## *FILES, STREAMS, AND PROTECTION*

From our investigations of requirements in the previous section, we concluded that we required some means of saving data in our computer system, and we have now invented files to satisfy this requirement, and streams to move the data around. A second requirement was that there should be provision to guard the safety of our data, so we now discuss how this can be put into practice.

We emphasise that the rather late appearance of this discussion in our description of files and streams does not indicate unimportance; we have deferred it only because we require a good understanding of the structures which support data management before we can discuss how they can best be kept safe. Safety considerations should certainly be given a high priority when designing the data management parts of an operating system.

PROTECTION CODES AND ACCESS CONTROL LISTS.

Both of these are now seen as file attributes of a sort. Protection codes are simple in form, constant in size, and easy to handle. Access control lists might be more complicated, are commonly unrestricted in size, and a bit less neat, but provided that they are accessible from the file descriptor and themselves kept safe from interference pose no special problems.

PROTECTION AGAINST ACCIDENT AND ERROR.

Access protection, as offered by protection codes and access control lists, is only part of the story. Another requirement which became clear in our discussion of protection was that for an **undo** operation, so that operations performed on the files could, in effect, be reversed. It was also clear that the only way to satisfy this requirement was to keep copies of anything valuable. In the next two chapters, we give examples of two such methods.

The first mechanism is intended to provide comparatively short-term protection by keeping several *generations* of a file, which record its history of changes over some time. The older copies need not be visible under ordinary circumstances, but can be restored by some variant of an **undo** instruction. ( A common name for the operation is **recover**. ) Two methods are described.

The second mechanism we describe is more concerned with long-term protection. This is the technique of *archiving* files, where files are copied or transferred to permanent storage media, ideally kept at a remote site to guard against accidents which might destroy the parent computer system. Again, there are two approaches, with rather different properties, and both are described in the chapter.

PROTECTION AGAINST FILE TABLE FAILURE.

What happens to your files if some accident befalls the system's file table ? In the worst case, they could be lost beyond retrieval, for if the only way to reach the files is through the file table, and there is no information elsewhere which can be used to reconstruct the table, then the files are gone. Archive and backup systems might offer some insurance, but archives only work if you use them, and in either case changes to the files since the most recent copy was saved are lost.

We have seen that the only sure way to guard against loss of information is to keep a copy somewhere, but the obvious way to maintain a copy of the file table is to keep a duplicate table. This would have to be kept up to date in just the same way as the main table, so, while it would give protection against physical failure of one of the copies, it would give no protection against erroneous instructions. How can we keep a copy of the information is such a way that it will survive the entry of wrong instructions ?

One way is to keep the information in the file. If each unit of the disc used for a file carries something equivalent to the file's pathname, then all units used for a particular file can be found simply by reading through the disc, and the directory can be reconstructed. A less thorough, but still useful, variant of that method is implemented in the Macintosh file system, where each file on a disc is allocated a unique identification number, which is

recorded on every unit of disc storage occupied by the file. If the file table is lost, it cannot be reconstructed, but the files themselves can be reconstructed, and – all being well – we will be able to recognise the useful ones by inspection. ( Just how well that will work when you lose the directory of your gigabyte disc and are left with fragments of, say, 10000 files is another matter, but it's better than nothing. )

A property of these methods which is perhaps just as valuable as the protection which they offer against file table failure is their potential as an **undelete** operation; if you accidentally delete a file, these techniques should be able to patch it together again even in the absence of any multiple generations, archive, or backup, provided that the disc area released has not been allocated to any other file. A disc allocation strategy which avoids reallocating areas for as long as possible can help to preserve such fossilised files for quite long periods.

## WHAT ABOUT STREAMS ?

The notion of file protection is easy to grasp; we have a well defined collection of data and certain associated system structures which we wish to guard against damage from any source. The nature of stream protection is rather more elusive, as there is no permanent object around which we can build a fence, so to speak. Nevertheless, it is clear that we have certain expectations of streams, generally associated with the principle that what we put into one end of a stream we expect to get out, unchanged apart from such stream operations as we might have defined, at the other end.

As with static data, there are techniques which can be used to provide both protection and security for streams. Protection is offered by schemes which add information to a stream in such a way that disturbances can be identified. Simple examples are the use of parity bits and checksums, but the same techniques can be elaborated to guarantee detection and even automatic correction of damage from transition faults of considerable extent. Of course, the greater the protection, the more information must be carried, so transmission becomes more expensive. Security techniques are based on encryption, which we discussed briefly in the earlier treatment of security.

These methods are usually regarded as in the province of data communications rather than operating systems. While we are by no means averse to extending the range of operating systems if it suits us, in this instance the topic is adequately covered in the department's Data Communications courses, so we won't discuss it any further here. Nevertheless, we thought it important to mention the topic here because of its real importance in the design of systems, particularly with the growing importance of distributed systems where the significance of streams in carrying data of many sorts between different, and possibly widely separated, computers is so great.

## COMPARE :

Silberschatz and Galvin[INT4] : Section 10.4.

---

## QUESTIONS.

Can you think of any hazards to files which are not covered by these techniques ?

---