TWO VIEWS OF MEMORY MANAGEMENT

The job of the memory manager as identified in the previous chapter is the traditional operating system's task of managing resources. It is interesting, and reassuring, that this function falls naturally out of our analysis. It has happened because we have now moved a couple of steps of argument away from people's immediate requirements – people don't usually ask explicitly for areas of memory.

> *A note to the specialists : We can reasonably assume that, if you are reading this, then you are some sort of computing specialist, and it might well be that you are quite accustomed to writing programmes which, explicitly or implicitly, ask for areas of memory. If so, we suggest that you take the statement above as a reminder that operating systems are not primarily designed for you. There should certainly be provision for you – that's why we've mentioned facilities like the APIs from time to time – but the function of the operating system is to provide services for people who are not experts, and it is useful for all of us to remind ourselves of this interesting fact from time to time.*

Nevertheless, we are still concerned to provide a service, but it's now a service to other software. We can extend the resource-manager model by defining more precisely what sort of memory management we need. That is, in brief, whatever sort of memory management best supports the software which will use the memory. It is fair to say that this factor has rarely been taken into account in designing memory managers; as we have seen, a flat memory model is almost universally used, and most memory managers are limited to providing chunks of flat memory on request, leaving it to the software to use the memory as it sees fit.

However this is managed, though, this argument shows that there are two different sets of considerations at work in memory management. One, commonly served sketchily, is the requirements of the software for memory structured in some predetermined way; the other, generally served much more effectively, is the underlying requirement for an area of memory which can be structured. There are correspondingly two parts to the memory manager's service :

- The software's requirements are met by giving each process as much memory as it wants, organised according to the preferred memory model; this is the argument from design. (In practice, the requirements are usually, inadequately, unfortunately, but not necessarily, met by maintaining the "flat memory" abstraction, and leaving the rest to a compiler.)

- The underlying requirement for memory space is handled by maintaining a pool of available memory from which allocations are made so as to let several processes share the "real" memory; this is the argument from efficiency.

Though there is some interaction between these two management activities, they are largely independent. The first is the service which must be provided to any process running in the system; the second is the concern of the operating system alone, for it is obvious that if we are to maintain our aim of providing a functional system then processes must be quite unaware of each other's existence.

The memory management itself must likewise happen at two levels. At the higher level, the immediate requirements of processes must be satisfied by providing appropriately organised memory areas; but this activity must be supported by a service which manages the raw memory available to the system, and finds available areas when required by the higher level. We regard the higher level as supporting the processes' view of memory, and the lower as supporting the system's view. The two views are quite different, as can be seen from this table :

The PROCESS sees –	The SYSTEM sees –
memory model	memory hardware
programme structure, data structure	*not interested*
internal memory organisation	hardware memory organisation
contents of programme and data	*not interested*
sees only its own area of memory	sees all of real memory, including unused memory
requests : get, release, resize, etc.	responds to get, release, resize, etc.; also move

We discuss the two views separately in the next two chapters; details of how the memory organisation is accomplished will turn up in the *IMPLEMENTATION* section.
