrong doesn't seem to concern manufacturers much, which *does* concern people who have to worry about the security of their company's data.) We haven't attempted a survey, but here are two examples which illustrate possibilities.

In MS-DOS, the two stages of configuration are separated[IMP22], so you have two configuration files to worry about, which is confusing for many people. That's because the configuration process itself is also separated, with different components of the system dealing with the two parts. The two configuration files are read by the operating system kernel (MSDOS.SYS) and the command interpreter (COMMAND.COM). The kernel starts first, and reads a file called CONFIG.SYS; this contains specifications of a few system parameters required for necessary tasks, such as allocating buffer space, setting up device drivers, and specifying the number of disc drives. The command interpreter is started later, and reads the file AUTOEXEC.BAT. This file is an ordinary command file, and you can in principle put into it anything you like, but common operations are instructions to control the screen format, search path specifications, and other such personal preferences.

The Macintosh system developers, hamstrung by their inability to use command files until very recently, have tried a number of expedients as the system has evolved. Generally, there has been rather little provision for exercising individuality on the running of the system – which is at least consistent with their strong defence of the GUI style. (You can commonly define preferences, both in the system – often using control panels – and in programmes, and these usually persist until you change them.) In place of an explicit configuration file, all you can do is specify certain programmes which will be executed when the system is started. In version 7 of the system, these are either *extensions*, thought of as appendages of the system itself, or *startup items*, ordinary programmes[EXE2]. There is a system directory for programmes of each of these types, and to define, say, a startup item you move its code file into the startup directory.

It's interesting to see how these two approaches match the personalities (for want of a better word) of the two systems. In MS-DOS, you have quite a lot of scope to bend the system to do whatever you want by fiddling about with the command files, which fits the monitor system principle that we give you a basic system which works, and it's up to you to do what you want with it. In the Macintosh system, you can still do it, but it's harder; all that's provided is a way of running programmes at a fixed stage of the system setup process. That matches the more authoritarian style which is necessary in a system which is designed to guard its house style.

The microcomputer provisions for startup also illustrate a trend to smaller startup files which has accompanied the greater emphasis on reducing the size of the kernel. The smaller the kernel, the less detail is directly accessible to the startup process, and the greater the tendency for the separate programmes used in the system to fend for themselves. We therefore now find many programmes which maintain their own preferences files – perhaps the ultimate in low level configuration. Another reason for using such low level configuration is just the common use of personal configuration files which we are discussing. If a system provides for configuration on any level below the global, packages can no longer rely on a perfectly standard environment, so must look after themselves.

This eagerness to serve the customer's every whim sounds like the ultimate answer to our aim of bringing computer service to people, but it doesn't always work out that way. The system is receiving instructions from several different directions, and they might not be consistent. We know of one case where a disabled man turned off the keyboard auto-repeat on his machine because his movements were sometimes slow, but a word-processing package which he wanted to use turned it on again as part of its local configuration, and provided no way to turn it off. That is perhaps an extreme case, but it illustrates the potential for conflict.

COMPARE :

Silberschatz and Galvin[INT4] : Section 3.8.

REFERENCES.

IMP22 : *Microsoft MS-DOS version 3.2 User's reference*, Zenith Data Systems Corporation, 1986.

EXE2 : *THINK Reference* (Symantec Corporation, 1992).