EFTS RELATED FILE RECOVERY AND INTEGRITY PROBLEMS

Jack Dugger

John Rhodes & Company

Abstract

Because of the vast sums involved in Electronic Funds Transfer, the techniques used to recover messages and files must be exceptionally reliable. There is a movement today in EFTS from special purpose, custom built operating systems to more general purpose operating systems with the intent of obtaining greater versatility (higher level languages, data base methodology, etc.) The drawback is that the general purpose recovery systems are usually insufficient, and their code may no longer be accessible to the programmer.

Introduction

A single money transfer may amount to hundreds of millions of dollars. The economic penalties associated with misdirecting or misprocessing such a transaction are sufficiently great that errors of very low probability must be guarded against. In the past, special purpose operating systems were developed to specifically provide extremely high reliability and sophisticated recovery techniques. Typical characteristics such systems included:

- . extreme efficiency.
- . mirror-disk handlers as part of the operating system which write each record to both disks.
- . logging of records to tape or disk or both for recovery.
- . recovery from disk in 30 seconds to 5 minutes.
- . message queue management so written that in the event of system failure it was possible to determine which messages were processed by which subsystem, and which messages were in the process of being delivered.

While providing very advanced recovery, such custom systems often lack flexibility and versatility in other areas. Specifically:

- . little or no support for higher level languages.

- . simple file structure, unsuited for multi-index or data base operations.
- . extreme maintenance problems, caused by the unique design, which was often quite far from a computer manufacturer's standard release.

During the past few years of rapid improvement in computer price performance, it has become possible to consider using general purpose operating systems. Their relative inefficiency is overcome to some extent by increasing the machine power of the configuration, while their general support for higher level languages and data base processing gave them a desirable generality. The problem of nearly perfect recovery has not always been met, however, and cannot be cured simply by purchasing additional or more powerful hardware. This paper considers some standard techniques for file recovery, and indicates the points at which the special requirements of EFTS may exceed the capability of generally available recovery techniques.

Pre and Post Imaging

The basic design goals of file recovery are:

- . to be able to preserve file integrity after any failure by restoring the file to the condition it would have been in had the failure not occurred.
- . to accomplish recovery in the minimum possible time.
- . to impose as little overhead as possible so that the throughput of the system is not seriously impaired.

It is unfortunately true that these design goals are not mutually compatible. There is no way to restore a damaged file unless additional writing of records occurs, probably to other devices than the file itself exists on. And additional writing imposes additional overhead. Indeed, the quicker one wants to recover, the more writing is generally involved, as will be seen.

Any recovery schemes for EFTS must

assure that the transactions themselves are being logged—that is, written to secondary storage. Not only are the transactions necessary to recovery, but they are used for audit trails, retrievals, etc. In addition, file information must be written out. Two general techniques are available: pre-imaging and post imaging (indeed both may be used for ultimate flexibility).

In pre-imaging, a copy of each record in a file that would be changed by a transaction is written out to a sequential file as it exists before the update occurs (this may be the data record of any indexes pointing to it). Periodically a synchronization point is written out on the recovery file when the file being protected is quiescent; that is, there are no updates pending against it (such quiescence may have to be forced by periodically holding transaction processing). The recovery algorithm consists of:

1. If the file is totally destroyed, restore from last full file backup and reprocess the full day's transactions.
2. If the file is not totally destroyed, read the sequential recovery file backwards to the last syncpoint, writing out each record pre-image to the protected file. The file is then in the state it was in at the time the syncpoint occurred. All transactions since the syncpoint are then re-processed.

Obviously, the time to restore a complete file can be very high indeed. To eliminate the time required to reprocess all the transactions since the last full dump was taken, post-imaging is often used. This means that in addition to writing out the record as it exists before the update, a copy is also written out as it exists after the update. In the event of complete file destruction, a restore is still necessary from the last full dump. But, instead of reprocessing every transaction, the recovery file is read and each update record is written out to the protected file -- until the last syncpoint is encountered. Then the transactions encountered since the last syncpoint are re-processed.

This is a good example of the trade-off mentioned above. Faster recovery time is obtained at the expense of more overhead -- namely, the additional I/O's required to write out the record.

The above recovery algorithm describes in broad outline a workable file recovery that is used on many good general purpose systems. Before describing where problems may occur in various implementations, it might be well to speculate whether a user can build his own file recovery along these lines if the system does not have a standard file recovery supplied with it. In general, it is almost impossible to do this without altering system software. The reasons are:

- . indexed or logged files generally use pointers that are simply not available to the application programmer, therefore, he cannot log or recover them, and hence, cannot rebuild an indexed file.
- . if multiple users access the same file, it may be difficult to synchronize the updates to the log.

Assume, then, that the system has a central logging facility coupled with a general file recovery scheme. There are a number of potential weak spots which are, in fact, found to some extent in many general file recovery systems.

- . As mentioned above, the logging facility must be central and shared. Two potential difficulties arise here:
 1. The logging facility usually buffers its output. The user may not be able to force the buffers to write out. In that case, a "window" exists where a system failure could occur after a call to the logger, but before the buffers are written out. This would mean a file might not be recovered properly.
 2. The logging facility may not give the user any indication that the checkpoint was completed. This can be especially troublesome if the user has a need to initiate an acknowledgement to a transaction initiator when his transaction has been completed and safely received by the system. Such an acknowledgement often carries legal weight, and it is often not permissible to risk acknowledging until the transaction has been safely processed.
- . Some systems may not offer both pre and post imaging. If only pre-imaging is used, a penalty is paid when a file must be completely re-

created. If only post-imaging is used, then every recovery entails a re-creation of the file.

- . Simultaneous disk and tape logging is not usually permitted. This means that either all recovery must be tape (and relatively slow) or else that recovery on a different computer system may be difficult (especially with the proliferation of non-removable disks)
- . A lack of operator interface during a recovery. In general it is important to allow the operator to:
 - initiate full or partial recovery. True, the system may wish to indicate that a full recovery (starting with a restore tape created the night before) is necessary. But the operator may need to initiate a full recovery even if the system has not detected a major error.
 - initiate a full recovery, re-processing the traffic with a new program. This would be the case if a program bug were detected which was creating incorrect records.
 - suspend a given transaction. It may happen that a freak combination of contents in a message may cause a program fault. Rather than keep the application down while program bug is fixed, it may be desirable to process the other transaction while the unusual one is suspended.
- . No mirror disk facility. This can be implemented by the user, but it must be done in each application. Common code should be place in a common area.
- . No message queue management. This makes recovering messages so that each relevant application processes them once, and only once.

Conclusion

This brief introduction to EFTS standards for file recovery may make it clear why the transition from a special to a general purpose computer system must be done with careful analysis of file recovery.