DISCS – THE HARDWARE VIEW

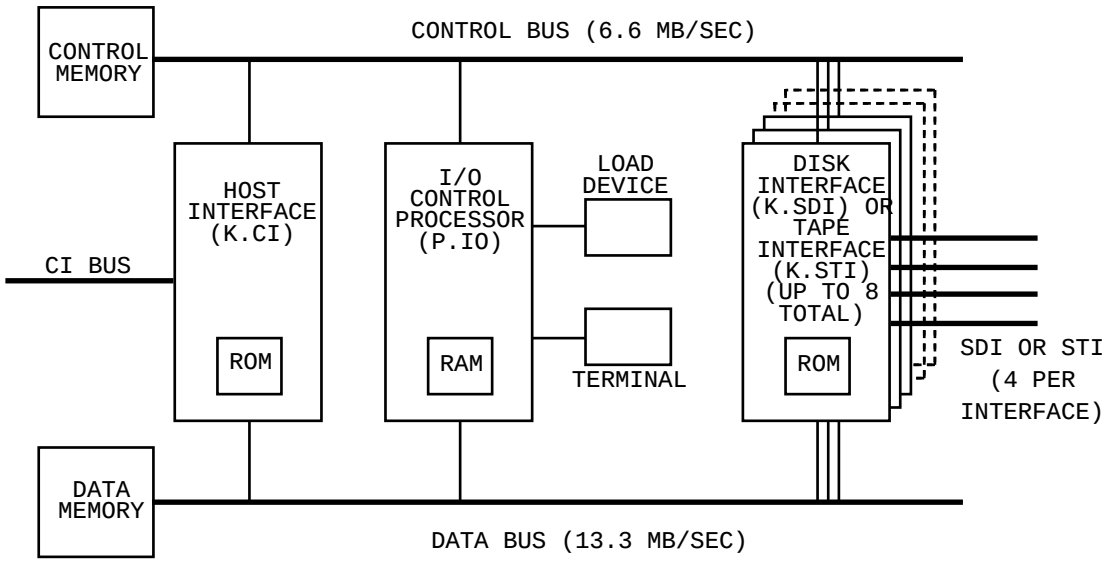
In the preceding chapter, we considered what sort of material we needed to keep on a disc, and some aspects of its logical structure. Now we reconsider the same question, but from the point of view of how to make it work on a real disc system. Once again, we observe marked differences in practice between large and small systems. Much of this chapter will be concerned with large systems, because there's more to say about them, but the principles, where they apply at all, should be universal.

The operating system's responsibility in a large disc system includes not only the administration of a single disc, but possible decisions as to how the material is to be distributed over the whole disc system. It is therefore useful to begin with a brief description of a possible disc system, to define terms and – we hope – to clarify the material which follows. (There is a more detailed account of a real disc controller system in a later chapter on *DEVICE CONTROL HARDWARE*.)

The disc devices are connected to the computer through one or more *input-output ports*. In small systems, this connection might be direct, with the computer's processor heavily involved with the details of administering the data transfer between disc and memory, but in larger systems special processors might be provided to deal with the trivia, leaving the main processor for more constructive work. These processors are called *controllers* or *channels* : as usual, the terms are not precisely defined, but it is often supposed that a channel is cleverer, and might be programmable from the main processor. If such hardware is used, it is usual for the normal input-output ports to be used only for control instructions, with the data transfer itself conducted by direct memory access, again reducing the load on the main processor.

Controllers can be quite elaborate. Not only can they transfer information : they can deal with several simultaneous requests for disc access, and optimise the order of attention to the requests to maximise the performance of the system (*disc scheduling*); and they can cope with several disc devices and several input-output ports, and perhaps communicate with other controllers as well. The aim in this complexity is twofold : on the one hand, the controller removes as much as possible of the hard work from the main processor, and on the other it seeks to optimise the disc system's performance, both by getting material to and from the discs as quickly as possible and by balancing the load on the various communications paths available to it.

Here's a diagram of a reasonably elaborate disc controller[IMP24] from a few years ago. Multiply the numbers by a small factor for modern practice; disc performance has not increased nearly as quickly as processor performance over the past years. You will see that it is a rather specialised, and quite capable, multiprocessor computer in its own right. Such a controller can operate as a file server shared by several computers, and can control up to 32 disc or tape drives, controlling transactions in real time for optimal performance.



The processors named K.* run at over six MIPS and handle the real-time processing needed to deal with the input to and output from both discs and attached computers. The CI bus on the left can be connected to up to sixteen processors. Data transfer rates of the order of one megabyte per second are expected. The P.IO processor is less ambitions than the K.* machines; it is used for the comparatively light load of planning the scheduling for the discs, which is done continuously to balance the load on the disc channels and to optimise the performance of the individual discs.

PREPARING FOR USE : LAYING OUT THE DISCS.

We saw in the previous chapter that the operating system made use of the disc system in several different ways. A part of the task of system configuration is to define which parts of which discs may be used for what purposes.

For each disc, one component – part of the system information – is not optional; this is the *volume identification*, which is used by the system to distinguish between different discs and to find its way about the discs. It includes such material as the disc name, its owner, dates and times, directory of contents, and so on. Notice that, following from the discussion of the previous chapter, this directory must record what parts of the disc are used for what purposes and a map of vacant space as well as the ordinary file directory.

The layout of other components must be defined. There are certain constraints on the definition : how many discs of what sizes and speeds are available, how these are allocated to disc controllers, and so on. The aim is to find a layout which maximises speed of response, or minimises the number of disc drives required, or satisfies whatever other goals are deemed to be important.

THINGS TO CONSIDER :

- The sizes of the various areas : plausibly system < swapping < spooling < store.
- The demand for the areas : plausibly swapping > system > spooling > store.
- Balancing the traffic through various disc channels.
- Special requirements : keep the system code on one disc drive, keep a spare drive for use in emergency, etc.

(The relative sizes and demands are honest guesses. In making them, we have assumed a fairly old-fashioned system, so "system" includes all the system software, not just the kernel; and under spooling we have included other appropriate communications, such as electronic mail.)

All these decisions are hard to make in a vacuum; you need to know a lot about the sort of work your system will be doing, the demand for file space, spooling, etc. You also need to know about patterns of work; if a lot of work goes on at weekends when

operators aren't available, that can make a big difference to the need for areas which depend on operator actions, such as archive buffer areas. It's very valuable to have good *statistics*, which system managers can use to improve estimates of future requirements, and software for collecting useful statistical information about performance is included in many operating systems. We mentioned this as a requirement of the system in *MANAGERS ARE PEOPLE, TOO*; here we discuss, rather briefly, how the principle applies to a disc system, but this should be seen in the context of planning the whole computer system. We describe this further in the next section, on *MANAGEMENT*.

In making the allocations, some things are best spread over several discs, while others are best concentrated on one disc.

Things that spread : swapping space, spooling space.

These are usually better distributed over several discs so that a single disc fault only hurts a few people, and doesn't stop the system. The common feature of these requirements is that the disc space is used as a resource; disc space anywhere will do.

Things that don't : one process's swapping space, system files.

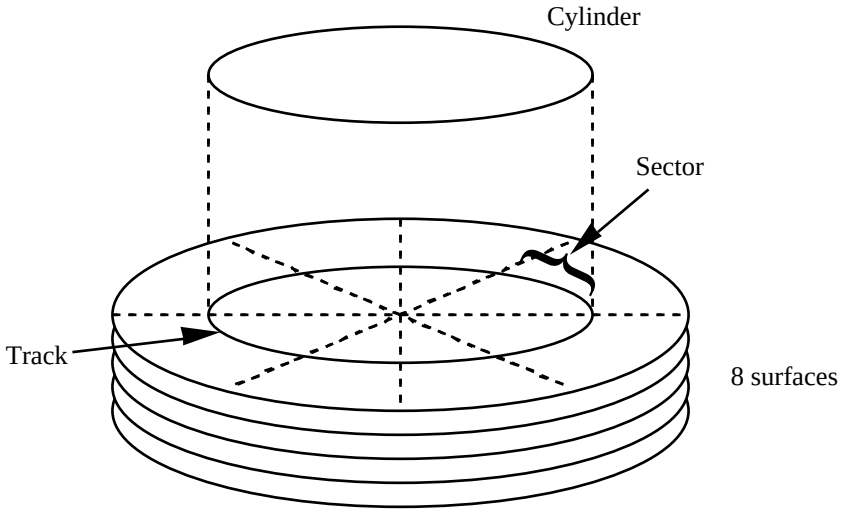
These are best kept on a single disc. In these cases, we don't just want the disc space – the data stored on the disc are significant, and different parts of the data are interdependent. Therefore, if one part goes, the rest isn't much use, and spreading the data over several discs only increases the chance that they'll be affected by a breakdown. If we really want to use many discs, as with a RAID installation, then it is sensible to take special precautions against disc failure. (We shall discuss RAID methods in more detail shortly : see the chapter *REAL-TIME DISC SYSTEMS*.)

READING AND WRITING.

The details of reading from and writing to a disc are determined by the disc hardware. Logically, this looks like a structure of the form **array [cylinders, surfaces, N] of sector**, where a *sector* is the unit of data transferred by one **read** or **write** and N is the number of sectors in one track. It is on this base that the software must implement the various sorts of structure we listed earlier.

Physically, a *cylinder* is all the disc area accessible without moving the disc access arm, so anything within a cylinder is accessible within one rotation period of the disc. Material in other cylinders cannot be reached without a movement of the access arm, and therefore typically takes longer to reach. A *surface* is, almost, what it says – one surface of one physical disc. A slightly better definition for our purposes is *the part of the disc accessible to one of the disc's reading or writing heads*; the two definitions are the same except with hardware which provides more than one head for a physical surface. In that case, the definition based on heads better describes the logical structure, though the distinction is only important to the disc controller.

A *sector* is part of a *track*, which is the intersection of a surface and a cylinder – physically, the area on one surface accessible without moving the heads. For good reasons not connected with operating systems, it is customary to record material in several bursts spread round the track and separated by unused gaps; these bursts are the sectors, and they are significant as the disc's unit of reading and writing. You have to start reading at the beginning of a sector; and you must write a whole sector at once. Any reading or writing operation always begins at the beginning of a sector, but successive sectors on a track in a cylinder can often be read sequentially in a single operation. It is therefore common to allocate sectors or clusters for a serial file in sequential groups so that this highly efficient reading and writing can be exploited. Each sector has a *disc address* by which it is identified; the address must be specified for each read or write operation.



A disc pack. The extreme outside surfaces are commonly not used.

DISC SCHEDULING.

The time taken to complete a request to read from or write to the disc depends on the relative positions of the heads and the disc address required when the request is made. There are two main contributions to the time. The first, and usually the greater unless it's zero, is the *seek latency*, which is the time taken for the heads to move to the right cylinder; the second, the *rotational latency*, is the time taken for the sector required to come round to the heads. Once the sector is found, the data transfer is typically very fast.

Given a single request for service, the disc system can do nothing clever, and just has to go through the obvious routine and put up with the delay. In a multiprogrammed system, though, there might be several requests for disc operations, each requiring service by the same physical disc, and then there is scope for optimisation. This operation is called *disc scheduling*, and the principle is easy to understand from an example.

Suppose the heads are on cylinder number 5, and a request for an operation on cylinder 90 is received. The seek begins, but while it is in progress another request, this time for an operation on cylinder 60, arrives. If the heads have not yet reached cylinder 60, it might well be quicker overall to deal with the second request first than to go all the way to cylinder 90 and then return to cylinder 60.

In practice, a disc system might receive a continuous stream of requests, and disc scheduling can be a full-time job; it is therefore commonly carried out by a disc controller rather than by the system processors. It is complicated by different discs having different characteristics, and discs with fast seek modes for long range movements.

COMPARE :

Lane and Mooney[INT3] : Chapter 12 (and see Appendix B for details of the disc drive machinery); Silberschatz and Galvin[INT4] : Chapter 12.

REFERENCE.

IMP24 : R.F. Lary, R.G. Bean : "The hierarchical storage controller; a tightly coupled multiprocessor as storage server", *Digital Technical Journal* # **8**, 8 (February, 1989).