3. FILE INTEGRITY AND SECURITY

All the subjects and techniques described in this manual assume that processing is proceeding without any interruption or errors. This of course is an ideal situation and one that will hopefully continue. It is also an unrealistic situation because at some time, no matter how infrequently, an error may occur that could corrupt one or more items of data, possibly the whole file. The programmer must therefore write his application to ensure that the *integrity* of a file is maintained under all circumstances, regardless of the source of the error. File integrity means that the file contains valid, meaningful, and usable data. If the application is batch processing the file then the effects are only damaging in the sense that time can be taken to reconstruct the file before it is required again. To achieve this reconstruction can be relatively simple. If the application is using the file for on-line processing then the problems of an error become far more serious. A file is required to be available immediately and one or more work stations are thereby prevented by the error from performing any transactions.

From this it can be seen that there are two areas of file use that must be supported in the event of corruption:

- files used for batch processing applications
- files used for on-line processing.

The latter includes files used for batch *and* on-line processing. To maintain the integrity of a file, it must be made *secure*. As previously stated in this manual, a file is a logical concept in that it is a body of information used during the processing of an application. The physical representation of that information is not important from the point of view of the application that uses it. If the medium on which the file is held is irrecoverably damaged it does not matter to the application *as long as an exact copy of that file is available*. A file can be made physically secure by ensuring that a copy exists, either an the same storage medium or on an alternative medium if the file can be easily copied to the original medium. An on-line file requires actions to be taken within the application.

The different aspects of file security, and file integrity, are described in this chapter for both batch and on-line files. Note that the subject of *data security* is considered to be completely application dependant – the program must check that all data handled is valid and must secure the confidentiality of data according to the requirements of that application.

3.1 Securing batch files

Security of files used for batch processing is simple to achieve if the following system is adopted. The major requirement for any security system is to ensure that a complete copy of a file is available if the working copy is corrupted or destroyed. It would be expensive to duplicate all disk files on other disks so cassette, magnetic tape, or flexible disk can be used as a back-up volume.

Suppose we have a file called DKFILE on disk that we wish to secure (see figure 3.1). After the file has been set up (stage 1) we make a copy of the file (stage 2a – this process is usually called 'dumping' the file.). DKFILE is now available for the batch application.

At stage 3 DKFILE is required by the application. There are two main sources of damage to DKFILE that we can consider:

- internal, i.e. damage that occurs as a result of the application, system error, hardware faults, operator errors.
- external, i.e. damage to the medium away from the computer such as mishandling by the operator or environmental damage (fire, left in direct sunlight, or magnetic fields).

When the damage has been discovered, copy the file from the security volume (stage 4) onto the same disk or onto another disk of the original was damaged physically. If the damage was environmental, it is possible that the security copy has been damaged as well. To avoid this, the security copy should be stored apart from the working copy, or make another copy (stage 2b) and store this in another room or building.

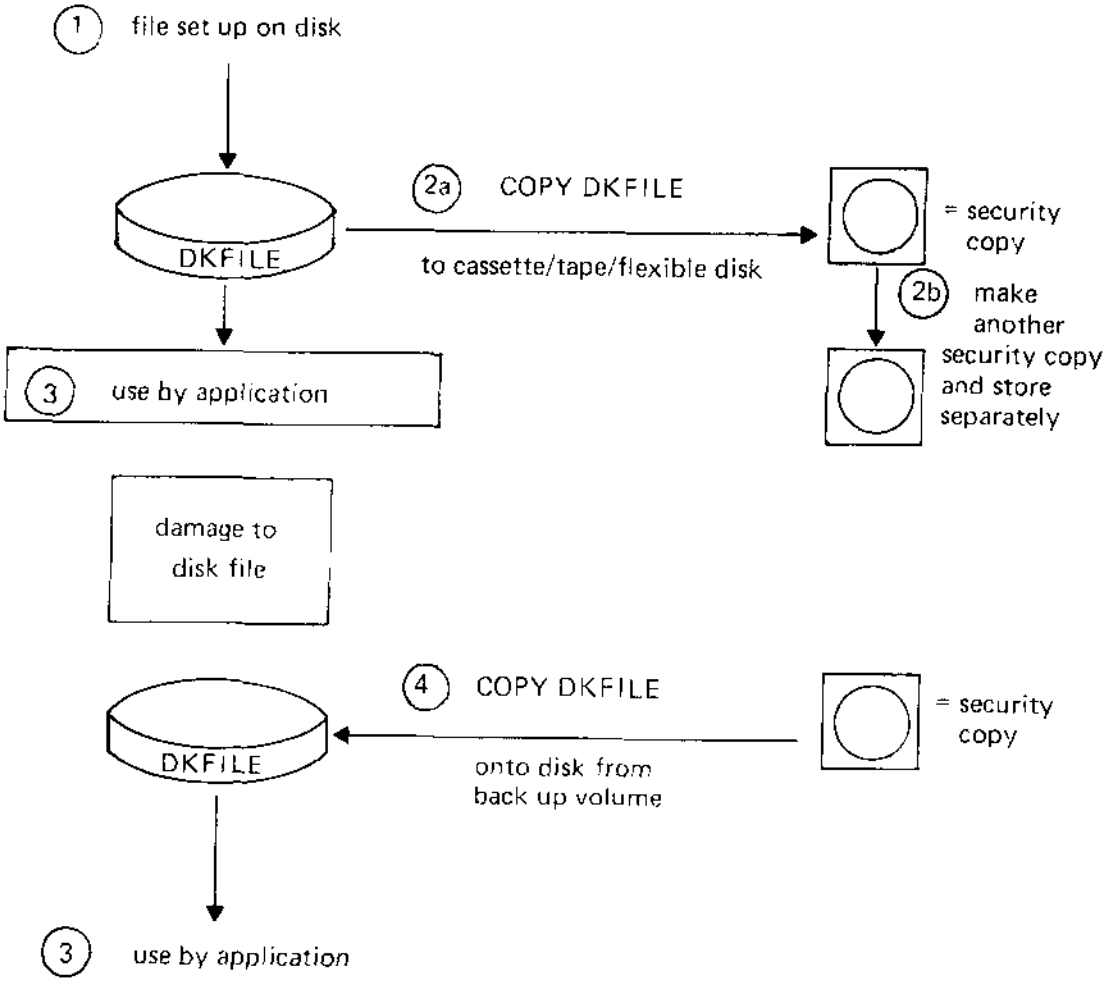


Figure 3.1 Simple security system for batch files

This is of course a very simple situation because (apart from listing, reporting or statistical applications) the application will probably change some of the data in the file. If this is the case, then the security copy will no longer represent a true copy of the disk file. Every time there is a change to DKFILE (stage 5), a copy must be made, see figure 3.2.

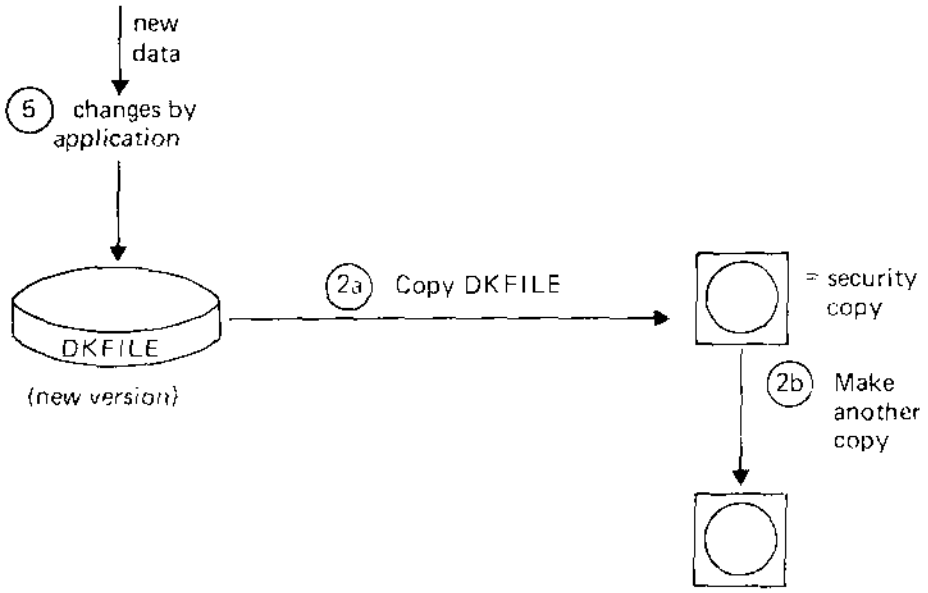


Figure 3.2 Updating the file and its security copy

It would appear that the disk file is now completely secure against corruption or damage because a copy of the latest version is always available. There is still a source of corruption that has not been allowed for, and that is the new data itself. If one or more of the updating records were wrong it might not be discovered until the updated records on DKFILE were used in later processing. This could be prevented by keeping generations of the file. For all practical purposes it is only necessary to keep three generations of the file (including the latest) plus the changes that updated each generation into the next. This method of file security is known conventionally as 'grandfather, father, and son', or gfs, and is maintained as follows, (refer also to figure 3.3):

1. DKFILE has been set up (version 1) and a security copy exists (one or two copies as required, A1 and B1 respectively).
2. During the next run of the application, changes are made to existing records, new records are added, or old records deleted (C1). The security copies no longer represent the latest version (version2) of DKFILE so new security copies must be made, A2 and B1. It is debatable whether a new B level volume is required, so three A level volumes plus one B level will give adequate levels of security. If a large number of changes are made frequently there is a case for using three B level volumes. For the purpose of this discussion only one B level is shown.
3. DKFILE can be reconstructed on disk by copying from A2 (or B1). If the changes were proven to be corrupt (wrong records deleted or changes) then DKFILE could be reconstructed by copying version 1 to disk from volume A1 and running the application with the correct changes. There is still the remote chance that the latest version on disk was copied incorrectly to A2. If DKFILE were then to be corrupted it could be reconstructed by copying the previous version to disk from A1 then updating the disk file from C1 to make the latest version of the file. When the previous generation and its associated

changes must be saved. This will ensure against practically all sources of corruption.

4. To achieve three generations of security, the process of copying is continued at each stage until there are three A level (and B level if required) volumes.
5. When the fourth version of DKFILE is generated (with changes C3) the new version (4) is copied to A1, or to a new volume A4 (and A1 then released). Thus we have 3 generations of security, or two in the remote event of corruption occurring during the process of generating A4.

Once set up, this system becomes completely automatic and can be performed either through the application or as part of the operator's 'housekeeping' jobs on the system. The decision to 'secure' any particular file must be made at the system design stage of the application. If DKFILE were only a transient file to be passed to another part of the application, or between applications, it would be unnecessary to maintain this kind of security.

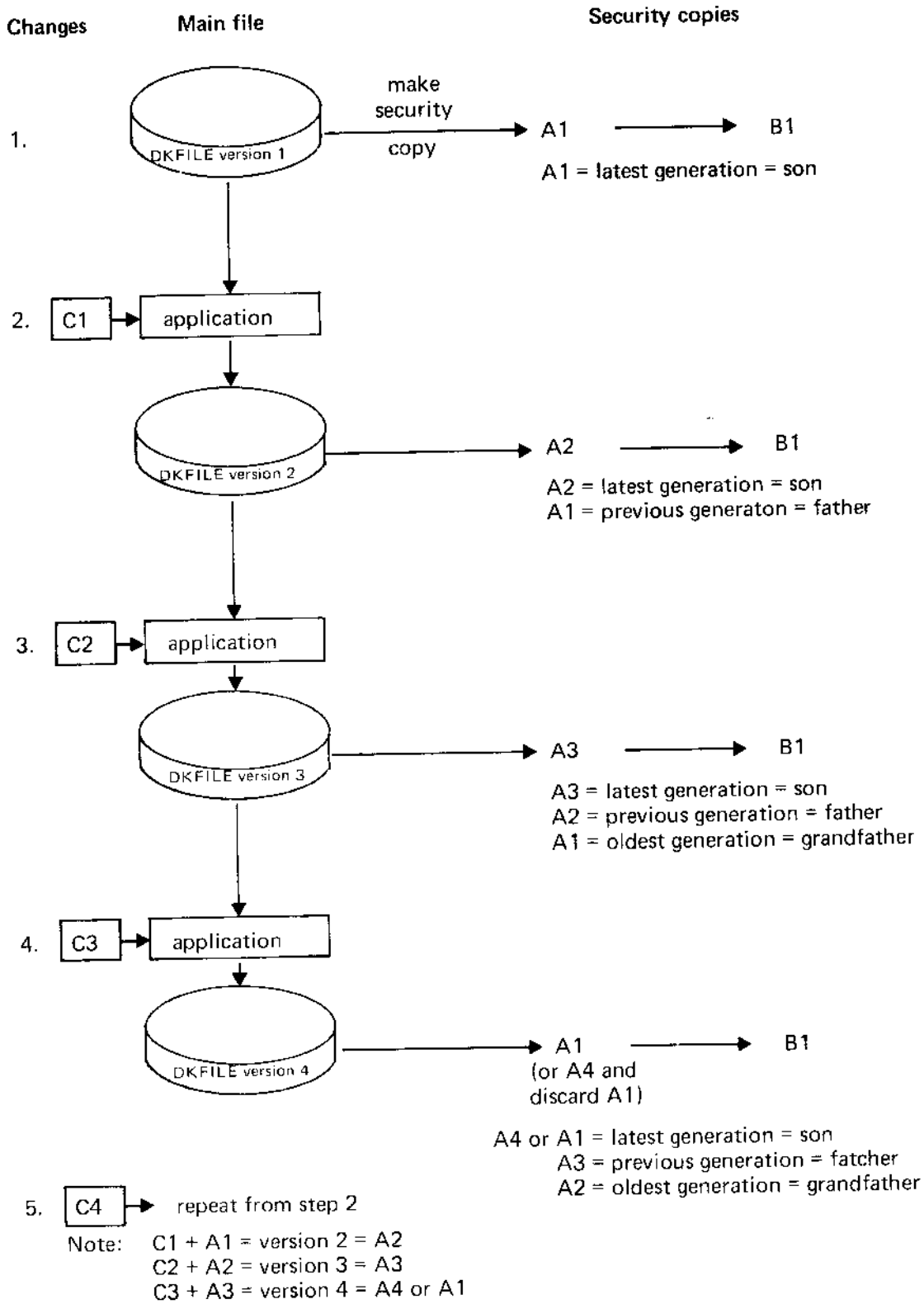


Figure 3.3 Full security system for batch files

3.2 Securing on-line files

Files that are being used by an on-line application present a more difficult situation from the viewpoint of security. The files are required to be available all the time the application is running. If the work stations and the application are providing a service to the public, like a bank, tax office, or local authority office, we can hardly expect people to come back later when a corrupted file has been reconstituted. The application must be able to provide the same service in as short a time as possible after an interruption.
Consider the situation where an application is providing a service at a number of work stations, see figure 3.4. All work stations have access to the file DKFILE through the application-records can be accessed for information, for changing, for addition to the file or for deletion.

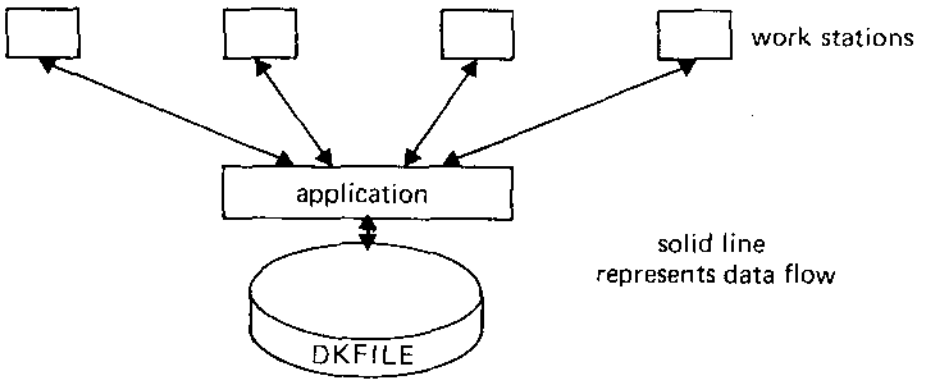


Figure 3.4 Multiple access to an on-line file

The first consideration when an on-line file is corrupted is to ensure a continuation of the service. A copy of the file must be available in as short a time as possible. Note that the original on-line file may have been updated only seconds before the corruption occurred or was discovered. How do we ensure that those changes exist on the back-up copy of the file?

3.2.1 *Duplicated on-line files*

If the installation had a limitless budget allocation one could afford the luxury of two disk units with identical files mounted – a change to one would be duplicated on the other. For most situations this is not possible although it does feature in systems where the immediate availibility of information is absolutely critical, see Figure 3.5. If the application detects corruption of some kind in one copy of the file it still has access to the other and can continue providing a service at the work stations. Where an alternative volume is mounted, the application could activate a subordinate task to create a file with the name of the corrupted file (in this example DKFLE1) then copy from the uncorrupted version. This would provide an uninterrupted service and only the operator at the computer would know that anything untoward had happened.

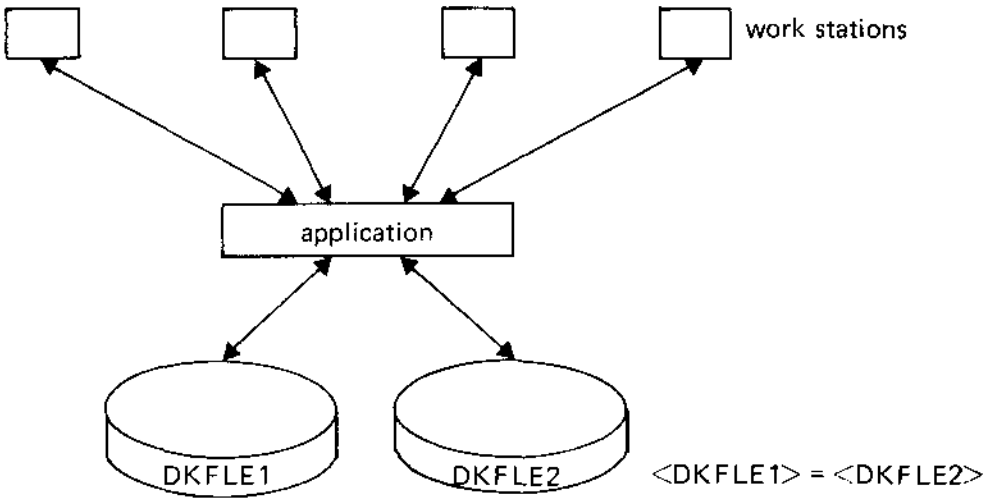
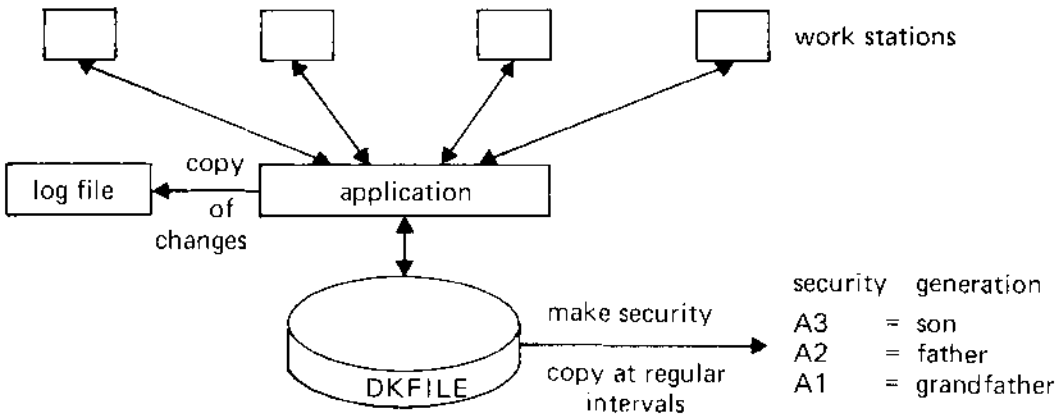


Figure 3.5 Duplicated on-line files

3.2.2 *Logging the changes*

Duplicating on-line files is an expensive solution to the problem of maintaining the availibility of information. There is a cheaper way of doing this that is similar to the method of securing batch files. It features both a back-up of the disk file and a file contaning the changes to the disk file since it was 'dumped'. The difference is that the changes to the file are recorded *as they are made,* that is, all changes are sent to a log file on cassette, magnetic tape, or flexible disk. It is not necessary to have one log file for every disk file as each record copied to the log could have an indicator or identifier attached to it showing which disk file it belonged to.

At certain intervals, decided at the system design stage, the on-line file is dumped to a security volume. The decision to dump a file could be taken at every 1 000th update, every hour or the end of the day depending upon the requirements of the application. The security volumes can be kept in the same way as those for batch files, that is, using three generations in a rotating gfs system. Figure 3.6 illustrates this situation.



In the event of corruption to DKFILE, it could be reconstituted by a subordinate task which copies from the latest generation security volume and merges this with the records belonging to DKFILE that have been logged to the log file. The result is a new copy of DKFILE complete with all the latest changes. The application could then be back on-line within minutes of the corruption being discovered.

3.2.3 *'Graceful degration'*

From the point of view of the work stations, is there any reason why the application should go 'off-the-air' at all? If the installation is giving a service to the public, is it right to make people wait because some part of the system has sustained damage? If the corruption is due to physical damage to a disk unit it could be sometime before full service can be restored. The fact that we are recording changes to the file on the log file implies that we can indeed continue to give a restricted version of the service. The application could continue to accept changes as a result of the individual transactions and send these changes to the log file. If any transaction wanted an inquiry only it would not be possible, because the disk file is unavailable due to corruption. This is all a matter for system design of course, but there is no reason why the application should not provide an alternative to the log file if that device should fail too. The important factor is the integrity of the file DKFILE and changes to the information that that file represents could be accepted as long as there is still a device operating at the central computer that can accept those changes. This method of offering a restricted service in the event of device malfunction or file corruption is known as 'letting the user down gently' or 'graceful degradation'.