LABORATORY FOR
COMPUTER SCIENCE
(formerly Project MAC)

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-78

IMPROVING INFORMATION STORAGE RELIABILITY USING A DATA NETWORK

ARTHUR J. BENJAMIN

OCTOBER 1976

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

MIT/LCS/TM-78

IMPROVING INFORMATION STORAGE RELIABILITY USING A DATA NETWORK

Arthur Jay Benjamin

October 1976

MIT/LCS/TM-78

IMPROVING INFORMATION STORAGE RELIABILITY USING A DATA NETWORK

Arthur Jay Benjamin

October 1976

This research was sponsored in part by the Advanced Research Projects Agency (ARPA) of the Department of Defense under ARPA order No. 2095 which was monitored by the Office of Naval Research under contract No. N00014-75-C-0661.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LABORATORY FOR COMPUTER SCIENCE
(formerly Project MAC)

CAMBRIDGE MASSACHUSETTS 02139

ACKNOWLEDGMENTS

The success of any research project depends upon the interaction of ideas among interested people. The research reported in this thesis is the result of many comments and criticisms from the people around me.

First, I would like to thank Professor Saltzer, my thesis supervisor, for encouraging the discovery of new insights by helping to define the essence of the problem. His continued patience, interest, and comments have helped both the development of the ideas, as well as their presentation in this thesis.

During the implementation phases of the research, Raj Kanodia, Ken Pogran, and Doug Wells have been invaluable in providing the necessary details for using the ARPAnet. Similarly, Jerry Farrell and Hal Murray at the Computer Corporation of America have been very helpful in providing assistance in using the Datacomputer. The Computer Corporation of America has been very generous in making the Datacomputer facility available for this research.

Finally, thanks go to Nancy Federman, Harry Forsdick, Jeff Goldberg, Doug Hunt, Allen Luniewski, Drew Mason, and Dave Reed, not only for their technical contributions, but also for the interesting distractions which they provided to help put this project in perspective.

DEDICATION

This work is dedicated to my mother.

IMPROVING INFORMATION STORAGE RELIABILITY USING A DATA NETWORK *

by

Arthur Jay Benjamin

ABSTRACT

Backup and recovery methods using magnetic tapes are common in computer utilities, since information stored on-line is subject to damage. The serial access nature of the tape medium severely restricts the flexibility and simplicity of accessing and managing the stored data. A method using a data network will be described, to present a backup mechanism which takes advantage of a large, inexpensive, random access remote data storage facility to provide data access and management functions that are more flexible than those provided by a traditional backup facility. Although data transfer rates will be reduced, data access and management will be simplified, and system availability will be improved. The work described is based on a network backup facility built for the Multics computer utility, using the ARPAnet.

Thesis Supervisor: Jerome H. Saltzer

---

* This report is based upon a thesis of the same title submitted to the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, on September 27, 1976 in partial fulfillment of the requirements for the Degree of Master of Science.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Page

LIST OF FIGURES

Chapter One

Introduction

1.1 Background

Many contemporary computer systems are too complex to understand well enough to guarantee that they are free from design errors. It is even harder to guarantee that an implementation of the design is correct. As a system grows in size and sophistication, it becomes more and more complex, and as the ability to understand this complexity diminishes, so does confidence in correct and reliable operation. Even if the correctness of the implementation could be guaranteed, external environmental factors preclude 100% reliable operation. For example, hardware failures, power failures, etc., are inevitable. Faced with the fact that failures will occur, the system designer must consider reliability enhancement and failure recovery strategies for the important components of the system.

One of the most important, and complex, components of a computer operating system is the file storage component. File system reliability enhancement and recovery techniques are often overlooked in the haste to design and implement an operational system as quickly as possible. Although

this approach may seem to be the most economical in the short term, looking back later often shows that an unfortunate tradeoff has been made between system development time, and subsequent utility, performance, and reliability of the system. Due to the importance of the file storage system, the complex nature of the interactions among its components, and the resulting inevitability of software and hardware failures, a file system storage backup and recovery facility is essential.

The most common file system reliability enhancement and failure recovery facility used today utilizes magnetic tapes for storing backup copies of files. The complexity of such systems varies from a simple strategy for dumping a bit-for-bit image of the entire file system storage, to a versatile tape management facility allowing selective retrieval of individual files. In all cases, the backup facility is a specially designed file storage system which can only be used for limited file system purposes (such as storing backup copies of files), and can not provide much flexibility, due to the restriction in access capabilities inherent in the tape storage medium. However, tape is not the only feasible storage medium for storing backup copies of files.

1.2 A Data Network Approach to Reliability Enhancement

In this thesis, we investigate the use of a data network for providing file storage capabilities for backup purposes, using a random access storage facility, in such a way that the backup facility can be regarded as an extension to the regular file system. The backup facility maintains a consistent backup image of file storage, and the local file system is regarded as a cache storage facility for efficient access to files stored at remote computer systems within the network. The major advantage in using a network over using magnetic tapes is the availability of a large, inexpensive random access storage medium which provides increased flexibility in storing, accessing, and managing data. Separation of file contents from information describing the file, and its structural relationship to the rest of file storage, allows independent management of file contents and catalog information. Independent management of different classes of data, although possible, would be inefficient using a tape system, because access to data items scattered throughout tape storage requires the use of slow sequential access techniques.

1.2 A Data Network Approach to Reliability Enhancement

The use of a data network for file backup improves reliability at a reasonable cost for recovering from situations in which a failure does not cause extensive damage. It is assumed that catastrophes are rare, and therefore the discussion in the thesis is oriented toward the more frequent but less catastrophic loss of information typically affecting less than half of all file storage. The network facility hastens recovery by allowing the system to be made available to users as soon as the catalogs and system libraries are intact. Files need not be retrieved in a special predetermined order, and no time consuming tape searching is required. Instead, a flexible random access data management facility allows files to be retrieved "on demand" as they are referenced by users.

The increased flexibility in data management afforded by a network system allows better user control over backup operations. Specification of how to produce consistent backup copies of a database is possible, and the user can know the precise state of backup copies of files. This is accomplished by separating the mechanisms of detecting modifications to local copies of files from the policy of incorporating those changes into backup copies to make them consistent. The provision for allowing a user-defined backup policy provides the necessary flexibility.

In addition to flexibility of data access and management, the network provides diversity of file storage facilities. The ability to widely share a file storage service among several computer systems makes a large random

access storage facility economically feasible, just as timesharing made large centralized computer systems economically feasible. The accessibility of multiple file storage facilities increases the total reliability of file storage, since failure of one instance of a network file system does not imply the failure of the whole backup facility. Using multiple storage facilities improves reliability, but introduces problems of multiple distributed copies of files. In fact, the backup facility is viewed as a special case of a distributed file system, and the major problem is to keep backup copies consistent with local cache copies of files. Since backup storage is mainly accessed to update inconsistent copies, and only occasionally to retrieve copies, a scheme is used to centralize distributed copies by a file migration technique. Under normal circumstances, all backup copies will be located at one storage facility, and only when that facility is unavailable will copies become distributed, and then only until they can again be consolidated. The central consolidation scheme is shown to be analogous to some common tape management strategies.

Throughout the thesis, backup is viewed as a solution to problems of reliability. However, the incorporation of specific designs within the framework of contemporary computer systems and computer networks raises other issues which are investigated at the same time. A prime interest is the investigation of mechanisms for managing backup copies of files as a special case of the harder problems inherent in distributed file systems. It is hoped that some of the solutions to the special case problems will provide some

insight into the nature of the harder problem. The organization of the thesis leading up to this goal is described below.

1.3 Plan of the Thesis

Chapter two presents a model of a file system that will be useful for describing various mechanisms in the operation of the backup facility. Access to information in the address space is distinguished from access to information in file storage. The controlled, predictable nature of the latter type of access is the basis for detecting modifications to local files.

Various approaches to defining and improving reliability are outlined in chapter three. Reliability is related to availability, and a goal of maximizing availability is set for the backup facility to work towards. A model for a backup system that improves reliability by maintaining redundant copies of information is described, and some existing implementations are outlined. The model identifies four interacting components: a file modification detection mechanism, a backup policy implementation, an I/O facility, and a file retrieval technique.

A detailed design for a general purpose backup facility is presented in chapter four. Based on the models of chapters two and three, each component of the system is considered in turn. The interfaces are described, and

related issues, such as security and protection of information, user requirements, etc., are considered. The nature of the backup file storage facility using a data network is also described.

Finally, in chapter five, the specific issues that are important in a network implementation of the backup facility are described. A solution to the problem of naming objects in a network, using globally unique identifiers, is followed by a discussion of consistency of files. Consistency is guaranteed by the use of temporary shadow copies, which themselves remain consistent before and during backup operations. To make the backup system more reliable, the use of several storage facilities in a network is considered, but the resulting production of multiple copies of files can cause difficulties in locating and accessing the correct copy. To counter the effects of these difficulties, centralization of file storage is described, followed by a discussion of protection of remotely stored information using techniques of data encryption. Since the use of a network usually restricts the bandwidth for data transmission, as compared with local I/O channels to tapes and disks, for example, the expected performance of the proposed system is analyzed using elementary concepts from queuing theory, and is found to be adequate for the file usage patterns expected in a typical shared computer utility. Finally, some ideas about charging and accounting for network backup services are considered.

1.3 Plan of the Thesis

Chapter six provides a summary of the results of the research that led to the ideas in this thesis. These ideas are then extended slightly to illustrate their applicability to other types of facilities that might serve users. The chapter ends with a presentation of suggestions as to how some of the earlier ideas might be useful in considering the problems of the distributed file system. The extensions of the ideas as complete solutions to distributed file system problems are left as topics for further investigation.

Chapter Two

A File System Model for Storage Reliability

2.1 Introduction

One of the most important functions provided by a computer system is the management of stored information for users. A file system provides a facility for managing stored data, and names and other attributes associated with the data. This chapter presents a file system model that will be useful for describing various mechanisms employed in the operation of a facility for improving information storage reliability.

In most systems, the access characteristics and representation of stored information in files (usually in secondary storage) differ from the access characteristics and representation of the information as referenced by program instructions executing on a hardware processor. For example, access to secondary storage may be slow, but information is stored in "blocks," while access to information in primary memory by the hardware processor is fast, but only a word at a time is referenced. The contents of a file, as a collection of information, is a user level concept. The individual units of information (e.g. words, pages, records, etc.) that comprise the contents of the file are

implementation level concepts. These information storage units are referenced by programs during their execution, and are stored as ordered sets of bits in primary memory or in a hierarchically structured memory system. The names used to reference information represented in this form are used by the hardware processor, and comprise the program namespace.

The user is concerned with ordered sets of bits called files. Names for files are typically human interpretable arbitrary length strings of characters, and these names make up the user namespace. The investigation of two classes of file systems, the read-write and the direct access file system, and a subsequent look at information storage representations in more detail will help clarify the relationship between the user namespace and the file system, and the program namespace and the address space manager.

The next section describes the two classes of contemporary file systems in terms of the data access operations and name mapping functions which they implement. This description is followed by a brief discussion of the use of catalogs in the file system implementation. The major content of the chapter follows with the identification of two states for information accessibility. File contents that are currently in use by programs, and therefore have a representation that is susceptible to modifications if referenced in a program namespace, are distinguished from a file storage representation that implements a more permanent storage. The explicit nature of updating the permanent storage from the volatile address space representation of the

information will later provide a mechanism for detecting modifications to files.

2.2 Two Classes of File Systems

The function of a file system is to maintain a binding between a user oriented filename and a unique identifier (UID) which is used to name the file in the context of its implementation. This binding allows filenames to be mapped into program namespace implementation oriented names for use by programs executing hardware instructions. Whereas the implementation of the program namespace and address space are necessary components in the architecture of a computer system, the implementation of the user namespace and the file system are not. They are provided for the convenience of the user so that operations on stored data can be managed in a higher level naming context than the hardware instruction level.

In addition to providing a mapping between filenames and UID's, the file system may also store descriptive information about each file, such as its length, its creation time, its access attributes, etc. This descriptive information is stored in catalogs, or directories. Catalogs also implement the filename-UID binding, and will be discussed in more detail in the next section.

2.2 Two Classes of File Systems

The method by which the file system enables program namespace references to information stored in a file, given the filename, leads to two kinds of file systems. In the read-write file system, information referenced by the filename is identified from the filename-UID binding maintained by the catalog, and copied into (or out of) a program namespace context (i.e. the address space), and this copy is accessed in the usual address space way directly by the hardware. In a direct access file system, the filename is translated via the catalog into a program namespace name, and this name will subsequently allow the program to access the information directly in its usual address space way. Thus, the read-write file system, given a filename and a program namespace name, will move information between the two implementations of the namespaces. The direct access file system provides only a name translation function. Access to the information is accomplished in the program namespace by using the name provided by the translation. Given a direct access file system, a read-write file system can be modelled by simply performing the additional functions of moving the information between the file (as identified by the name translation) and a copy of the file named in the program namespace. Therefore, we will just consider a direct access file system.

2.3 Catalogs and File Storage Organization

The file system catalog or directory is used to record associations between names (and possibly other descriptive information) and the stored representation of files. A file, defined as an uninterpreted ordered collection of bits, is a very general object. A file may in turn be used to represent objects in a higher level context, e.g. ascii bytes representing stored text, or machine instructions representing a program which can be run on a processor. We will not be concerned with any higher level semantic context associated with a file. All files will simply be ordered sets of bits. However, there is one significant implementation strategy that will be considered, which defines a structure on a particular type of file that is used by the file system itself.

Consider the implementation of catalogs in the file system. If the catalog object is implemented using the file, an interesting property results. In general, the catalog stores information about files. If some of these files are actually catalog objects, then catalogs can contain information about other catalogs. This leads to a structured hierarchy of files, with non-terminal nodes being catalogs, and terminal nodes being either catalogs or non-catalog files. A catalog is therefore a special type of file that is implemented, interpreted, and manipulated only by the file system.

2.3 Catalogs and File Storage Organization

Given a tree structured organization of catalogs and files, one may hierarchically order information according to one's needs. For example, the administrative hierarchy often leads to a convention of assigning each user a unique identifier recognized by the system at login time. In addition, each user is associated with a project for administrative and accounting purposes. Adopting this convention in a tree structured file system organization usually results in a set of catalogs for the projects, with a sub-catalog for each person in the project. This strategy can be extended horizontally by the system administrator for system related purposes (e.g. projects, libraries, resource control and accounting strategies, etc., may be system relevant categories realized by catalogs across the highest level in the tree), or vertically by the user for personal requirements (e.g. organization of information according to subprojects, realized by catalogs within catalogs), as pictured in figure 2.1. The important point is that any structure imposed on the organization of user files is implemented by the catalogs, and no inherent assumptions about such organization is required in the stored representation of non-catalog files themselves.

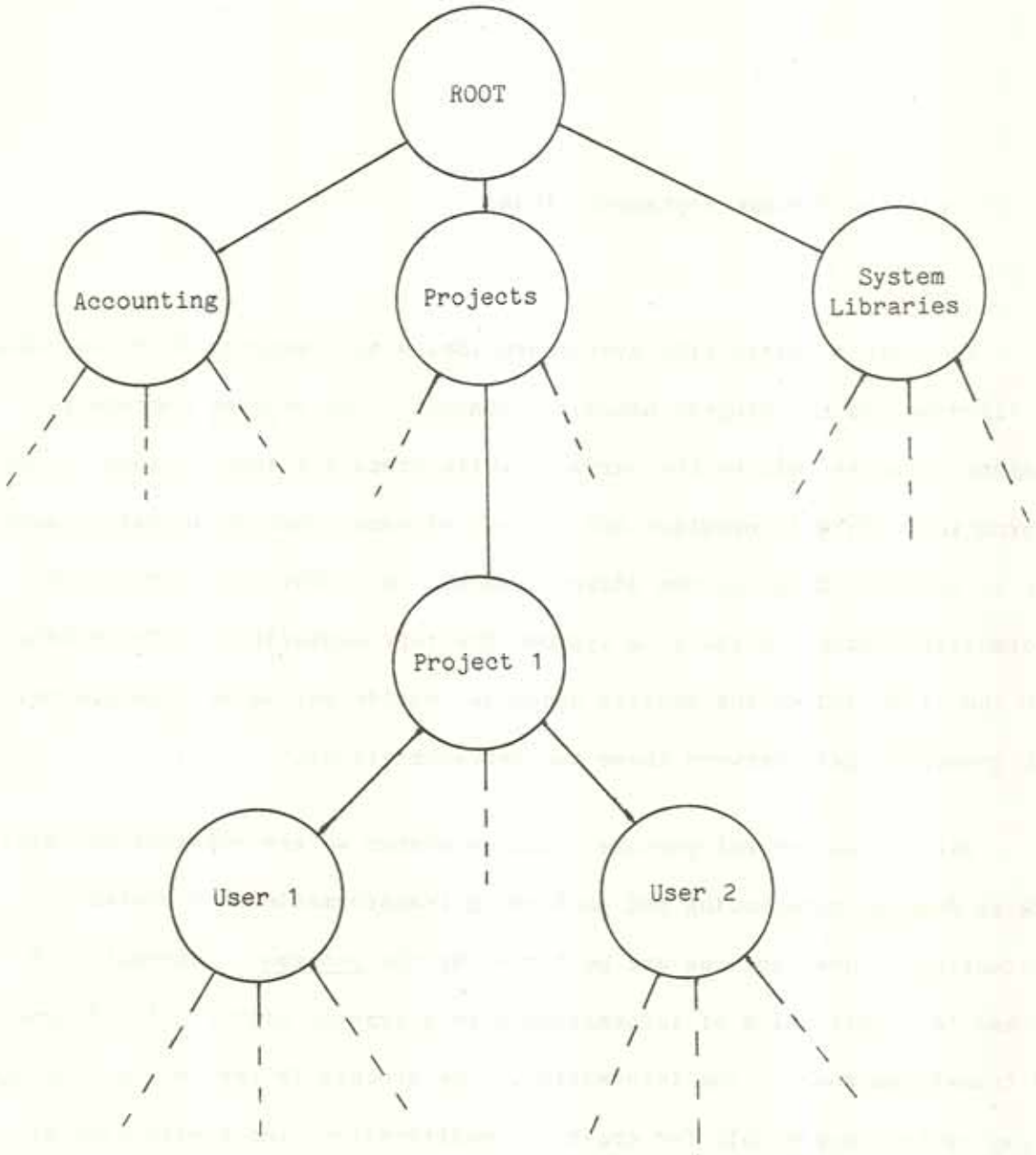

Figure 2.1 Hierarchical Organization of Files

2.4 Information Storage Representations

The direct access file system provides a name mapping function between the filename and the program namespace context. The program namespace consists of names used by the hardware while executing instructions to access information. This information and the set of names used to access it make up what is commonly known as the address space. In studying reliability of information storage in the file system, the representation of stored data in both the files and in the address space is considered, as well as operations that transform data between these two representations.

Within the general purpose computer system we are considering, useful work is done by referencing and performing transformations on stored information. These actions are performed by the process. Informally, a process is a collection of information with a dynamic history of references to and transformations on the information. The process is the only active agent in the system responsible for creation, modification, and destruction of information. Each process is associated with one address space (previously defined as an ordered set of bits) named by hardware interpretable names. Furthermore, it is assumed that the address space is implemented in a

segmented hierarchically structured virtual memory [BCD 72]. Many other systems can be modelled as restricted sub-classes of this type of system, so it will be adopted in this model.

A major reason for building a hierarchical virtual memory system is to multiplex a scarce primary memory resource. The result provides a uniform way to address information as if it were in primary memory, even though special mechanisms may actually be required to support this illusion. This usually implies a subsystem (the "paging" subsystem) for managing the multiplexing of primary memory by moving copies of information between the allocated faster memory and the more abundant but slower secondary levels of storage. The policy governing the allocation of primary memory is based on performance considerations, so that, for example, the information referenced most recently in the address space of a process executing instructions will remain in primary memory, since it is likely to be referenced again immediately (the locality of reference principle). Otherwise access may require the movement of information among the levels of the memory hierarchy. This implies that infrequently named information will not be in primary memory most of the time. In fact, information will only be moved into primary memory when it is named, and it can only be named by a process in the context of its address space.

The operation of expanding the size of an address space (in this model, the process of adding a segment to an address space) is called initiation. In the direct access file system model, initiation involves adding information

(identified by the filename and the file system mapping function) to the address space by creating a new segment, and returning the program namespace name for this segment. Subsequent execution of hardware instructions that name this segment invoke the virtual memory machinery to create the appearance that the contents of the segment are actually stored in primary memory. The inverse operation, called termination, removes a segment and its name from an address space.

There must be some mechanism for long term storage of information when it is not initiated, i.e. for storing the informational content of segments when they are not part of any address space. Such storage is not the responsibility of the hierarchical virtual memory manager, but of the file system. Since such uninitiated information can not be named by any process, it will not be stored in primary memory, but in secondary memory (probably at the least expensive, slowest access level). Furthermore, since it cannot be named by any process, and only the process can modify stored information, such information will be static. The static long term representation of stored information managed by the file system will be called an immune file, since it is not subject to modifications by the actions of any processes. When this information is a writable part of an address space under the auspices of the virtual memory manager, it will be called a susceptible file, since it will be susceptible to changes by processes which can name it. Thus an immune file is the long term storage representation for segments stored in the file system, and the susceptible file is the storage representation for segments in the

virtual memory initiated from immune files. In this thesis, we will be concerned with the reliability of information managed by the file system and stored in immune files. The discussion of reliability enhancement mechanisms in later chapters requires an understanding of the distinction between immune files and susceptible files, and how the file system and the virtual memory manager should cooperate in using these objects.

When an immune file is named by a process (using the filename), the file system will communicate with the virtual memory manager (using the storage implementation oriented UID bound to the filename) to request creation of a new segment, initiating the immune file into a segmented address space, and returning the new name of the segment in that address space. The file remains susceptible until it is terminated from all address spaces.

For reliability and efficiency purposes within the virtual memory implementation itself, it is often the case that a copy of an immune file is made, and this copy is initiated and made susceptible to modifications by processes. In this strategy, there is a special action performed by the virtual memory manager, called an update, which periodically reflects changes made to the susceptible copy, back into the immune file. This special update operation occurs outside the context of a process or an address space, and actually changes the immune file. The important point is that such changes should occur only by the update operation according to a well defined strategy. We will adopt this latter model for three reasons. First, a model

in which a copy is not first made can be described by requiring an update operation to occur after each change to a susceptible copy of the file. Second, since changes to an immune file occur outside the context of a process or an address space, updates can occur at well defined times governed not by the unpredictable dynamics of a particular process or set of processes, but by a precise strategy defined by the virtual memory implementation. Finally, the controlled operation of updating an immune file will be an important mechanism in the strategy to be developed later for improving reliability of stored immune files by making copies of them when they change.

In summary, susceptible files are managed by the virtual memory subsystem, and are subject to numerous, frequent, and unpredictable modifications by processes. On the other hand, the virtual memory manager reflects these modifications back to immune files by the update operation in a well defined way. Immune files are managed by the file system, and are not subject to numerous, frequent, and arbitrary modifications by processes. Reliability of stored immune files is studied because the update operation provides a precise way of determining their state of modification (based on strategies and policies implemented in the virtual memory manager), and because this type of storage is more abundant and more permanent than storage used in implementing the address space. The next chapter introduces the area of reliability and presents a model for describing the virtual memory manager and its interface with the file system in terms of improving reliability of immune file storage.

2.5 Summary

This chapter has presented a model for a file system that will be used for describing various mechanisms in the operation of a backup facility. The read-write and the direct access file systems were described, and a hierarchical file structure was illustrated. Finally, the implementations of information storage in the address space and in the file system were defined. In the address space implementation, information in the virtual memory is susceptible to modifications by processes, while information in file storage is modified only in a well-defined precise way, by an update operation. The update operation will be the basis for detecting modifications to files.

Chapter Three

File System Reliability Enhancement

3.1 Introduction

Reliability can be a subjective impression based on how "useful" a facility is, or it may be an objective measure of a system's performance based on operating statistics. Subjectively, a data storage mechanism is "reliable" if its users are confident that data can be stored and later accessed correctly and when desired. Making a system more reliable boosts confidence in both availability and correct functional operation. Measures of availability can be used to provide an objective measure of reliability, but characteristics such as user confidence and correctness of operation are harder to quantify. Therefore, this thesis considers a measured improvement in availability to indicate an improvement in reliability.

During the course of a system's operation, failures are possible. Generally, a system will function until a failure occurs, and then it will become unavailable. Unavailability may be a direct result of the inability to proceed after a serious failure, or the intentional result of an action by an error detection mechanism in response to a failure, in order to prevent

subsequent widespread damage. Techniques for error detection for information storage systems will not be discussed here. Instead it is assumed that incorrect functional operation of the system is detected by an error detection mechanism that responds by making the faulty system unavailable until the cause and effects of the failure are corrected. Thus, a system is available only when it is functioning correctly, and availability (and reliability) depends on how quickly recovery from failures can be completed. The thesis investigates mechanisms for redundant storage of data to facilitate rapid recovery from failures, increase availability, and thereby improve reliability.

Measures of reliability based on measured availability and failure recovery time are discussed in the next section. Recovery time is not the only important factor, however. The extent of recovery possible, in terms of the amount of unrecoverably lost information, is also considered. A goal of the reliability enhancement system is to minimize both recovery time and the amount of unrecoverable loss of information, at a reasonable cost.

Traditional approaches to improving reliability are summarized and an approach which provides for recovering backup copies of information is selected. Some desirable characteristics of this type of system are stated, and shown to be properties of existing implementations using magnetic tapes.

3.1 Introduction

The last section presents a model for a backup system that will be referred to extensively throughout the rest of the thesis. In this model, four components are identified. A file modification detection mechanism is used to invoke a backup policy implementation. Based on the particular policy in use, requests for movement of data are made to an I/O component. The I/O component is also used by the file retrieval mechanism.

Finally, it is suggested that a network implementation is advantageous because it retains the desirable properties of a tape system, while offering improvements that offset the penalties of new constraints. This suggestion is the subject of detailed consideration for the remainder of the thesis.

3.2 Measures of Reliability

Reliability and availability are related. When the inevitable system failure occurs, information may be lost. How quickly and how completely recovery can be done determine reliability, also. In some cases, rapid recovery and high availability are important, while in other cases it is more important that recovery be complete and no information be permanently lost. Many systems attempt to reach a compromise between the two costly ideals of rapid and complete recovery.

3.2 Measures of Reliability

In applications such as air traffic control, stock inventory data bases, and computerized bank accounts, it is important to prevent unrecoverable loss of information. In commercial timesharing applications, it may be tolerable to accept some permanent loss of data if recovery can be made more rapidly, thereby improving availability. The effort invested into improving reliability, and the tradeoff between speed and extensiveness of recovery, depend upon the cost of a failure, in terms of resulting danger to human life, financial loss, etc., for a given system application. This thesis will investigate reliability considerations for a general purpose computer system providing file storage facilities for user applications. Such a system might be a large shared computer utility (e.g. Multics [Cor 65, Org 72, MIT 74]), or a small dedicated private system. The issues and solutions will apply to both classes of systems, but the difference in scale may dictate differing implementations. The larger system will be the primary object of interest in this thesis, and the reader may imagine how the ideas apply to a small system.

Availability is the first factor in measuring reliability. It can be measured as the mean time between failures (MTBF). The type of system being considered here is assumed to exhibit a MTBF on the order of several days.

The second factor useful for measuring reliability also relates to availability. This factor is the mean time to recover (MTTR). The MTTR measures how fast recovery can be accomplished, and thus indicates how soon

the system can be made available again. The MTTR depends on the severity of the failures encountered, the amount of lost information, and the method of recovery used, and is assumed to be no longer than an hour. The MTBF and the MTTR together lead to a measure of availability expressible as a percentage of total time that the system is available.

The last factor that relates to reliability is the system's susceptibility to unrecoverable, permanent loss of information. This measure depends on the other two factors, and on the particular recovery strategy employed. The basic idea described below is that information which is newly created is subject to permanent loss until some measures are taken to assure its reliable storage. The amount of such information provides a measure of susceptibility to unrecoverable loss of stored data.

According to the file system model described in chapter two, information is created only when a susceptible file is modified. This modification will result in an update to the immune copy of the file, thereby storing new information in file storage that will require the invocation of the reliability enhancement facility to protect against loss and permit recovery if needed. The invocation of the reliability enhancement facility results in a backup operation.

For the system, there is an average update rate and an average backup rate. The backup rate will always be smaller than the update rate. The difference between the two rates defines the resolution of the backup facility, i.e. how quickly backup operations can respond to updates.

3.2 Measures of Reliability

Unrecoverable loss of data occurs if the system should fail after immune files have been updated, but before backup operations have completed for those updates. The average number of these "outstanding" updates provides a measure of susceptibility to unrecoverable loss of information, and is a function of the resolution.

Improving the resolution reduces susceptibility, but may be expensive. A tradeoff is required between the amount of work relegated to the system for improving the resolution, and the amount of information loss acceptable to users. The extent of such a loss that is acceptable depends on how hard it is to regenerate the information.

Changing the resolution of backup operations need not affect the MTTR, because improvements to resolution can be made in the backup facility, while the MTTR depends on the recovery techniques. The separation of backup and recovery strategies allows the resolution and the MTTR to be optimized independently by strategies specially adapted for their needs. A goal of a network approach is to do just that: to develop backup and recovery strategies that exhibit both high resolution, and low MTTR. The network makes this separation feasible, while traditional tape systems prohibit this separation, resulting in a situation in which attempts to improve the MTTR cause resolution to be degraded, and vice versa. Some reliability enhancement techniques are outlined in the next section, and the tape backup facility is discussed.

3.3 Approaches to Improving File Storage Reliability

The strategy for improving reliability is to increase availability by better error recovery techniques. Error recovery techniques include backtracking, majority consensus [We 72], and redundant storage of data [Fr 69, Pe 71, St 74]. Other techniques in structuring operating systems help to limit the effects of errors. These techniques include the use of structured programming [Par 72], protected domains [Shr 72], dynamic reconfiguration [Fab 73, Shl 71], and distributed processing [Orn 75, Row 73]. Careful construction of system components may facilitate the use of special a posteriori repair utilities for rebuilding databases, regenerating files, etc. Since repair utilities require explicit knowledge of the structure of the damaged objects, they will not be discussed here.

One structuring technique for the file system catalog and file storage implementation fits particularly well with the objectives of recovery in helping minimize recovery time. By allocating storage for catalogs and the files they describe in physically local storage units, damage is likely to be confined to a minimal number of subtrees in the hierarchical file organization. By confining damage to the fewest number of catalogs, the number of users that are affected and the chances of damaging critical shared

file system contents that need to be available for system operation (e.g. system programs and data, system libraries, administrative databases, etc.) can be minimized. If the system can be made available as long as this critical data remains undamaged, then a technique that confines damage improves the chances for survival over a situation in which scattered damage can occur. Only those users whose catalogs sustain damage will be affected, while the system could be made available to others. The recovery strategy used is to quickly restore critical data that was damaged, making the system available sooner and lowering the MTTR, rather than requiring that all data be restored before any user can use the system.

Other structuring techniques can be useful for improving reliability, for example, by building a system with internal redundancy so that damaged components can be repaired or reconstructed. The approach taken in this thesis is to build a system that provides external redundancy of information in stored files, so that intact current copies of damaged information can be recovered from a storage facility different from the file system. By using an independent storage facility for backup copies, it is less likely that damage to the file system will also affect the backup system. In fact, it is desirable that the backup system be remote from the file system, in a literal sense, so that physical damage to the file system (e.g. fire, explosion, etc.) will not affect the backup system. There are many strategies in use now for implementing such a backup system. Most use magnetic tape as a storage medium since it provides an inexpensive way to store large volumes of information,

and tapes can be removed from the site of the computer and stored remotely. Whatever storage medium is used in a particular implementation, there are several ways to produce backup copies of information.

One technique for maintaining redundant file storage information is to produce a dump of the entire storage implementation. To produce a consistent backup copy requires that the system be unavailable during the backup operation, so this approach is not acceptable. Although a raw dump of the file system may speed recovery from a complete loss of all stored information, it is assumed that such failures are rare, so we will look for a technique better suited to more frequent but less extensive loss of information.

For efficiency purposes, it may be desirable to make backup copies of the smallest unit of information storage that is managed by both the file system (for long-term storage) and the address space manager (for access in the virtual memory implementation). However, such implementation dependent information units (e.g. pages, disk records, etc.) are not usually visible to users, who view information in terms of files. Since backup operations are concerned with immune files, and not necessarily with their implementation for use by the address space manager, the file is selected as the basic object for which backup copies will be maintained. (1) A model of a backup system that maintains redundant copies of files is discussed in the next section.

---

(1) The selection of the file as the fundamental object managed by the backup facility does not preclude the implementation of a strategy for maintaining backup copies for smaller storage units that comprise the file implementation,

3.4 The Backup System Model

In this section, a four component model of a general purpose backup system is presented. After describing the function of each component, the application of the model to a tape backup system is outlined. Finally, it is suggested that a data network backup approach offers advantages of simplicity, diversity, and reliability over the tape approach, and the remainder of the thesis presents a more detailed consideration of the issues.

The four component model of a backup system is illustrated in figure 3.1. First, there is a mechanism for detecting when a backup copy should be made, based on the state of the file system. Second, a policy implementation governs if, when, and how a copy will be made. To actually maintain a backup copy of a file, an I/O facility is invoked. Finally, there is a mechanism for retrieving backup copies of information that has been lost or damaged.

---

for efficiency and performance purposes. However, this level of detail would usually require an explicit knowledge of the file storage implementation, with a complicated facility for mapping between pieces of a file and backup copies of those pieces, and for determining the logical location in the file system of backup copies. If files are large and are modified in small, localized areas (such as in database management applications), this approach might be worthwhile. However, as discussed later in chapter five, most files in the type of system being considered here are small, so the added complication of this approach is not justified.

3.4 The Backup System Model

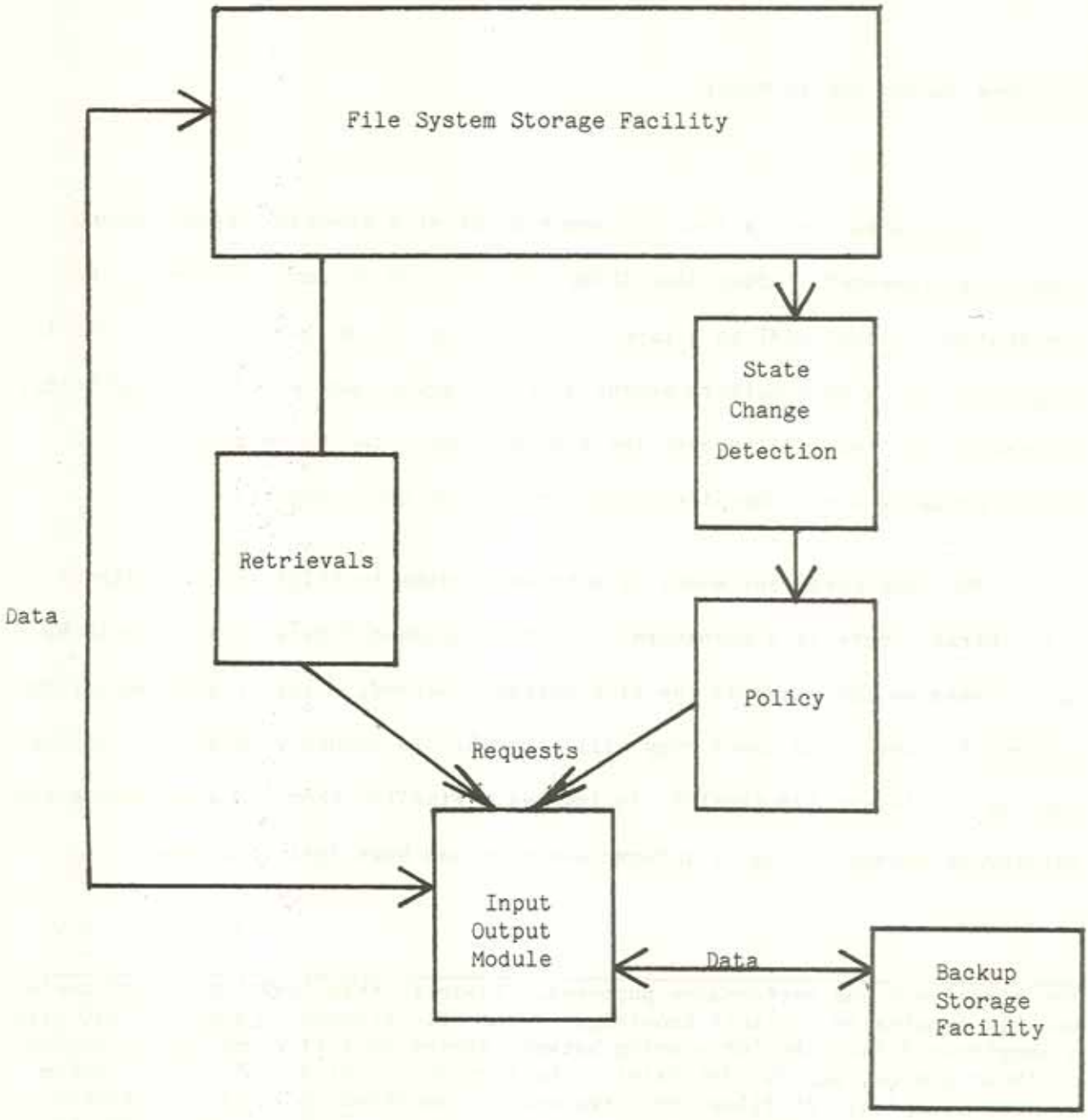

Figure 3.1 Four Components of a Backup System

3.4 The Backup System Model

The purpose of the backup facility is to maintain redundant, up-to-date, consistent copies of files. (2) The local file storage facility provides a cache for efficient access to files which are saved in backup storage. Changes to the state of the local cache storage are detected by the backup system, and reflected into backup storage by periodically copying those files which have been created, deleting copies of those files which have been deleted, and re-copying those files which have been modified since the last time they were copied into backup storage.

Some state change detection mechanisms require the inspection of file storage to passively notice that an immune file has been modified, making it inconsistent with respect to the backup copy. A more responsive detection mechanism that can provide better resolution relies on a strategy to actively notify the backup policy implementation when an inconsistency has been introduced (by an update operation) in an immune file in the local cache. While the policy is usually incorporated within a state change mechanism that operates by inspecting file storage, the model described here separates the detection mechanism from the backup policy implementation. This separation provides greater flexibility in defining a policy (which is an administrative issue), and simplifies the design by isolating the essential mechanisms (which are technical issues).

---

(2) In fact, backup copies may not always be up-to-date, or even consistent. These and other problems are discussed in detail later in the thesis.

3.4 The Backup System Model

The policy implementation determines if, when, and which backup operations are to be requested when it is notified of an inconsistency. It invokes the I/O facility to perform operations for maintaining backup copies. The I/O facility contains knowledge about the implementation of backup copies, and manages the backup storage facility.

Finally, a retrieval mechanism restores copies of files after they are lost or damaged by a failure. Essential files, such as the system libraries, are restored before the system is made available. This can usually be done quickly, and the system can be made available shortly after the failure. In addition, all catalogs are assumed to be present. As users reference files not already in the cache (using the catalogs), copies are retrieved from backup storage. This strategy also uses the I/O facility to reference backup copies, and allows the system to be made available to users, even though files may still be missing from the local cache. Total availability and perceived reliability benefit from this recovery technique that does not require all files to be restored before users (many of whom may not have been affected by the failure) can perform useful work.

The actual maintenance of backup copies is the responsibility of the I/O facility. Most contemporary systems use magnetic tape storage facilities, as mentioned earlier. In a tape oriented system, backup storage is incrementally updated by sequentially writing new copies of files on successive tapes. A major disadvantage lies in the inability to re-organize

previously written information in the backup storage. Because it is unreasonable to replace an inconsistent copy on a tape with an up-to-date consistent one, due to the sequential access nature of the medium (the inconsistent copy may be on another tape, or it may be of a different size, etc.), an elaborate tape management scheme is necessary to balance the volume of backup tape storage and the cost of locating and accessing information, against the level of reliability enhancement desired. The inability to access backup storage in a flexible, efficient manner also makes it difficult to determine its state, and hence it is difficult for users to determine just what state backup copies of their files are in. Nevertheless, such tape systems have been successful in providing reliability enhancement for file storage [Fr 69, St 74].

If more flexible, efficient access to backup copies of information is convenient, many of the management problems become simplified or disappear. A random access data storage facility available in a network offers these advantages. This is not to say that new problems and constraints do not arise with a network approach; indeed they do. However, under the operating assumptions made about the expected frequency and severity of failures, limitations caused by such new constraints (principally caused by limitations in realizable bandwidth for data communication) will be outweighed by the simplification and improved availability obtainable with a random access information storage facility.

3.4 The Backup System Model

A network implementation should retain the advantages of contemporary tape systems (e.g. remote storage of data, economical storage of large volumes of data, etc.), while providing improved reliability and greater flexibility so that users may take better advantage of backup services that are provided. A design for a file storage backup system that uses a data network, based on the four component model, is described in the next chapter.

3.5 Summary

This chapter has presented a model for reliability enhancement for stored information. Availability of the system and rapidity and extensiveness of recovery from a failure were the major factors determining reliability. Use of techniques such as structured programming, protected domains, dynamic reconfiguration, and distributed processing provide useful approaches to constructing reliable systems, but the technique selected for recovering from failures in which data is lost is one which provides redundant storage of data.

A model of a backup system which maintains redundant copies of information was described. In this model, four interacting components are responsible for performing backup operations. A state change detection component determines when a backup copy should be made. A policy component decides if and how a backup operation is to be carried out. An I/O component

is responsible for the actual transfers of data, and a recovery component provides a facility for retrieving backup copies of files after a failure has caused loss of or damage to information.

Chapter Four

Design for a File Storage Backup System

4.1 Introduction

The previous two chapters have presented models for a file system and for the reliability of such a system. This chapter will elaborate on the design of each of the four interacting components shown in figure 4.1. This design derives from an implementation of a facility providing backup file storage for the Multics computer system [Cor 65, MIT 74, Org 72], utilizing the ARPAnet [ARPA 74, Cro 72] and the large capacity data storage facility known as the Datacomputer [CCA 75, CCA 73, Mar 75], which is accessible via this network. The purpose of the backup facility is to improve file storage reliability by providing an interface between local file storage and remote backup storage.

It is recognized that each component must solve particular application problems for backup, and at the same time satisfy global requirements of the system design. The state change detection mechanism and policy implementation are designed with system security issues in mind. Only information that is needed for these components is made available, and only in a controlled way.

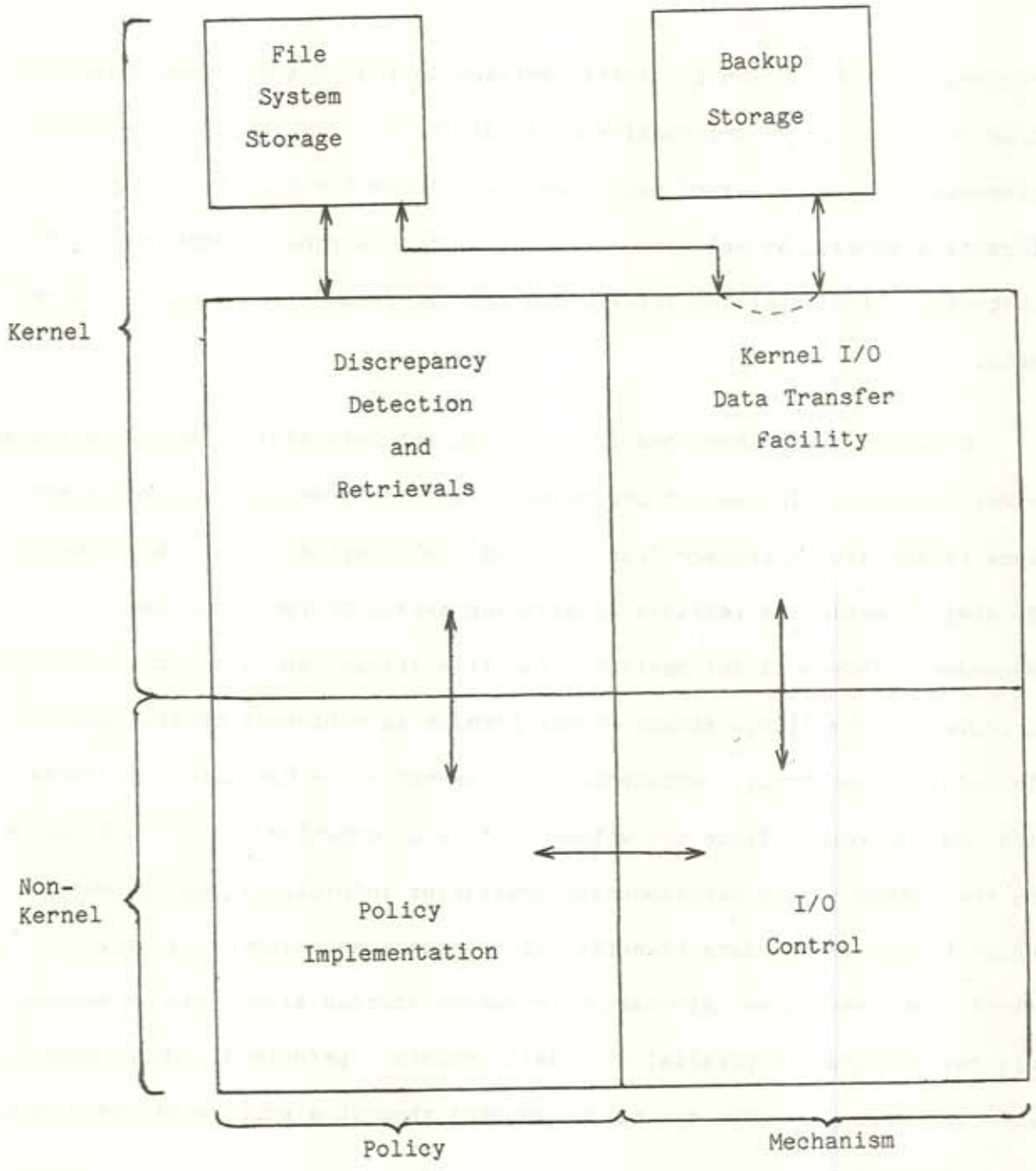

Figure 4.1 Interacting Components in the Backup System Design

4.1 Introduction

The access to information is clearly defined so that unauthorized release of and/or modification to information will not occur. Isolation of a policy implementation from a kernel mechanism facilitates the required control over information access, by defining a specific inter-component interface. This architecture also separates information release issues from denial of service issues.

A mechanism is described for creating and maintaining backup copies on the Datacomputer. The use of the network and the Datacomputer makes large volume random access storage feasible, and also simplifies data management. This simplification is realized by dividing backup storage into two independent classes of information: the file itself, and a description of the attributes of the file. Random access permits asynchronous control over a file and its descriptor, corresponding to asynchronous operations to local files and catalogs. Since the volume of file information can be much larger than the volume of the corresponding descriptor information, and network bandwidths may impose data transfer rate limitations, separate access and control over descriptor information in backup storage allows data management functions to occur in parallel with data transfer operations. Other uses for the backup storage system can be implemented through a particular descriptor management strategy.

4.1 Introduction

The last component is concerned with system recovery after a failure, and specifically with file retrievals. The local file storage is accessed by users in an unpredictable way, while the backup storage is accessed only by the process responsible for managing it over the network. If the file system is considered to be the backup storage and the management programs for implementing this storage on the Datacomputer, then the local file storage can be viewed as a cache memory for the backup image of file storage. This image is accessed through a single, well defined, controlled interface, and should be less susceptible to damage. When local damage occurs, the file can be removed from the cache. Subsequent reference to the file will cause it to be retrieved automatically from backup storage, if the catalog indicates it should be in the file system image. Therefore, only the catalogs are required to be present after a failure, for the system to be made available to users. An empty cache will then be loaded on demand as users reference files. This permits file retrieval to proceed with a limited bandwidth network by first restoring files that users need immediately, and then restoring others. Since only catalogs need be available before the system becomes useful, system recovery time is lowered, and limited network bandwidth is used to best advantage.

Each component described involves an internal set of mechanisms, and a set of external interfaces. These will each be described in the following sections.

4.2 The Discrepancy Detection Mechanism

The set of files in backup storage represents the contents of the file system, and local storage provides a cache memory structure for accessing it. The backup facility manages this cache by keeping backup copies consistent with respect to cache copies. The basic driving force for maintaining this consistency comes from the discrepancy detection mechanism.

The backup file collection is maintained as an image of user's information, while the catalogs define the structural relationships between the files. Two classes of changes can cause the cache state to become inconsistent with respect to the backup file collection. Changes to catalogs cause alterations to the structure of the file collection (e.g., create or delete operations). Catalogs are managed by the file system, so such operations as create and delete which cause discrepancies between cache state and backup storage state will be detected by the file system.

The second class of changes involves modifications to user's files. Only susceptible files in an address space are subject to intentional modification. Such modifications made in an address space by a user's process will eventually be incorporated into the immune copy of the file by the update

operation. An update, in general, (3) is an operation which is invoked occasionally to reflect a set of changes to a collection of susceptible information, out to the immune version of that collection of information. An update operation should be an explicit, well-defined event initiated by the address space manager, such that it will be the only operation that will ever intentionally modify an immune file. As such, this event will also provide a discrepancy detection mechanism for events that modify immune files, making them inconsistent with respect to backup copies.

In a general purpose computer utility allowing controlled sharing of information, access to system data and state information needs to be controlled by the kernel. The discrepancy detection mechanism is a kernel facility with an interface that provides control over the release of file storage and address space state information. These interfaces are shown in figure 4.2 and are described below.

The objective of the discrepancy detection mechanism is to communicate to the backup policy implementation information about candidates from among all susceptible files, which become inconsistent with respect to the corresponding backup copies, and hence become eligible to be copied into the backup storage system. Some current schemes utilize a mechanism which

---

(3) The update operation is employed to maintain consistency among multiple copies of data. Making a new backup copy of a modified file is also an update operation. However, in this chapter, the term "update" will refer only to the operation of keeping the file system (immune) contents consistent with the address space (susceptible) contents.

4.2 The Discrepancy Detection Mechanism

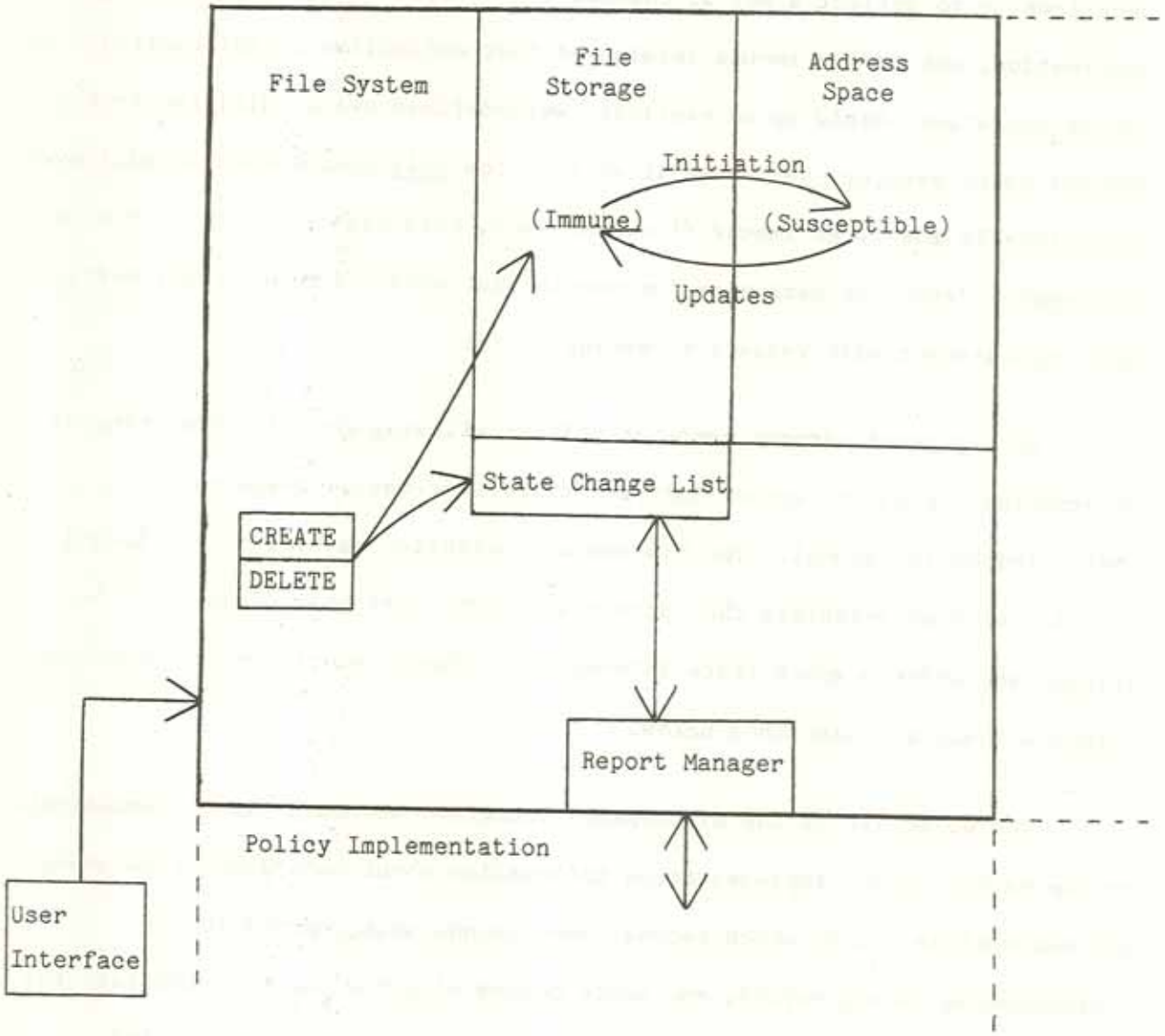

Figure 4.2 The Discrepancy Detection Mechanism

combines the detection and policy implementations. We will attempt to separate the two. Instead of using a passive strategy whereby the detection mechanism leaves "clues" about candidates for being copied from the file system into the backup system, which the policy implementation must discover on its own before acting, we use a dynamic technique whereby the detection mechanism notifies the policy implementation about candidates for backup as soon as they are detected. This strategy allows backup copies to be made sooner after modifications occur, improves resolution of the backup facility, and reduces the potential for unrecoverable loss of information that could result from a failure.

In principle, the state of the file storage cache could be determined by inspection. In practice, the volume of storage precludes techniques that require inspection of the entire cache, because frequent inspection would be too time consuming. To determine eligible files for backup would entail a search of the cache each time state information was needed, and furthermore would not be a dynamic operation. To facilitate rapid and frequent access to cache state changes would be sufficient for purposes of discrepancy detection. A state change list is maintained by the file system for this purpose. (4)

---

(4) The state change list provides an efficient mechanism for determining the current state of the file system, if the previous state is known, and therefore provides redundant information for efficient access. If this information is lost in a failure (an event assumed to be rare, relative to the frequency of state changes), it can be reconstructed by the more time consuming operation of inspecting the entire cache, thus allowing recovery.

4.2 The Discrepancy Detection Mechanism

Only two operations can cause intentional cache state changes. CREATE or DELETE requests change the structure of cache storage, and updates to immune files change the contents of cache storage. Each type is an explicit operation, and results in an entry being placed on the state change list. This list element is called a discrepancy report.

The state change list, or discrepancy report list, records the history of state changes in the cache storage system. Discrepancy reports will eventually result in some action being taken, through a report processing mechanism. Report processing is done by the policy implementation, through a kernel interface to the file system.

The policy implementation exists outside of the operating system kernel as a user-level process. Its purpose is to decide if, when, and how to dispose of discrepancy reports, and subsequently request backup-related operations. In order to respond dynamically to state changes as they are detected, it communicates with the file system through a report manager interface. The file system informs the policy implementation of the arrival of each report, through an interprocess communication (IPC) operation called notification. Conversely, the policy implementation processes reports it receives from the file system and returns information about their disposition. In addition, the file system provides a user interface for the usual types of file system operations. Access to the report manager interface will usually be restricted to the policy process, which runs as an administrative support

function in the system. This policy implementation is discussed in the next section.

4.3 The Policy Implementation

The policy implementation provides a facility for processing discrepancy reports. This processing can be arbitrarily complex, depending on the particular administrative needs. However, this complexity is a purely administrative issue, and is divorced from the mechanism issues of report notification and subsequent backup operation. In the global picture, the policy implementation provides an administrative interface between kernel-detected state changes and the available primitives for performing backup operations.

The major function of the policy implementation is embodied in a decision module, shown in figure 4.3. When notified of the generation of a new discrepancy report, the policy decision module begins processing it. Based on inputs from the file system and/or users, the decision module informs the report manager of the disposition for the given report, and (optionally) places a request for backup operations in the request buffer (e.g., priority queues). The decision module should implement a fair and flexible policy, so that unconcerned users receive a default level of reliability enhancement for their files, while users with special requirements can easily specify

4.3 The Policy Implementation

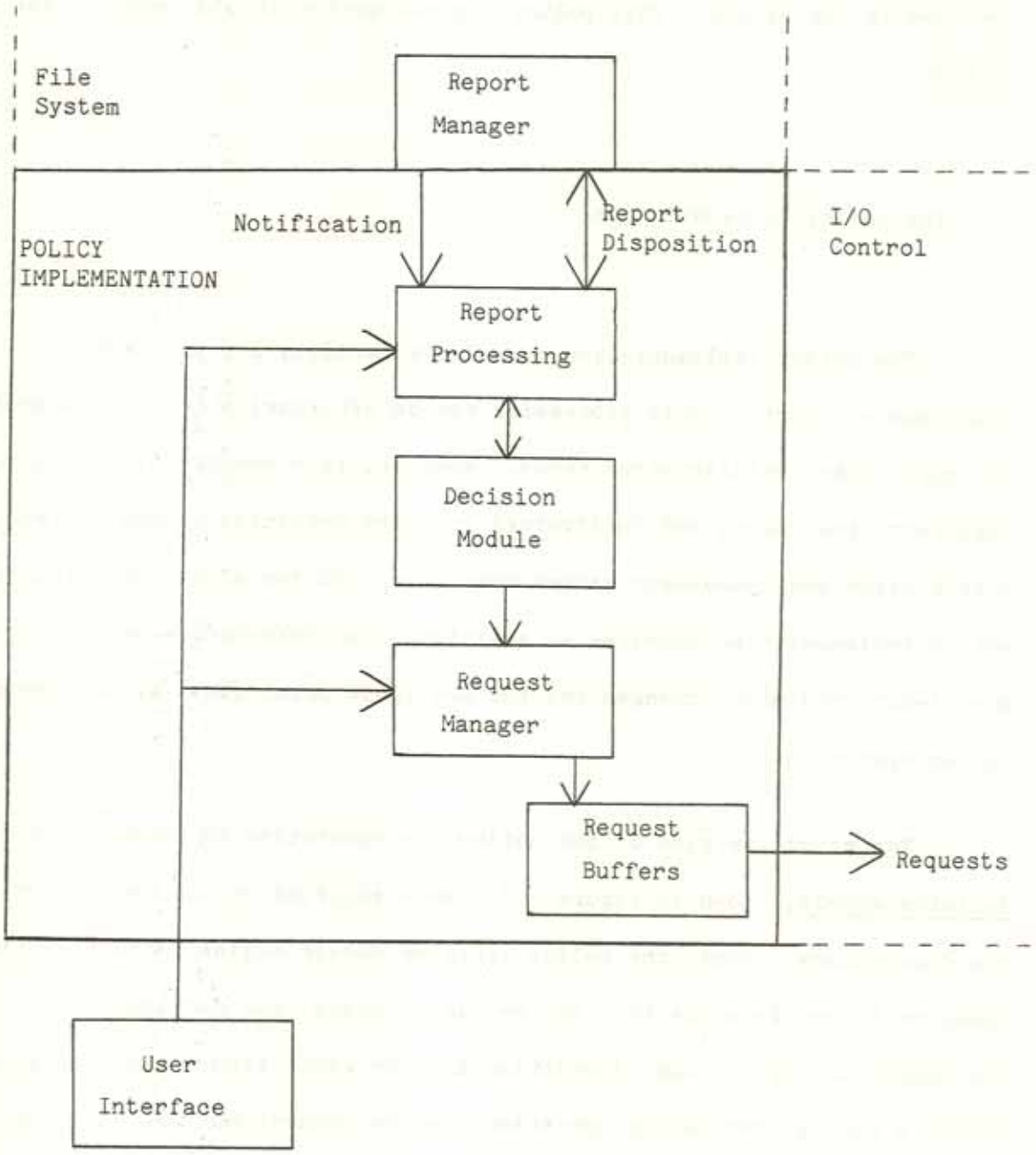

Figure 4.3 The Backup Policy Implementation

non-standard decision criteria without adversely affecting backup operations for unconcerned users.

For example, notification of the occurence of an update from the susceptible file to the immune file in the cache can be passed on to the user. Instead of relying on a default policy which provides for making a new copy of the modified file, it may be desirable for the user to journal the changes just incorporated in the immune file by the update, and later produce a new complete copy in backup storage. Allowing the user to journal changes as they occur to immune files can provide better resolution than waiting for the system default strategy to produce a backup copy. The flexibility of this type of mechanism provides more precise user control over backup operations, so that the precise state of backup storage is always known to concerned users.

The list driven backup scheme outlined here can model the operation of current typical backup systems. A particular policy implementation requests backup operations based on file storage state information. For example, the complete, incremental, and catchup dump strategies can be modelled as decision criteria based on dates and times of file modification and backup, stored as file storage state information. Similarly, user-defined strategies are implemented as user-level policy procedures which embody specific decision criteria. An appropriately authorized process can then obtain report

notifications, process reports based on file storage state information it is authorized to access, and request backup operations through the user interface to the policy implementation's request buffers. (5)

The kernel interfaces for accessing report list entries and file storage state information can be controlled to only allow access by an authorized backup policy procedure, using the standard system-implemented access control mechanisms. The policy procedure itself, being an administratively maintained component of the system, can be audited to insure that it makes proper use of the kernel interfaces to which it requires access. This architecture partitions (protected) mechanisms and (administrative) policies into separate security environments (e.g. kernel vs. non-kernel), and still permits a flexible user interface without compromising security.

After the policy implementation is invoked, there will usually be requests for backup operations placed in the request buffer. These requests will be translated into specific operations for accessing and controlling the backup storage facility, which are discussed in the next section.

---

(5) In order to prevent different users from using incompatible private policies for a given file, an access control list specifying "backup" permission for certain users will be associated with files backed up by users. Otherwise, private backup techniques for guaranteeing consistency among backup copies (of a database, for example) could be subverted.

4.4 The Backup Storage Facility

The key reason for using a network for backup storage is that it provides access to a large, inexpensive, remote, random access data storage facility. Most large random access storage facilities are expensive. However, just as the timesharing concept made large computer systems available to many users by sharing expensive resources, in a similar way the Datacomputer makes large random access storage available to hosts on the ARPAnet. Timesharing led to more general resource sharing, and the Datacomputer holds the possibility for more extensive sharing of data in a network community. In this section, however, we will be concerned with only the file system-like attributes of the Datacomputer.

The purpose of the backup storage facility is to retain information for possible retrieval at a later time. In the organization of file system storage, there are usually two classes of information. There is the file itself, and there is descriptive information about the file (e.g. author, length, time of creation, access capabilities, location in physical storage, etc.), maintained in the catalogs. Backup storage organization will be divided into these two classes of information for each file. There will be the contents of the file, and an associated descriptor. This separation and

the random access nature of the storage facility allows independent accesses to the file contents and to the attributes of the file. Structural changes to the cache file storage are represented by modifications to catalogs, and will be reflected in backup storage by changes to the descriptor. Cache file storage content modifications will result in sending a copy of the file contents to be stored in the file on the Datacomputer. Any access to a descriptor will be called a status operation, and any operation that writes a file in backup storage will be called a copy operation. Status operations result from status control requests from the policy procedure. The policy procedure may also make service requests, which result in the informational content of a data file being transmitted. The copy operation is one of two types of operations for carrying out service requests. The second type, the retrieval operation, is discussed in the next section. The types of requests and what types of operations they result in are illustrated in figure 4.4.

The component of the backup system that manages storage on the Datacomputer is called the I/O Control component. It interfaces cache file storage and policy requests in the local system, with remote backup file storage on the Datacomputer. Communication between the I/O control process and the Datacomputer is accomplished using a data management language called Datalanguage [CCA 73]. Local backup requests are translated into appropriate Datalanguage statements by the I/O control process, which sends them to the Datalanguage interpreter process on the Datacomputer.

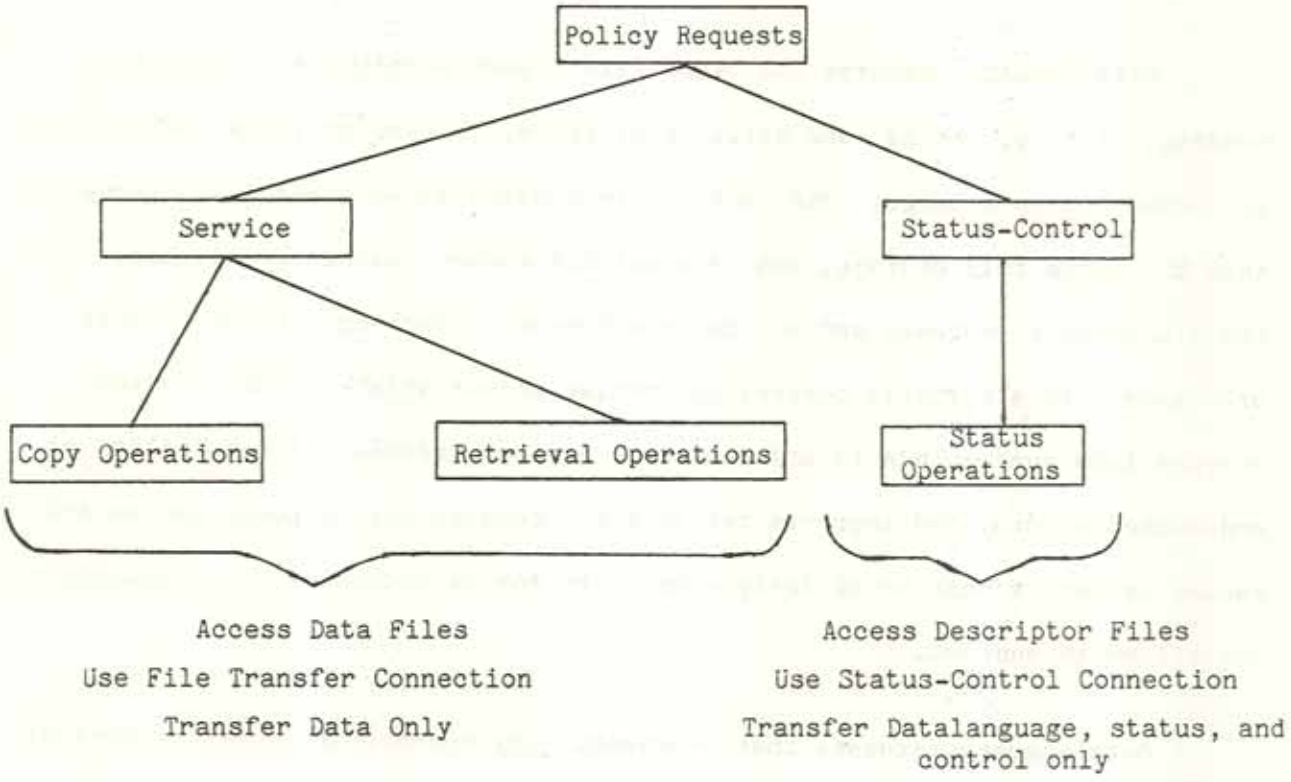

Figure 4.4 Request Types and Corresponding Operations

4.4 The Backup Storage Facility

Datalanguage supports the usual file system primitives of creating, reading, writing, naming, and deleting of files, (6) and provides the only way to access backup storage. The backup file storage is accessed less frequently than the cache file storage, and in a uniform manner (using Datalanguage) by the I/O control process, and not in an arbitrary reference pattern by user processes. This strictly controlled precise access strategy makes backup storage less susceptible to unintentional access, reduces the possibility of undetected errors, and improves reliability, because Datalanguage syntax and backup protocols must be strictly adhered to for Datacomputer file management operations to succeed.

Datalanguage requests that represent copy operations cause the contents of a local cache file to be sent over the network, to be stored in a backup data file. Status operations result in local catalog information being sent over the network to descriptor files, or retrieval of (status) information from descriptor files. Actual I/O operations are performed by the kernel, and are controlled via a kernel I/O interface to which the I/O control process has

---

(6) Other more sophisticated data management capabilities are also supported in Datalanguage, and although they are not essential for operation of the backup system, they can be quite useful. For example, to obtain a list of names of all backup files created by a given user, all descriptors could be read to search for the appropriate ones. However, Datalanguage can effectively treat all descriptors as a database, perform a keyed search of the database, and return only the requested information. This allows database management-type processing to be done by the Datacomputer concurrently with any other local processing, and only the requested results need be sent over the network.

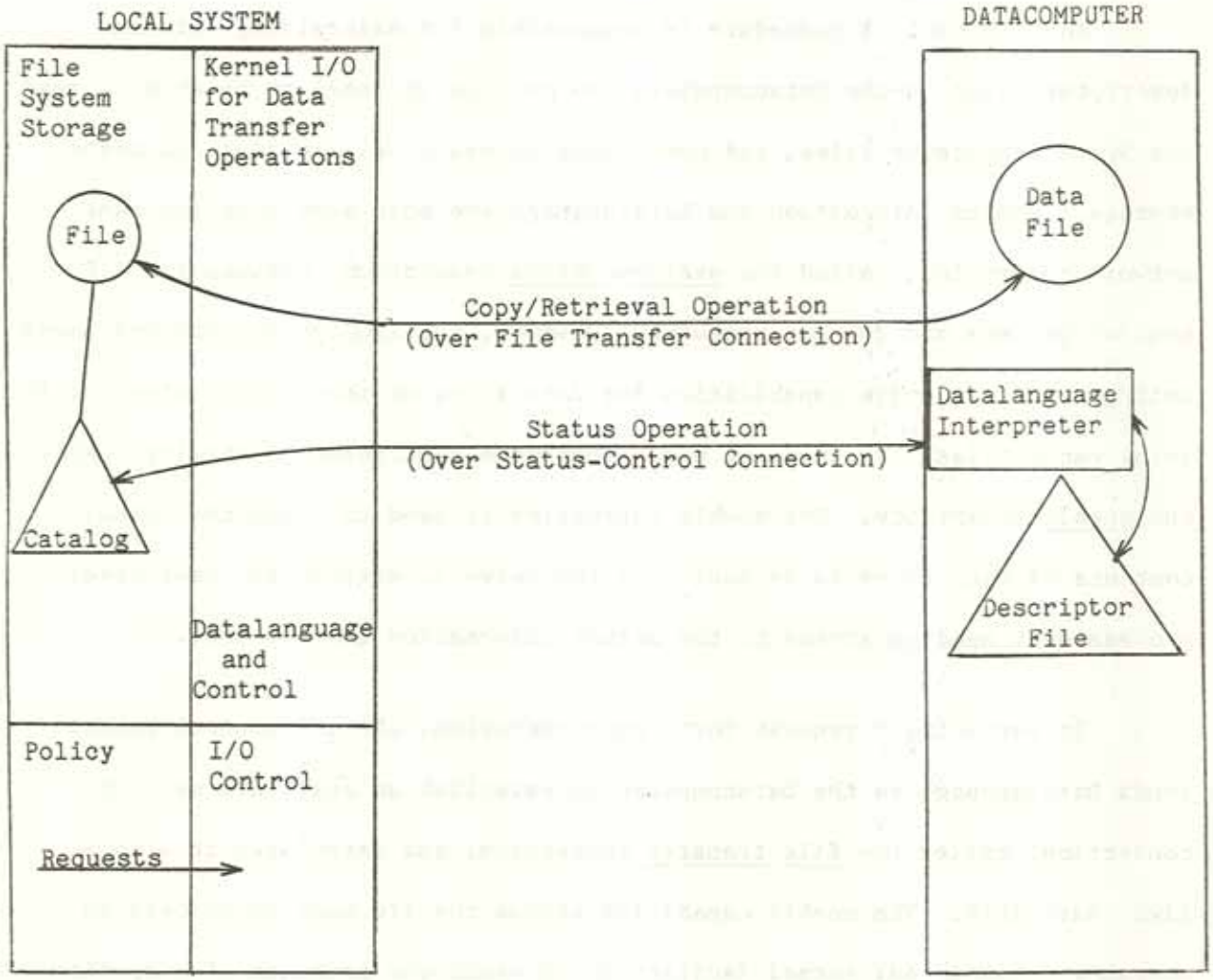

Figure 4.5 I/O Control of Data Transfers

access. These operations are diagrammed in figure 4.5, and discussed further below.

The I/O control procedure is responsible for maintaining data and descriptor files on the Datacomputer. To do this it needs to be able to read and write descriptor files, and hence have access to all catalogs in cache storage. Status information and Datalanguage are both sent over the same network connection, called the status-control connection, between the I/O control process and the Datacomputer. However, the I/O control process needs neither read nor write capabilities for data files on the Datacomputer, or for local cache files. Instead, it has a much more restricted capability, called the enable capability. The enable capability is used to cause the actual contents of data files to be sent over the network, without any user-level process ever needing access to the actual information that is sent.

In servicing a request for a copy operation, the I/O control process sends Datalanguage to the Datacomputer to establish an auxiliary network connection, called the file transfer connection, and associates it with a given data file. The enable capability allows the I/O control process to access a (restricted) kernel facility which sends the contents of a specified file over a specified network connection. At the conclusion of transmission, both ends close the connection. The progress of the transmission can be monitored by the I/O control process over the status-control connection, which

can be used to update the descriptor file at the end of transmission. Thus, the I/O control process, which has the responsibility and authority for maintaining descriptor files and requesting data transfers for data files, has only the capabilities for doing just that, and must rely on the protected kernel interface (and its own enable capability) for actually transmitting the data. Similarly, the kernel cannot transmit any data unless the I/O control process has enabled a file transfer network connection with the Datacomputer, and informed the kernel about this connection. Thus, both the kernel and the I/O control process must cooperate with each other before any access to the data file on the Datacomputer will succeed.

The backup storage management strategies described above illustrate the precise, deliberate control that is necessary to effect changes in backup storage, and also show the need for cooperation that tends to insulate backup file storage from the effects of unintentional attempts to cause changes. It might be observed, however, that while only the kernel can send and receive the contents of data files, user-level process can access descriptor files in all ways. In the implementation of the Datacomputer, an access capability is required in order to access files, and by providing only the I/O control process with this capability, other user-level processes will be unable to access backup storage. On the other hand, with this capability, the I/O control process is totally responsible for the management of data and descriptor files (but not the contents of data files). To protect remote data from unauthorized accesses that succeed, the kernel enciphers the data before

sending it. Data encryption will be discussed in more detail in the next chapter.

So far, we have discussed only one side of the Datacomputer management story -- local to remote storage. Discrepancy detections result in information transfers from local files to remote data files, and from local catalogs to remote descriptor files. The ability to use the backup storage facility in the other direction -- remote to local storage -- is primarily based on how useful the descriptor information is. One could envision elaborate descriptor management strategies for maintaining numerous versions of files, archival storage for files deleted from (logical) file storage, etc. Such elaborate schemes, although realizable as descriptor management strategies, are beyond the scope of this thesis. Instead, we are interested in reliability issues, which in the context of descriptor management, involve recovery strategies for cache file system damage. This will be discussed in detail in the next section.

4.5 File System Storage Recovery

During normal system operation, users initiate immune files into their address spaces, by referencing them through a catalog. If the cache file storage is damaged, some catalogs and files may no longer be accessible. When damage is detected, the system is made unavailable to users until the extent

of damage can be determined and appropriate recovery operations can be completed. This section discusses different degrees of damage that might occur, and how to recover as rapidly as possible. The cache memory view is the basic perspective from which recovery will be described.

A basic assumption is made that catastrophic failures are rare, and the purpose of the backup system described here is to facilitate recovery from more frequent but less extensive failures. (7) The extent of damage can sometimes be confined by careful organization and management of file storage. For example, redundant copies of the root in a hierarchical file system may prevent damage to a few bits from precluding access to the entire file storage contents. While such techniques are useful, they will not be discussed here. Instead, it is assumed that a recovery mechanism exists for the case in which extensive damage occurs.

Certain contents of file storage are more valuable than others, in the sense that their loss or damage has a more widespread effect. One example is the root, or other high-level catalogs in a hierarchical file system. Shared files comprise another example. In the general purpose computer utility, many system files are shared. In fact, many, such as the system itself, accounting programs, editors, etc., are used by every (or almost every) process. These commonly used shared programs and data files are required for normal system

(7) A performance analysis in the next chapter will give a better feel for the tradeoffs that might be reasonable between the degree of a failure, the recovery time, and the resolution supported by the backup facility.

operation, and are usually grouped in a system library. While these files are accessed frequently, they are modified relatively infrequently. On the other hand, user files are generally modified more frequently, and shared less extensively. The larger the group of users for a particular program, the more stable it tends to be, since it has been debugged and will be modified only occasionally. The same may not be true for data files, however. In general, there is a negative correlation between frequency of modifications to a file, and the number of users who access it, as illustrated in figure 4.6. The loss of frequently modified files would affect only the few users who access it, while loss of rarely modified files might affect all users. It is the former class of files that the network backup system is designed for. If a system library is damaged, it is assumed that it can be recovered rapidly and easily, for example, from tape, since it is a relatively stable set of files.

Availability of the system libraries is assumed to be a prerequisite for system availability. Another assumed prerequisite is the presence of all catalogs in the file system cache storage. After damage is detected, the system is made unavailable. If system libraries are damaged, they are repaired or restored by some local mechanism (perhaps using the network backup facility, but not necessarily if that would take too long). Damaged files are repaired, otherwise they are removed from the cache, and the catalog marked accordingly. Damaged catalogs are repaired or restored, and then the system is made available to users.

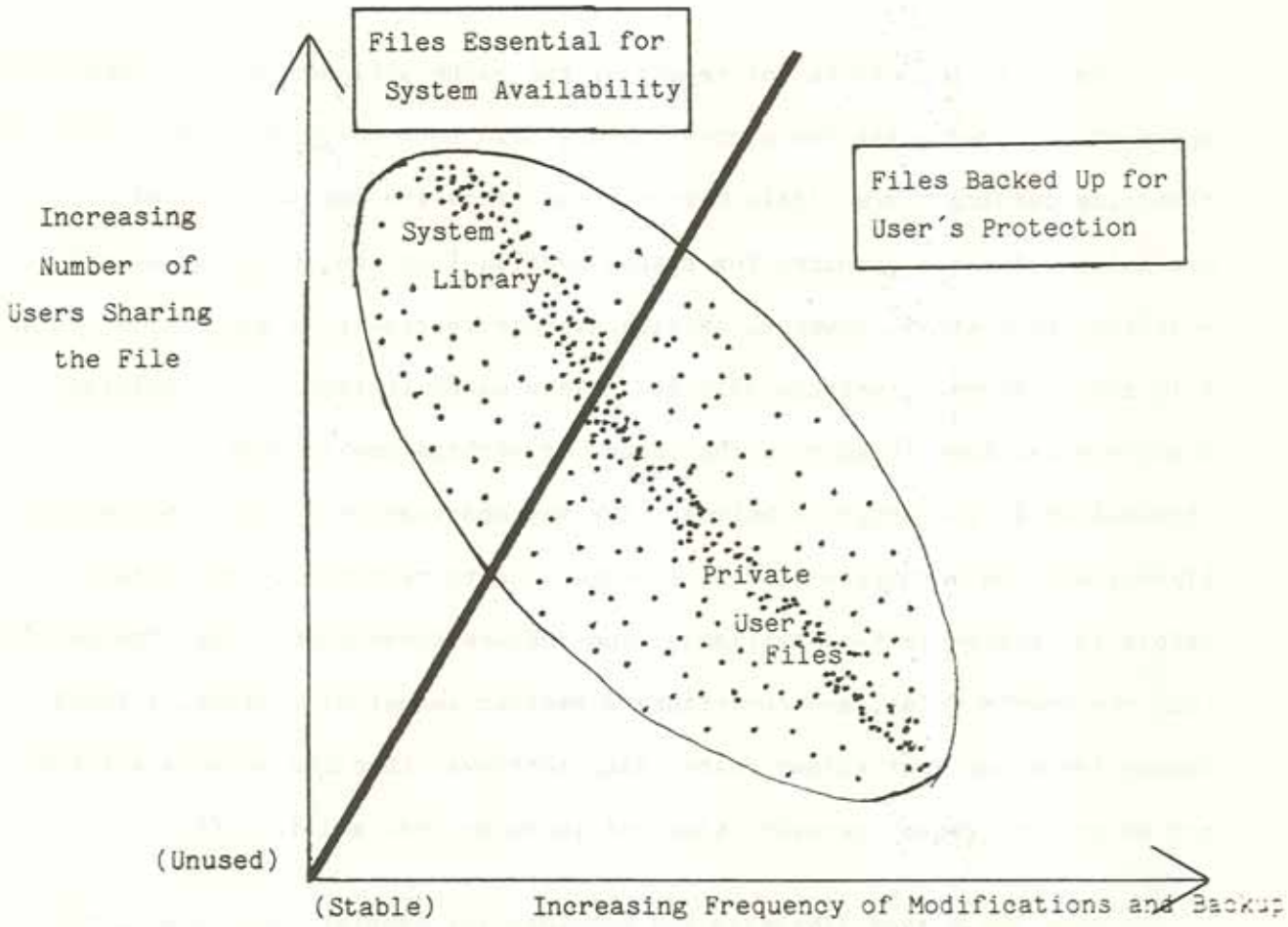

Figure 4.6 Correlation Between File Stability and Extent of Sharing

4.5 File System Storage Recovery

Specific techniques for repairing the cache will not be discussed here. Approaches to restoring the system library have been mentioned. The issue of restoring catalogs is a little harder, since they are not only shared extensively and are required for system availability, but they may also be modified frequently. However, catalog storage represents a minority of total file system storage, perhaps only 10%. This makes it feasible to maintain duplicate catalogs locally in the cache, or perhaps use another high-reliability storage technique. The key observation is that the system library and the structure of file storage need to be complete and intact before the system is made available, and because these change less frequently than the user's files, and constitute a smaller amount of storage, a local backup technique that allows faster data retrieval than the network affords can be used to reduce recovery time and increase availability. (8)

Once the system libraries and catalogs are complete and intact, the system can be made available to users. However, there may be files missing from the cache that are backed up on the Datacomputer. When a user tries to initiate such a file, the catalog entry will indicate that it is not in the cache. The file system will place the filename on a missing file report list and will signal the policy procedure to request retrieval of the file, by the

---

(8) These local backup techniques could use the network, but it is assumed that this would be slower than using an extra local disk, or tape storage for the system libraries. Techniques for restoring libraries and catalogs are described in [St 74].

I/O control process, from the Datacomputer. The sequence of events is described below, and the steps are keyed to the picture in figure 4.7.

Reference (via the catalog) to a file that is missing from the cache (1) causes an entry to be placed on the missing file report list, and generates a missing file fault notification to the policy (2). The missing file report is processed (usually with high priority), and a retrieval request is sent to the I/O control component (3). The I/O control component establishes appropriate network connections for retrieval of the backup copy of the missing file from the Datacomputer (4).

Retrieval operations occur "on demand" as missing files are referenced. The effect seen by the user is a delay in referencing the file, since it is stored on the Datacomputer. However, the file may not be the most recent version, since the failure causing the damage may have resulted in unrecoverable loss of information. The discrepancy detection and notification strategy described earlier attempts to minimize this undesirable effect by tracking changes dynamically so that resolution is minimized and backup copies can be made as closely as possible to the time of the actual modification that caused the inconsistency. Similarly, retrievals are made as closely as possible to the time of the reference to the missing file. The missing file processing technique resembles the normal operation of a multi-level memory system, and fits the cache file system model introduced earlier.

4.5 File System Storage Recovery

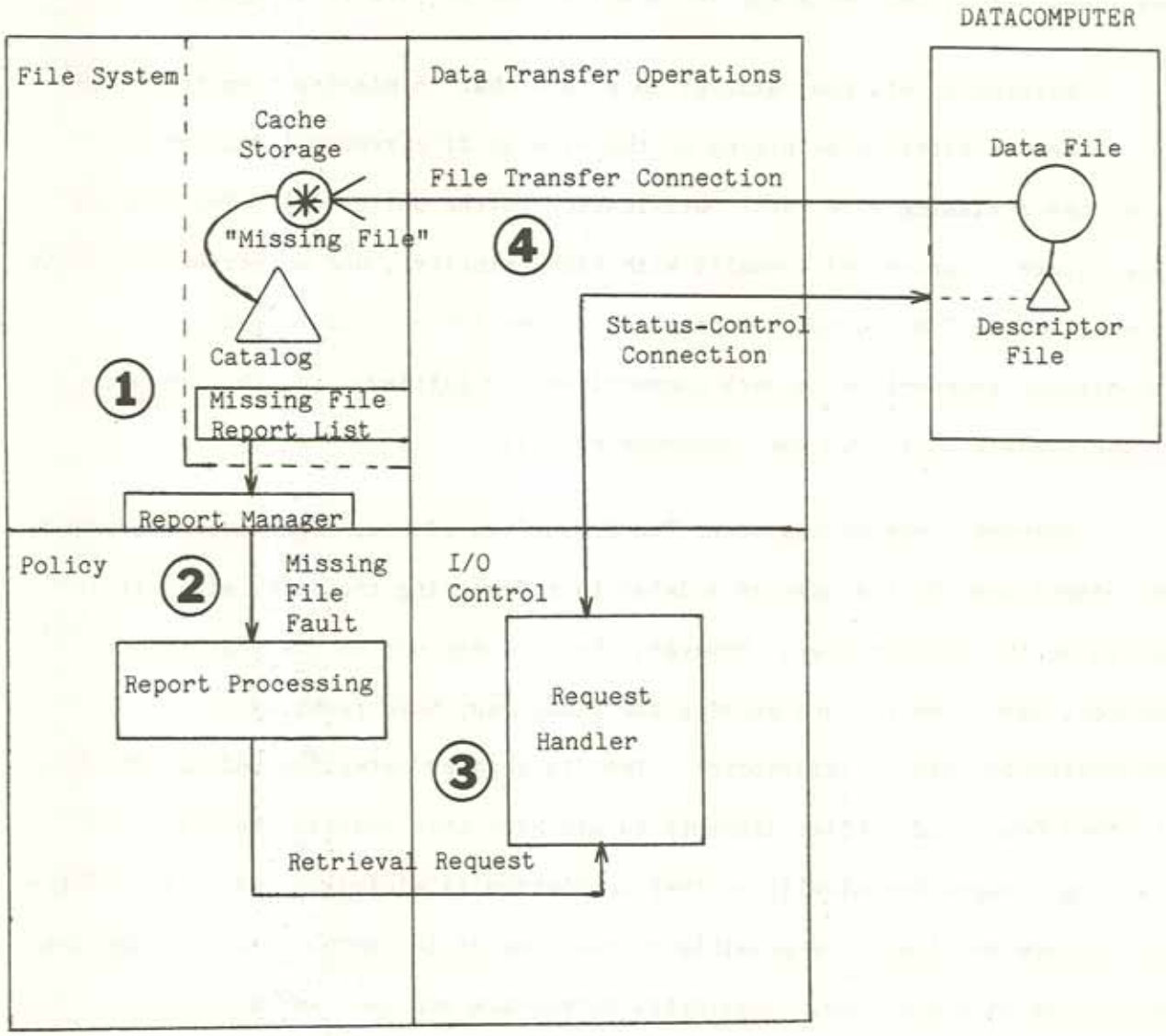

Figure 4.7 File System Storage Recovery

4.5 File System Storage Recovery

Files that are not yet referenced may also need to be restored. These can be retrieved at a more leisurely pace, perhaps when no demand-retrievals are in progress. These retrievals will make subsequent reference to the file faster for the user, since the file will already be in the cache. Strategies for background retrieval of unreferenced files can be imagined by the reader, and will not be discussed here.

A retrieval operation can be the result of a missing file fault, or it could result from a background retrieval request. Retrieval operations originate from within the file system, and cannot be made by user processes. They are therefore secure. They are handled in a similar fashion to copy operations. A data connection is enabled by the I/O process, and the file system is then able to receive the contents of the file from the Datacomputer. The connection is closed, and the file system removes the missing file tag in the catalog, and proceeds as if the file were there when originally referenced by the user.

Under the assumption that damage will usually not be widespread, the rate of retrieval requests when the system is first made available will result in a small, but not unmanageable delay in accessing the missing file. Given the alternative of system unavailability, this is a bearable condition because it still allows some functioning at a reduced level. Depending on user reference patterns, the delay in access will decrease as retrieval requests

become less frequent, and more files are restored. Typically, most users should not experience any delay, especially if the extent of damage is confined to only a few catalogs and/or files. The philosophy is to provide full service for most users and degraded service for a few, rather than no service for any until full service can be provided for all. The effect is to improve the overall reliability of the file storage system.

As a final note, it is suggested that the file retrieval mechanism could be generalized by an appropriate implementation of the report processing module in the policy component. Such a generalization would not only allow backup copies of files to be retrieved from arbitrary storage facilities (e.g. tapes, other nodes in the network, etc.), but would also provide a mechanism for extending the usefulness of the local catalog. Such an extension, for example, could be used to implement a distributed file system in a network, in such a way as to allow uniform access to distributed files. This would make all files appear to be local to the user. With this suggestion, the possible implementations will be left to the reader's design.

The interactions among the file system, the policy procedure, the I/O control process, and the retrieval mechanism have been described in order to give an overall view of the network backup facility. The goals have been to minimize unrecoverable loss of information and minimize system unavailability by a secure and flexible strategy. Some of the issues discussed in detail pertain to backup facilities in general.

4.5 File System Storage Recovery

There are still several issues pertaining to a network implementation that have not been presented. Problems in naming files and in keeping backup copies consistent are found in backup facilities in general. However, the possibility that specific components in a network backup facility might fail leads to techniques of maintaining (temporarily) multiple backup copies on different systems. Protection of remotely stored data, performance of the network implementation, and how to charge for backup services are also topics yet to be discussed in detail. The next chapter will concentrate on these issues as they relate to the use of a network for carrying out backup operations.

4.6 Summary

This chapter has discussed in detail the components of a backup system. Each component has a well defined function and interface to produce a secure and flexible system.

The discrepancy detection component reports file modifications to a policy component, which processes discrepancy reports and then requests backup operations. The I/O control component, in cooperation with the kernel, maintains backup copies on the Datacomputer in servicing requests from the policy component. Data and descriptors are maintained independently, and only the kernel needs to access file contents to transfer them over the network.

4.6 Summary

Recovery after a failure occurs in two steps. First all catalogs and system libraries are restored, and then users are allowed on the system. As they reference files, ones that are missing from the cache are retrieved from backup storage automatically. The result is a slight delay in accessing missing files, but an improved recovery time and greater system reliability.

Chapter Five

The Network Implementation of the Backup Facility

5.1 Introduction

The previous chapter has described the local operation of a backup facility, but has not considered the issues that arise specifically in a network implementation of a backup storage facility that interfaces to the local system. This chapter presents a discussion of these issues, some solutions, and a comparison with alternative approaches.

The ability to name objects of interest is essential in order to access them. Names for objects that exist in a network environment are discussed in the next section. Specifically, these objects are network connections used for communication, network hosts which provide file storage facilities, and backup copies of files.

In making backup copies, it may be required to produce copies of a file in a known state. This is important for guaranteeing consistency of a set of files that comprise a database. By first making a consistent "shadow copy" of a database, subsequent backup operations can reference this consistent copy

while users can continue to modify the database, without producing inconsistent backup copies. The method by which shadow copies are modified is analogous to the method for updating immune files from susceptible files.

The purpose of the backup facility is to improve reliability, but the reliability of the backup facility itself depends on the reliability of the components from which it is constructed. In the face of unreliable operation of network facilities for data access and storage, the backup facility must be able to take an alternative approach to continue providing service. A technique using several network hosts to store backup copies of files is investigated.

When several distributed hosts are used to store data, there are problems in locating, accessing, and keeping copies consistent. Distributing storage facilities so that backup service is not terminated if one (or a few) such facilities should fail may result in multiple distributed copies of data. However, the nature of access needed for backup purposes allows a special type of "distributed file system" to be implemented. The maintenance of distributed files in the network is compared with maintenance of copies using a tape system. The network implementation is shown to be more flexible.

By storing data at a remote facility which can be accessed only remotely, one loses the ability to locally control access and protection of the data. To provide some confidence that the use of a remote data storage facility will not circumvent local protection strategies, the use of

encryption is described whereby the degree of protection provided is maintained under local control.

The use of a network for data communication imposes bandwidth constraints on the ability to move large amounts of data frequently and quickly. A queueing model is presented and an analysis given to show that by careful selection of requests for backup service (which can be accomplished by an appropriate policy implementation), the level of performance realized by the limited bandwidth network backup implementation provides improvements in reliability over a tape system, without undo sacrifice in performance when recovering from a failure.

Finally, some ideas on charging for services are presented. Backup can be considered as an example of general services provided for users. Special services are charged at a rate proportional to the amount of resources required to provide the service.

In the next chapter, generalizations of some of the ideas presented in this thesis will be suggested as feasible extensions of the backup facility for solving harder problems. Of main interest is the problem of distributed file systems, which includes all the issues discussed in this chapter.

5.2 Naming of Objects in a Distributed Environment

There are usually two types of names for objects accessible to users in a computer system. An implementation-oriented name is often used by the primitives which manage the object (e.g. storage pointers, UID's). A user-oriented name consisting of character strings is available as a convenience to the user, so that character string names must be mapped into implementation-oriented names in the actual access to the object. Examples of objects maintained in a computer system include files, processes, principals, catalogs, and domains. Names of interest in a network-oriented backup facility are for file and network-related objects. These are discussed in this section.

Every file in the cache file system is named with a UID in a catalog. In addition, the catalog implements a mapping from user-oriented character string names, to the corresponding UID name. This mapping is reliable because the system is assumed to provide a backup mechanism for catalogs, which implement this mapping. The UID of the cache file is used to derive a unique character string name for the backup copy of the file on the Datacomputer. The character string name by which the user knows the file is not used. This strategy prevents the disclosure of catalog information (the user's character

string name for the file) in a non-kernel environment (e.g. the policy implementation), because only the cache UID-derived character string name is used by the policy implementation and the I/O control process. (9) No accesses to local files or remote backup copies of files need the user's character string file name, except when referenced through a user interface. In this case, an authorized user may map a character string name to its corresponding UID, and the UID in turn is used to request backup operations, as shown in figure 5.1.

This uniform naming convention allows objects to be named from user environments, kernel environments, and over the network in another file system environment, while controlling access to the name mapping function of the file system by the standard access control mechanism in the local system. All accesses to files will be by UID name, which can be obtained from the file system mapping function by an authorized user. The key point is that the inverse mapping from UID to user-oriented name is never required, because only the UID name is used for requesting backup operations.

Associated with each file on the Datacomputer, there is a descriptor file, also stored on the Datacomputer, which, by convention, is named "UID.descriptor" for data file "UID". The descriptor file may contain

---

(9) The file system provided by the Datacomputer also maintains its own implementation-oriented UID name for files stored there. These files are not accessed by the I/O control process by the Datacomputer defined UID name, but by the character string name, which is derived directly from the UID for the cache file.

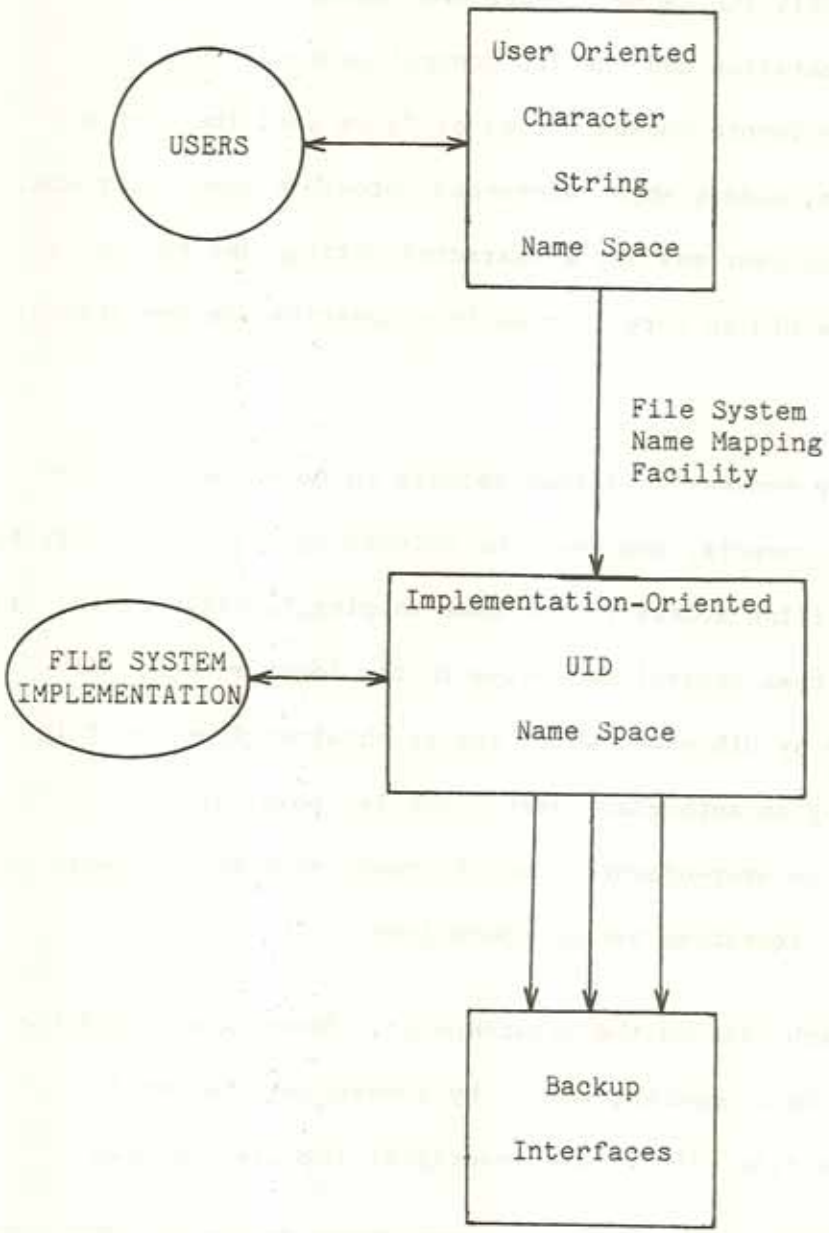

Figure 5.1 File Name Usage

character string names for the associated data file, as a facility for redundantly storing catalog information, but these names are not essential for backup operations, only useful. For example, if local catalogs are damaged, information in descriptor files can be used by a recovery facility to restore catalog information over the network. Furthermore, it was indicated in an earlier footnote that the Datacomputer can treat the set of descriptor files as a database describing the contents, structure, and state of file system backup storage. Database operations that would be time consuming to perform in the cache file system (e.g., inspection of every catalog entry) could be done faster, and concurrently with other operations, by an appropriate database management request to the Datacomputer. Such uses were hinted at earlier, and numerous applications can undoubtedly be imagined by the reader.

Another name that is needed in the network backup facility is for the network connection object. As described earlier, there are two types of network connections: the permanent status-control connection for sending and receiving Datalanguage and catalog/descriptor information, and the temporary data connection for sending/receiving the contents of data files. A connection is named by a pair of complementary port names. A port is a sender or recipient of data on a network connection, and is named by a host-socket pair, uniquely identifying the host, process, and socket number for the port. A connection consists of a sending and receiving port. The status-control connection is actually a pair of connections, between a local send and receive

port, and a Datacomputer receive and send port, respectively, allowing two-way communication between the I/O control process and the Datalanguage interpreter. The port names used for establishing the two connections are negotiated via the ARPAnet initial connection protocol (ICP) [ARPA 74].

Data connections are established differently. For copy operations, a send data connection is established using the enable capability of the I/O control procedure. The Datacomputer associates a receive port name with a data file, and this port name is sent to the I/O control process over the Datalanguage connection. This foreign port name is then sent to the kernel I/O facility, which establishes a simplex connection, sends the contents of the requested file to the foreign port, and closes the connection. The Datacomputer takes all incoming data and places it sequentially into the data file previously associated with its receive port, and then closes its side of the connection. Note that this type of connection is set up under control of the I/O control process and the Datacomputer using the status-control connection, and requires the cooperation of the kernel for transmitting the data. This description illustrates the separation of a privileged data transfer facility and a trusted port name management facility, and the mutual cooperation required for moving data.

The retrieve data connection for retrieval operations is set up in a similar way. The Datacomputer informs the I/O control process of the send port name it will use to transmit the data. This port name is passed along to

the kernel, which reads all data from that port into the appropriate file, and both ends of the connection are closed. In both of the above operations, no data file contents ever pass through the I/O control process; only port names are necessary to enable data transfers between cache files and backup files (through port interfaces) over the network.

For most purposes, the mechanisms and facilities described up to this point will be sufficient for normal backup operations. However, if there are consistency constraints on sets of files, a special user policy implementation is required to define those constraints. The issue of consistency is described in the next section.

5.3 The Consistency Issue for Backup Copies

For most user files, consistency means that the copy created on the Datacomputer is identical to a (recent) copy in the cache. This may not be the case if the file is modified while it is being copied. However, such a modification will cause it to be copied again later, so it will eventually result in a consistent copy.

The harder consistency problem arises in the case of multiple files with a defined relationship among their separate contents. The default backup policy does not allow an arbitrary user specification of the relationship

defining the consistency constraint among several files. A user implemented policy permits this specification, however. In fact, a user implemented policy is necessary for specifying consistency constraints, because the system has no way of determining the constraints the user wishes to impose on a set of data files.

Upon notification that a component of the user's database (i.e. a user's file) has become inconsistent with respect to a backup copy, the user policy implementation can decide whether or not to request backup operations for the database, based on database state information. If it is desirable to backup the database, it is likely that the state would change before actual copies could be written out to the Datacomputer. Since the state of the database at the time of the copy operation could be different (and inconsistent) than at the time of the discrepancy notification, it would be desirable to have some control over the state of the database at all times. The user policy implementation can include this control.

At the time of the notification, the user's decision module may decide to backup the database, if it is consistent at that particular time. The consistency constraint requires that the database be in the same (or at least a consistent) state at some later time when the copy operation is actually to be carried out. To guarantee this, it will be necessary to make a shadow copy of the database, and request a backup operation for the shadow copy. The shadow copy is a consistent snapshot of the database at the time the decision

to perform backup was made. When the actual backup operation is performed, the shadow copy will still be consistent, and will be copied to the Datacomputer instead of copying the real database. Completion of the copy operation will be communicated to the user, so that the time that the latest consistent backup copy was made, and the identity of the shadow copy from which it was made, will be known, and the shadow copy can be disposed with as the user process sees fit.

It is possible that several discrepancy notifications may be received before a copy operation for a shadow copy is completed. If backup is requested for each one, then old shadow copies that have not yet been copied to the Datacomputer can be superseded by new shadow copies. This results in a more recent, but still consistent copy being written out. The observation being made is that shadow copies are analogous to immune files, but are maintained by the user. This means that updates to shadow copies are made by explicit, controlled, well defined operations that the user policy implementation specifies in order to enforce the database consistency constraints. Backup operations then reference the shadow copy instead of the real database, just as they reference immune files instead of the susceptible files in the address space of a user. Disposition of shadow copies is also handled by the user policy implementation.

5.3 The Consistency Issue for Backup Copies

Whereas an immune file might be updated during a copy operation, making the backup copy inconsistent, updates to shadow copies should not be permissible during a copy operation. To prevent copy operations from commencing while an update is in progress, or to prevent an update from occurring while a copy operation is in progress, a locking scheme is required for the database. (10) This information enables the user to know at all times and with absolute certainty, the state of the database, the shadow copy, and the backup copy. This knowledge is essential if any sense is to be made from a backup copy of the database that is later retrieved from the Datacomputer. The fact that a consistent copy is retrieved, along with the time that the shadow copy it was made from was updated, will enable the user to determine the state of the retrieved database, and perform appropriate recovery operations for unrecoverably lost data.

Any ability to recover from a failure depends on the reliable operation of the backup facility and the resources it requires for its implementation. In a network environment, communication services are required, but may not always be reliable or available. The issues of reliability pertinent to the network backup implementation are discussed in the next section.

---

(10) When the request processing component encounters a shadow copy locked by the user, it should not wait on the lock. Otherwise, a user could indefinitely deny backup service to all other users. Instead, the request could be re-queued, or checked again later, etc. Note that a user could make separate shadow copies instead of performing updates, but the economics favor the ability to replace old shadow copies while only postponing the request for backup while they are locked for updates. The reliability of only the locked database is affected if backup is delayed by a malicious user. Service for other files is not affected.

5.4 Reliability in the Network Environment

The purpose of a file backup mechanism is to improve the reliability of file storage. This goal can be reached only if the backup facility itself is reliable, i.e. if it is available to operate correctly when required. In the network backup implementation, resources that are local and resources that are remote are required for backup operation. Local resources include those facilities of the operating system that are shared by all users (file system, address space, etc.) and will not be discussed here. The use of remote resources makes the network implementation quite different from the traditional local tape backup implementations. Whereas the tape medium is accessed locally, but can be removed so that data can be stored remotely to protect it from local perils, the Datacomputer storage facility is accessed remotely, and the information is stored remotely. The key issue to be discussed in this section is based on the observation that in the network environment, access to the data is not under complete local control, but depends on the reliability of remote resources.

5.4 Reliability in the Network Environment

One essential resource required by the backup system is a usable network connection to communicate with the backup storage facility (the Datacomputer, in this case). Such a connection may not be available for a number of reasons. The network itself may not be operational. We assume that this is rare, and when it does occur, it will be only temporary. Another possibility is the failure of the local network software (e.g. the Network Control Program). This failure is local and will not be discussed here. The remaining factor determining the ability to maintain a usable connection is the availability of the Datacomputer itself.

The Datacomputer is a hardware/software system that is prone to all of the problems of system reliability discussed earlier. As a network resource, it will be assumed to be less reliable (less available) than a local tape medium, (11) since many factors determining its availability are not under local control (e.g. a remote power failure might affect the Datacomputer, but not the local system). However, in a network, while a remote resource may exhibit reduced availability for a specific instance of that resource, the underlying ability to share resources around the network makes more instances of a particular resource generally available [Cos 75, Cro 75, Rob 70, Th 73].

---

(11) Since magnetic tape is prone to many types of errors from aging, deterioration from storage in poor environments, etc., it is less reliable than the Datacomputer for storage of data, but being local, is almost always available. Conversely, data storage is assumed to be very reliable in the backup storage facility, but the facility will occasionaly be unavailable for access to the data. The tape problem can be solved by storing many copies (locally) on different tapes. The availability problem for reliable network data storage facilities is the issue addressed in the text.

5.4 Reliability in the Network Environment

The overall availability of a backup storage resource, such as implemented by the Datacomputer, is not necessarily lower than with the tape system. In fact, availability may be improved if resources are shared widely enough. The resource management problem then becomes more significant if several resources are shared among several users. Some ideas about network resource management in the context of reliability will be presented in the remainder of this section.

While some action can be taken to recover from local failures, not much can be generally done to affect remote recovery. However, the network backup facility can use storage facilities other than the Datacomputer. In an extreme case, it could revert to writing copies of files onto tapes, as in the traditional system. In a mild case, it might be possible to use another instance of the Datacomputer as the backup storage facility. In a more general case, any network file system might be usable, as long as it can provide the essential primitive file system functions of naming, storing, recalling, and deleting (unnaming) data. The explicit knowledge of where the copy of the file is actually stored is not relevant to the operation of any component of the backup facility, except the I/O control process.

The I/O control process needs to know the identity of the network backup storage facility in order to establish the status-control connection according to the initial connection protocol. In the normal operation of the file backup system, all backup copies are assumed to reside on a primary

backup storage facility (e.g. the Datacomputer). If this facility is not available, some strategy for choosing an alternate storage facility is invoked. For example, a simple list of alternates can be used, and if none are available, a tape system can be reverted to. One such alternate facility might be another Datacomputer. The same I/O Control process would be used for performing (Datalanguage) backup operations, but the file system would have to record which Datacomputer was being used to make the copy. In the most general case, backup copies would be made on an arbitrary system, for example, a Multics, a TENEX [Bob 72], etc. This is feasible to do, because only four primitive file system functions of naming, storing, recalling, and deleting data are required. These primitives are available on most systems, and are independent of the structure of a particular file system. No information about the structure of the local file system needs to be reflected in the structure of the backup storage facility. A hierarchical local file system becomes a "flat" backup file storage structure. Optionally, descriptors can include structural information about the local file system. However, catalog information and structural reliability are both assumed to be maintained by the local system. The descriptor allows the option of using backup storage facilities on the network (e.g. database management facilities on the Datacomputer) for managing catalog information and improving catalog storage reliability.

5.4 Reliability in the Network Environment

For each type of file system, an I/O control process can be used to translate policy requests into the specific primitives provided for maintaining backup copies, for example, by the translation to Datalanguage described earlier. Either each file system could support a Datalanguage interpreter interface, and the same I/O control process could be used (the remote solution), or more likely, a separate I/O control process could be designed to interface to each type of file system (the local solution). The possibility of using several different storage systems for maintaining backup copies introduces naming, consistency, and multiple copy problems [Boo 72, PeM 75].

The name of a backup copy is the same as the UID of the cache file in the local system. However, if a backup copy can reside on several storage systems, a mechanism is required to identify its "residence." A unique name of the host system providing the backup storage facility will be sufficient. Since many systems may share backup storage, the inclusion of the name of the "home residence" of the file will make the local UID unique throughout the network. Each system can maintain its own UID mechanism, but global names must include the identity (address) of the system that generated the name, so that two different systems using the same UID mechanism will not produce duplicate UID names in backup storage. The combination of the address and the UID name of the file are enough to locate and identify a file uniquely throughout the network. This global name will be called the network-wide

unique ID (NUID). (12) When a copy of a file is made, its NUID is recorded in the local catalog so that retrieval operations will be able to find the copy.

The retrieval of files is triggered by a reference to a catalog entry for the file, that indicates that it is missing from the local file storage cache. In addition, the catalog maintains the network address of where the copy of the file is actually stored, uniquely identified by the NUID. Since all catalog information is backed up locally, and all catalog information is available before the system is made available, all references to missing files can be resolved. An appropriate retrieval request can be processed by moving the file from its storage location (tape, network host, etc.) into the cache file storage. Essentially, the extended backup facility described here maintains reliable operation by implementing a distributed file system, with a centralized catalog and a local cache storage facility for efficient reference to the data. However, the use of more than one storage system will undoubtedly result in several versions of copies of a local cache file. The multiple copy and version problem must be solved to allow retrievals to access the correct copy, and to allow copy operations to proceed using the primary backup storage facility when it again becomes available. These problems are discussed in the next section.

---

(12) Forging of host addresses, either as local or remote residences of files, can be prevented if the communication network provides a secure mechanism for establishing the identity of communicating hosts. One host naming files for another is prevented by authenticating the identity of the host requesting backup. Authentication occurs within the network implementation, and prevents a malicious host (or user) from acting as an imposter.

5.5 The Multiple Copy Problem for Backup

The abstract concept of the file system is one in which only one real copy of the file exists, and the cache is used to facilitate access. Whether actual copies are distributed or not is irrelevant from an abstract point of view. Any new copy made on a particular system becomes the "real" file, and the fact that other copies that may now exist are deemed to be obsolete needs to be recorded. Such a mechanism could operate as follows. Normally, copies are made to the primary backup storage system, and the NUID of the copy is recorded in the local catalog. When the primary backup storage facility becomes unavailable, a new copy will be made on an alternate facility. However, the obsolete copy's address (the file's "former address") is also recorded with the descriptor on the alternate storage facility, and the current NUID is recorded in the local catalog. This technique can be applied iteratively if the alternate facility becomes unavailable. Eventually, the obsolete copy will become available again, and the (temporary) new copy on the alternate storage facility can be moved (by the backup system) back to its "former address." The goal is to cause the eventual migration of all files back to the primary storage facility. Copies on alternate storage facilities are meant to be only temporary, until the primary facility again becomes available.

5.5 The Multiple Copy Problem for Backup

The random access network backup facility is capable of maintaining a complete copy of file storage. The incremental and catchup dumps used in managing a less flexible tape system are not required. Copy operations simply update the complete dump to keep it current. However, if the primary backup storage facility is unavailable, the use of an alternate facility during this time is analogous to making an incremental dump. The migration of files back to the primary facility is similar to the consolidation operation of the catchup dump, and can use a similar mechanism for producing copy requests. The catchup dump and distributed file migration to a centralized location both attempt to facilitate the access to backup copies of files, especially when retrievals are required.

Retrievals will attempt to obtain the current copy, since it is identified by the NUID in the local catalog. If the storage facility containing this copy is not available, the retrieval cannot proceed. However, the other alternate storage facilities could be tried, to retrieve an obsolete copy. There is no guarantee that one exists, and the use of an obsolete copy may not be helpful for most users. Therefore, obsolete copies will only be considered for retrieval at the user's direction (for example, by setting an appropriate bit in the catalog entry for the file). The ability to maintain multiple copies in the backup system is similar to the archiving function mentioned earlier. Since the backup facility is not designed to provide an archiving facility (although it could provide such a facility with a suitable extension to the mechanisms presented here), the management of multiple copies in a general way will not be discussed here.

5.5 The Multiple Copy Problem for Backup

One way to describe the above mechanism is as a "backup for backup." This means that if a copy operation is temporarily disabled, another facility will "fill in" for the primary storage facility. This allows the ability to keep making up-to-date copies, but makes it difficult to find and access them when they are widely distributed. To help alleviate this problem, the "migration" of copies back to their primary homes was described. Migration results in an opposing force to centralize file storage. The result is that a primary backup storage facility failure makes recovery (retrievals) difficult. This case arises only if both the local system and the primary backup storage facility should fail at the same time. It is assumed that the reliability of both systems is approximately the same. Assuming that the probability of failure of either one is p (typically, p might be approximately 1%), if failures are not correlated, the probability of a simultaneous failure is on the order of p squared, which is considerably smaller than p.

However, even if a simultaneous failure should occur, the time to recover is increased only by the MTTR for the primary backup storage facility to recover, which might be about the same as the MTTR for the local system. Thus, the probability of simultaneous failure is p squared, while the effect of this rare event on recovery time is linear (2p). Therefore, the migration of files to a centrally accessed storage facility is reasonable for backup purposes. (13)

---

(13) For a general purpose distributed file system, this is no longer true, because there are usually as many read accesses as write accesses. The key

5.5 The Multiple Copy Problem for Backup

The management of multiple copies under the special access requirements for a backup system is much simpler than in a general purpose distributed file system [Cro 75, FaH 72, Th 73]. Also, the file system image does not support the concept of multiple copies or versions of files. Nevertheless, copies of files may from time to time be distributed among several different storage facilities. The diversity of system designs makes any uniform external protection mechanisms difficult to devise. Even if a protection mechanism were common to several storage facilities, or only one storage system were used, the lack of local control over the remote system environment is sufficient to erode confidence in the security and protection mechanisms. System programmers at remote sites could dump the disks, for example, if sufficiently motivated to do so. The need for some local control over protecting against the release of information that is stored outside the local system leads to the technique of data encryption. Encryption should make any backup data worthless without the key. The issues of local control over protecting remotely stored data, and the management of keys, are discussed in the next section.

---

difference that makes this approach feasible in a backup system is the fact that most accesses are write-once to a (virtual) single copy. Obsolete copies are only referenced to delete them, and current copies are read for retrievals much more rarely than they are written. This is another difference that would make an archive facility more difficult to handle.

5.6 Protection of Remotely Stored Data

In a traditional tape backup system, all access to information is performed locally, and backup copies may be stored remotely. In a network backup system, backup copies are both stored and accessed remotely. Once information leaves the environment of the local system, there is no guarantee that it will be protected to the same degree it was in the local system, against unauthorized release or modification. Furthermore, the storage facility itself is not under control of the local authority, and neither is access to the storage facility. This lack of control over the access to remotely stored backup copies necessitates a local protection mechanism to prevent security violations.

Access to the backup facility requires a capability, which is implemented at the remote system. Assuming that this implementation is secure, access can be controlled by possession of this capability. In the network system described, only the I/O control process will have the required capability. In addition, the enable capability is required for proper cooperation between the I/O control process and the kernel I/O facility. This is implemented in the local system. Only an I/O control process will have the necessary capabilities for accessing the remote storage facility.

5.6 Protection of Remotely Stored Data

Certification of the local I/O control process, by auditing the programs used in that environment, will help guarantee that proper use is being made of the backup facility interfaces. This is under local control, and will not be discussed further.

There is still the possibility that the remote implementation of the capability mechanism for controlling access to the backup storage facility can be subverted, by system programmers, for example. There is no guarantee that a remote system kernel is secure, and even worse, there is no way to find out what types of security violations are likely to occur. Stored information could be compromised in any of a number of ways. From the time that information is sent out over the network, it could be intercepted by an intruder tapping the communication lines [Ke 76], or it could be copied from a remote storage device by a stand-alone dump program. The solution is not to prevent such activities, but to make them harmless, by encrypting all data sent over the network and stored remotely.

There are numerous techniques that have been proposed for the encryption of data [Fei 73, Fei 75, NBS 75, Sm 72, Tu 73]. We will assume that the kernel I/O facility implements some encryption/decryption mechanism for outgoing and incoming data on the network connections. Rather than discussing the mechanics of encryption and decryption, we will concentrate on applications of encryption to protecting remotely stored data, and on the management of keys.

5.6 Protection of Remotely Stored Data

There can be two types of remote files that need to be protected. Data files contain the actual informational content, and an associated descriptor file contains attributes describing the file. Each type of file may need to be protected by a different application of encryption. In the simplest case, the use of descriptor files is not needed, since all catalog information is kept locally and reliability is enhanced by a local redundancy technique. Descriptor files are not needed, and no catalog information is ever sent over the network. To protect data files, they are encrypted using a system-generated unique key, which is stored in the (local) catalog entry for the cache file. When the file is transferred over the network, it is encrypted/decrypted by the kernel I/O facility, using the key stored in the catalog. Any information outside the control of the local system exists only in the encrypted form, and is as protected as the local encryption mechanism allows. The point is that the degree of protection is under local control, in the implementation of the encryption mechanism.

The encryption mechanism makes remote backup copies worthless without the key, if the work factor required to cryptanalyze them is high enough. There is no way to prevent inspection of backup copies by an unauthorized intruder, but the local encryption mechanism determines the work factor required for decrypting the data, based on the value of the data. Similarly, there is no way to prevent unauthorized modification to remote data. However, unauthorized modification to data is a denial of backup service issue, and affects the reliability of file storage. Unauthorized release of data is a

security issue, which subverts the file system protection mechanism, and is more important to protect against. (14) The best that can be done is to detect the fact that a modification has occurred. A checksum technique, which stores a checksum in the local catalog, could be used to verify that a retrieved copy is probably unmodified. Unauthorized modification, being a denial of service issue that reduces reliability, might be protected against by some of the reliability enhancements discussed earlier in this chapter (e.g. by making multiple copies of the data), and will not be pursued here.

For a particular application of encryption techniques, there are three measures of merit based on the effects of: 1) how much information is divulged if the correct key is discovered; 2) how much information is made unuseable if a key is lost; and 3) how much work is required to re-encrypt files if the key is changed. In the simple strategy described above, each file is assigned a unique key, which is stored in the catalog. This limits the divulgence or loss of data if the key is compromised or lost, to the contents of the corresponding file. To change a key, only that one file needs to be re-encrypted. Multiple keys, one for each file, are useful in this respect. To compromise the entire set of backup copies requires knowledge of as many keys as there are files. However, if descriptor files are maintained in backup storage also, this strategy may have to be modified.

---

(14) There have been cases of security violations in the implementation of backup systems. For example, access to a backup tape with unencrypted data from the entire file system could allow a user to read the password file. The network facility separates the issues of control over (local) access to backup storage, and protection against (remote) inspection by encrypting the remote data.

5.6 Protection of Remotely Stored Data

Descriptor files contain backup copies of catalog information. If local catalogs contain keys, then descriptor files would also contain these keys, to improve reliability of key storage. By encrypting each descriptor file under a separate key, the advantages of the strategy just discussed, with respect to the three merits, are retained. However, there are two problems. If the set of descriptor files is to be searched as a database, the use of separate keys for each entry in the database would make the search, at best, inefficient. More importantly, however, the ability to retrieve catalog information would be eliminated if damage to a local catalog destroyed the key needed to decrypt the backup information! The solution is to use two special keys, as described below, and illustrated in figure 5.2.

In order to use descriptor files as a database, and to be able to retrieve catalog information even if local catalogs are destroyed, all descriptor files are encrypted by the kernel I/O facility using a single descriptor master key (DMK). Database access to all descriptor files can then be done uniformly under the same key, and maintenance of this key is independent of the local file system. The DMK needs to be specially protected, however, since its loss would prevent decryption of all descriptor files. (15)

---

(15) Loss of the DMK would not compromise protection of remote data files, nor prevent operation of the backup system. However, since backup information from catalogs would no longer be available, an alternate form of catalog reliability enhancement would have to be relied upon to make sure that retrievals could be accomplished after a failure. The extra work required in

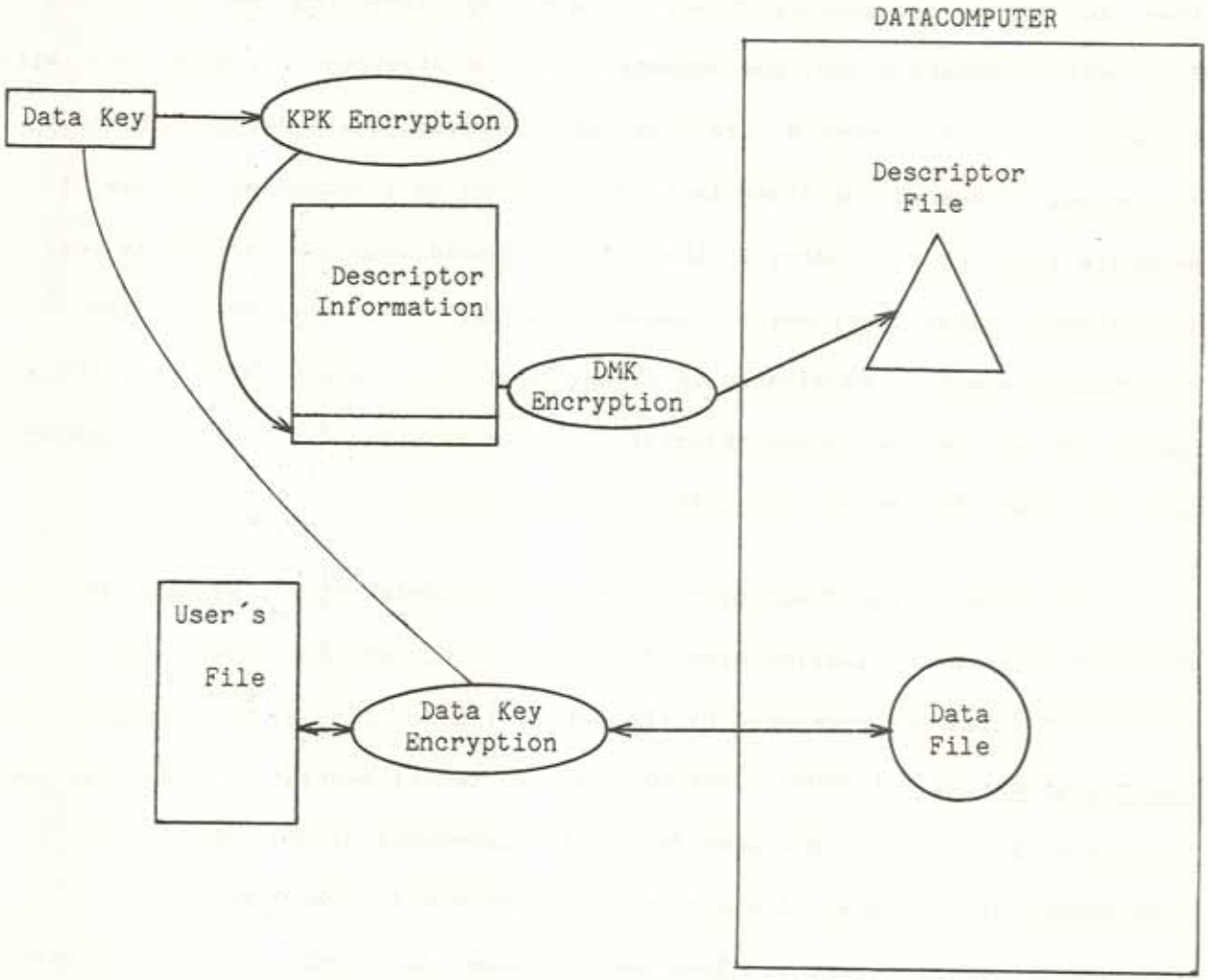

Figure 5.2 Double-Key Encryption for Descriptor Files

5.6 Protection of Remotely Stored Data

If it should be desired to change the DMK, then all descriptor files would have to be re-encrypted under the new key. Since descriptor files do not represent a majority of backup storage contents, this might be feasible, but would not be practical to do too often. Frequency of DMK change depends on the work factor associated with the encryption strategy. A goal should be to make the work factor high enough so that the DMK does not need to be changed often, especially if there are many descriptor files.

The effects of a compromised DMK exhibit the most serious flaw with this mechanism. By compromising a single key, an intruder could decrypt all keys to data files, and subsequently decrypt any data file. The degree of protection of the entire set of backup data files rests with the protection afforded the DMK. Although the DMK is maintained locally, even a small possibility of guessing or otherwise compromising this key might exist. To prevent compromise of the DMK (e.g by remote intruders) from compromising the protection of all data files, the following additional mechanism is employed.

The problem is that knowledge of the DMK exposes all data file keys. To prevent this, the system will also maintain a key protection key (KPK), which will be used to encrypt each data file key before the descriptor file is encrypted. Thus, data file keys are encrypted twice, first under the KPK, and

---

this case would justify special precautions being taken to guard against loss of the DMK.

then under the DMK. Compromise of the DMK will only release status information about the data file, and will not allow the contents of the data file to be decrypted. Both the DMK and the KPK are required to decrypt the data file. This technique effectively protects data file keys, and hence the contents of files, with a higher security key than is used to protect status information. The higher security key is represented by the composition of encryption/decryption operations using the two different keys. (16)

The two key data protection strategy fares well with the three merit criteria mentioned above. Key compromise has been discussed already. However, both keys are managed by the same (local) authority, and this authority must be trustworthy. Otherwise the system breaks down. Since the key management authority is local, it can be audited to make sure it is implemented in a trustworthy and reliable manner.

Loss of the DMK prevents catalog information from being retrieved. Loss of only the KPK still permits catalog information to be retrieved, but will prevent data files from being decrypted. Loss of either key must be prevented. A scheme for guaranteeing the protection and the reliability of the two keys is essential. Such a local scheme can take special measures to

---

(16) Determination of the DMK is "easier" than determination of the KPK, in the sense that one has more information available (all the descriptor files) for using cryptanalytic techniques. To determine the KPK, first the DMK would have to be known, and then only the set of data file keys would be available for cryptanalysis. Protection is "guaranteed" by an encryption technique requiring a sufficient work factor, coupled with periodic key changes.

realize this guarantee (but should not use the network), since it pertains to the very special objects, the DMK and the KPK.

Changing the KPK will affect the storage of the encrypted data file keys in the descriptor files. Since the KPK is less easily compromised, and the amount of work required when it is changed is small, it can be changed whenever the DMK is changed.

The two key idea protects status information (descriptors) at one level, and places a stronger protection on the keys to access the data files. The strongest level of protection is in the protection of the DMK and the KPK, but since these are always stored and accessed locally, a reasonable confidence can be encouraged that remote copies of local cache files will not be exposed to anyone outside the context of the local system, either in remote storage, or in the communication network.

The use of a network connection to communicate with a remote storage facility can be viewed as a thin pipeline through which information flows. At times, the demands for information transfer may be greater than available capabilities. Estimates of network bandwidths and performance of request servicing are made in the next section.

5.7 Performance Issues

Traditional tape backup systems have a high bandwidth I/O connection between the local file system and the backup storage medium. This is not the case with a network implementation. The demand for backup service will frequently exceed the capacity for providing that service. A general queuing strategy can be used to buffer requests when the demand rate exceeds the service rate. This section will investigate the performance issues and problems that arise in using a limited bandwidth network connection for backup. A simple queuing model is presented with assumptions about request rates and network bandwidth, to illustrate typical performance of the facility.

Requests for backup service are made as a function of the update rate and the particular policy implementation in use for processing notifications. In the queueing model chosen here, we will assume that requests are made in a Poisson manner (i.e. the distribution of inter-arrival times is Poisson). Since requests originate from modifications to immune files, and these files are referenced in a manner which depends on the number of user processes running in the system, the reference patterns used by these processes, the size of the primary memory facility, virtual memory management strategies,

etc., each notification can be assumed to be an independent event. This assumption is reasonable because most notifications arise from independent updates from independent processes. On the scale of the whole system, the independence assumption holds for the set of susceptible files in all address spaces. Based on network bandwidth, request processing overhead, and the processing strategy, we will derive a limit on the useful request rate that can be handled.

The queuing model used will be an M/G/1 system with a FIFO request processing discipline. Requests are made in a Poisson manner with average rate r (to be determined in the analysis). The average request processing rate depends on processing overhead, distribution of workload for incoming requests (i.e. service distribution times), and useful network bandwidth. Processing overhead is assumed to be constant for all requests, and is essentially time allocated to bookkeeping functions, sending Datalanguage, establishing auxiliary network connections, etc. Typical network bandwidth based on the ARPAnet is on the order of 40 thousand bits per second for a single connection. Future networks can conceivably support transfer rates an order of magnitude larger. Other techniques, such as multiplexing multiple connections in parallel to achieve higher effective transfer rates will also be considered later. The major workload is in transfering data over the data connection, either in performing copy or retrieval operations. First, we will consider copy operations.

5.7 Performance Issues

It is known from elementary queuing theory that the average waiting time in a FIFO M/G/1 queuing system depends on the request arrival rate, the mean service time, and the variance in the service time. The request processing overhead is assumed to be constant, so adds only to the mean service time, and not the variance. To determine the mean service time, consider the distribution of requests for copy operations for different length files. Based on observations on the Multics file system, most files are small. Ranging between zero and 255K words, they average about 4K words in size. Furthermore, the distribution is exponentially decreasing, with a rapid decrease in the number of files with larger sizes. Approximately 90% of the files are within 5K words of the mean file size. This sharp distribution has a relatively small variance. Given that the average file is 4K words in length, the average service time can be calculated from the network bandwidth. For Multics, the word size is 36 bits, and 1K words can be transferred over a 40Kb connection in about one second. The average copy operation will therefore take 4 seconds. For this particular case the variance was determined, by measuring an actual distribution, to have a value of approximately 2. Any per request overhead is added to the mean, but does not affect the variance.

The M/G/1 system exhibits an unstable, fluctuating behavior when operated near saturation, as determined by the utilization factor. The utilization factor is the ratio of the average arrival rate to the average service rate, and must be less than one for a steady state solution for this

type of system. Given an average service rate s, plus the fixed overhead, we can calculate the maximum serviceable request rate for a given utilization factor. Assume that the fixed overhead can be accomplished in parallel with other copy operations, and is thus zero in the steady state, since it does not add to any particular service time. The average service time can therefore be taken to be 4 in the example. Also assume a utilization factor of 75%, to keep the queues from becoming saturated (this essentially means that the server is busy 75% of the time). Then the maximum request rate is r such that $r*s < .75$, or $r < .19$ (approximately). This is about one request every 5 seconds. Some requests will take longer than 4 seconds to process, and new requests will back up in the queue. Others will take less time, and the queue will empty out. About 25% of the time, the queue will be empty and the server will not be busy. The average waiting time in the queue will be 6.8 seconds, and the average number of requests in the queue will be 1.3 in the steady state solution. This means that if the request rate is limited to one every 5 seconds, then each request can be serviced within an average of 10.8 seconds. This is desirable, since one is attempting to complete the copy operation as soon as possible after the actual request is made. If requests are made more frequently, the system approaches saturation, and there is a long delay between the time a request is placed on the queue, and the time it is serviced.

5.7 Performance Issues

The request rate calculated above can be interpreted on a per user basis as follows. The mean time between requests is 5 seconds. The average number of requests in 5 minutes is 60. If the system supports 60 simultaneous interactive users, each could make one backup request every 5 minutes on the average, and have a backup copy operation completed within 10.8 seconds. The submission of one backup request each 5 minutes for each user is reasonable, considering the way modifications to files usually occur. (17)

First, it may not be desirable to backup files generated by programs (e.g. output from compilers, text formatting programs, etc.), since they can be regenerated if needed. Second, temporary files used in generating output should not be backed up, since it will usually be cheaper (and more feasible) to restart the operation from the beginning, than to produce and recover backup copies of intermediate results. Also, modifications to most files that will be backed up tend to be bursty and infrequent. For example, a compiler may generate output into a temporary file and copy the final result into a permanent file when it is finished, or an editor may buffer changes in a temporary file and copy them into the permanent file when requested by the user to do so.

---

(17) These assumptions are corroborated by performance measurements made on the Multics system. The assumptions above allow for 720 requests per hour, independent of the number of users (these requests are somehow distributed among the users). On Multics, the request rate observed during mid-afternoon peak usage periods is under 350 requests per hour (about three quarters of which apply to user's files, and the rest to input/output queues and other system files), based on statistics from incremental dumps using magnetic tapes. These measurements imply that the network facility can handle peak request loads quite well, if the individual request rates are limited, as described and justified above.

The result is that the permanent file, which is to be backed up, is modified within a small time interval, and then remains unmodified for awhile. The small time window during which modifications occur, the larger time frame during which copy operations occur, and the small average time needed to perform a copy operation tend to decrease the likelihood of making inconsistent copies by copying a file while it is being modified. The assumption is that a particular file being modified this way will remain unmodified for about 5 minutes, or at least notifications will be filtered by the policy implementation so that copy requests for these files will be made once in 5 minutes. (18) This mode of operation means that a user can lose only 5 minutes worth of work on a file, but it will take about 11 seconds to make a backup copy of the bursty modifications. (19) Contrasted with a large

---

(18) For example, if a request is made to make a backup copy of a file before a previous backup request for the same file has been serviced, the new request need not be placed on the queue. Although requests may arrive more often, only new requests arrive each 5 seconds. This is reasonable, since the size of the set of modified susceptible files for which requests are made will be smaller than the number of requests generated. Several requests may arrive for a given file before it is backed up, but the assumption is that a file can receive backup service once every 5 minutes for each user.

(19) These observations are based on the performance assumptions made earlier in the text. Perhaps the average service time is 10 seconds instead of 5, but perhaps only 30 simultaneous users are modifying files. It is difficult to specify precisely the parameters of the model. The example is intended to illustrate a reasonable scenario, and demonstrate the feasibility of the implementation. The reader is free to imagine alternate scenarios and draw conclusions about the performance of the implementation in that environment, with a specific policy implementation, etc. The flexibility of the policy implementation, however, should permit the ideas presented here to be made useful in a wide variety of performance situations, and this fact, not specific performance assumptions made here, is the key to the utility of the mechanisms presented.

incremental dump tape system in which a file accumulates perhaps an hour's worth of modifications before it is backed up, although the copy operation may complete within far less than 11 seconds, the user stands to lose up to one hour's work in a failure. Thus, more frequent copying, but slower copy operations, will minimize unrecoverable loss of data, and dynamic recovery minimizes unavailability. The result is an improvement in reliability of file storage. Dynamic recovery performance is discussed next.

Higher bandwidth connections reduce the average service time, and reduce the variance as the square of the ratio of the new to the old average service time. Higher bandwidth permits higher request rates while still maintaining a given utilization ratio, average delay, and average queue length. For retrievals, however, the rate of requests for backup operations has a time-dependent average; at first there are many requests, and once a file is retrieved, it no longer has any retrieval requests associated with it. The retrieval rate tapers off as more files are recovered in the cache storage. The problem can be viewed in terms of how long it takes a request to be processed, and how long it takes for the retrieval operations, working continuously, to complete. Again assuming an average file size of about 4K words, we need to make some additional assumptions about the retrieval scenario to estimate typical performance.

5.7 Performance Issues

After a system failure, all catalogs and the contents of the system library are restored, and users can log in. References to user files will usually find the file to be already in the cache. However, for several users, the files will be missing, since the damage tends to be local, and only affects a few users. These early file references typically attempt to access such information as the user's mailbox, stored programs, or data files. Only a few files are actually required at a given time, for example to edit the source of a program, some text, etc. Several files may be needed if a program references several subprograms, but they need not be present until actual reference time in a system that supports dynamic linking, such as the Multics system. The delay experienced in retrieving a file that is referenced but missing is important to the user.

The analysis for copy service is applicable to the retrieval service problem. The basic assumption is that the system can service retrievals adequately with a delay time of 10.8 seconds if a request arrives approximately every 5 seconds. This is a steady state solution, which applies if the system operates in this mode for awhile. In fact, the longer the retrieval operations are in progress, the better the situation becomes, since the request rate tapers off. Depending on the particular user, the inter-reference time to different files may vary from a short time for programs that make a rapid sequence of references to subprograms, to a long time for editing a file before needing to reference another one. A reasonable average might be several minutes between references to different files. For

example, a user who references 20 files during an hour long session could probably accomplish some reasonable work, and would average one reference each three minutes. A reference rate of one per five seconds would support 36 users at the 75% utilization factor level. If the system could support 100 simultaneous users, this means that 36% of all system users could be affected by a system failure, losing all their files, but could still access their missing files with an average delay of about 10 seconds. This argument is not intended to specify a level of operation for which this approach to backup is feasible, but is intended to show that even after a fairly widespread failure, any user can profitably use the system for performing some useful operations. This does not imply that all users can function at full capacity, but only that most users can, while at first those most severely affected by the failure must "ride out" the delays encountered in the recovery transients, which should be minimal and as short lived as possible.

We close this section with a look at the extent of effect that extreme changes in performance an order of magnitude in size might have on the operation of the network backup facility. At times, the network or the communicating system might become so heavily loaded that for awhile, it can only provide service at a level an order of magnitude reduced from normal. Alternatively, with the advent of cheaper microprocessors and higher bandwidth communication media, it might be possible in the near future to dedicate processor facilities to achieve an effective communication bandwidth and request processing delay an order of magnitude improved over the level

described previously. First, we consider the degradation caused by reduced service levels.

If requests can only be serviced at a rate of one per minute, then the request rate must be limited to one every 80 seconds in order to maintain the same utilization factor of 75%, and results in an average queue waiting time of just over 100 seconds. This slow service is inadequate for processing requests for copy operations that result from notifications for the default level of backup. An alternative high-bandwidth backup facility should be used to empty out the request queues once they become jammed with requests. However, it might be appropriate to continue performing user-requested backup, which could proceed at the slower rate if the average user-request rate is much smaller than the system default rate. This is really a form of graceful failure of the network facility, which still allows a reduced level of operation for some users, but should be handled by an alternate backup strategy suggested earlier, until conditions improve. The future holds promise for tremendous improvements in performance, however. Some calculations are presented below.

Assuming a network bandwidth of one million bits per second, which is not unreasonable in some current network designs [FaL 72, Far 73, MB 76, Wo 75], the request processing time average drops to .2 seconds. The request rate that can be supported at the 75% utilization factor level is then almost four per second, and the average queue waiting time is about a third of a

second. Possible trends that could offset the improvements of higher bandwidth communications might be toward larger files and systems that support a larger number of simultaneous, interactive users. However, these trends certainly seem to be much slower than the opposing trends toward higher bandwidth communications. The net effect should be to improve the level of service that can be provided for backup of user files, reducing the cost of the service, and improving the reliability of file storage for computer systems in a network.

5.8 Charging for Backup Services

Although the issue of charging for computer services is largely an administrative one, there are several considerations to keep in mind for a network backup system. Charges for local services need to be distinguished from charges for remote services, since the two types of services are administered independently. This section will briefly consider the arguments for a charging scheme for backup in a shared system using the network.

It has already been mentioned that a default level of backup service should be provided for users who are unconcerned with particular reliability requirements. The operation of the backup facility and associated costs and charges should be invisible. The system provides a standard file storage facility, and users pay a fee based on the amount of file storage used, and

not on the amount of backup service provided for their files. Perhaps several classes of storage for several reliability requirement levels can be provided, at several different pricing levels, but the internal accounting and mechanisms of use remain invisible to the user.

User-defined backup is another story. In this case, special services are being performed at the user's direction. Four costs are involved: the cost of interfacing the notification and request submission strategy to the user, the cost of processing requests, the cost of transmitting data over the network, and the cost of storing data at the remote backup facility (e.g. the Datacomputer). The costs for invoking the user interface to backup will be reflected in the charge made for the user's process resource. The cost of processing a request submitted from a user policy will be small compared to the costs of information transfers and data storage. Only one copy of a file is maintained in the backup file system, so this charge is based on file size. The major cost is in communication, and depends on how often backup operations are requested, and how much information is sent for each request. To properly account for frequency and volume of user backup service, charges can be made in terms of the total number of bits transferred for the user, and the quantity of backup storage required. It might also be possible to give "credit" for user backup in the amount of the standard default backup that the user would normally receive. Only the added costs incurred for using more than the average share of resources would be charged.

5.8 Charging for Backup Services

It is mentioned in closing that use of the backup service resembles the use of other classes of service commonly provided in shared computer systems. Most such services provide a queuing mechanism for submitting requests, and an accounting strategy for charging for the amount of resources consumed in providing the service. For example, the line printer queue service receives user requests and charges for service based on how much paper is consumed, how many characters were printed, etc. Since it is not usually of concern to the user that the precise state of request processing be known at all times, this service can be accomplished in parallel with other user operations. It can be called an asynchronous service. Typically, users of asynchronous services must be aware of consistency problems that might arise when user accessible files are modified during the time interval between the submission of a service request, and the time the request is serviced. For line printer services, this problem is usually ignored. For backup service, consistency is a real issue. The facility described in this thesis provides the user with the flexibility to implement a special backup policy, but just as importantly, provides for control over the delivery of backup service, so that the state of asynchronous service operations can be determined, and more precise information is available about the state of backup files. This is a hard service to provide in a tape oriented system. The use of a network allows the random access nature of a large storage facility to be made available so that better service can be provided for the user in improving the reliability of file storage.

5.9 Summary

This chapter has considered several issues relevant to a network implementation of a backup facility. The generation and use of names for uniquely identifying network connections, hosts, and backup copies of files were described. A solution to the problem of enforcing user defined consistency constraints on backup copies of files was presented. In this solution, backup copies are made from temporary shadow copies of user files, which remain consistent before and during backup operations. The method of updating shadow copies is analogous to the method of updating immune files in a well defined manner.

Since the primary network backup facility may not always be available, a strategy was described by which the I/O control component of the backup facility may use several different network hosts for storing backup copies. This strategy results in distributed copies of files, which may make locating and accessing a correct copy difficult. While the retrieval process is severely hampered by the unavailability of the backup storage facility, it will always be possible to continue making backup copies, which is the primary operation of the backup system. To facilitate access to backup copies, temporarily distributed copies are consolidated at the primary backup storage facility when it again becomes available.

5.9 Summary

The ability to control access to data is lost once the data is stored on a remote system which may be accessed by other users. A data encryption scheme for protecting backup copies of files was described. This scheme provides confidence that the use of a remote data storage facility will not circumvent local protection goals.

A queuing model analysis was presented to demonstrate that reasonable performance can be expected from the backup facility using contemporary network technologies and for typical user demands. Finally, charging for backup services was described. The charge for a service is proportional to the cost of the resources used to provide the service.

Chapter Six

Conclusion

6.1 Summary of Results

It can be a frustrating and disappointing experience to find that a system is unavailable when it is needed. The situation is improving today with the ability to construct more reliable hardware and software. There is a parallel psychological trend for users to become less and less tolerant when a system is unavailable, as the system is generally accepted as more reliable. Whereas users might have tolerated delays of hours in the past, the reliance on computer utilities to provide almost continuous service now makes delays on the order of minutes unacceptable. This thesis arose from the observation that in a network, which provides an environment where redundant instances of resources can be widely shared, the unavailability of one specific instance of a resource does not necessarily preempt the availability of the service that it provides.

In particular, one of the most important services provided by a computer system is that of file storage. Recognizing that a file storage facility is subject to a wide variety of failures which can result in

unavailability of the system and an inability to accomplish useful work, one can take several measures to cope with this situation. The early content of this thesis has discussed a number of approaches to improving reliability of file storage in an unreliable environment. Each particular technique is more or less appropriate to a particular application, depending on the goals and purposes of the system. In this thesis, we consider the use of a general purpose computer utility which provides file storage for users, and which might be used for a wide variety of applications. The underlying assumption is that the system should be reliable in the view of the users, and this is evidenced by high availability and low mean-time-to-recover. By using redundant storage of information in a high availability network environment, these goals can be reached.

The network is useful for providing backup storage of files not only because of the high availability of storage facilities, but just as importantly, because of the economic feasibility of using a large remote random access service. Random access capabilities significantly reduce the backup copy management task that is a major burden in tape backup systems. By maintaining only a single copy in backup storage, it becomes much easier to determine the state of backup copies relative to local copies, and recovery is much more rapid because all files are consolidated at a single storage facility, and can be accessed for retrieval dynamically while the system itself is available to the general user community. There is no need to search a tape library, mount several reels, and restore all files before the system can be made available to users.

6.1 Summary of Results

Dynamic recovery while the system is available to users improves the perceived reliability of file storage, but only if backup and recovery work together in a way that reduces the amount of unrecoverable loss if information. A dynamic backup notification mechanism was described that closely tracks changes to local file system information so that these changes can quickly be reflected to backup copies. A useful way to view these operations is to think of the local file system implementation as managing a cache for efficient access to information in the logical permanent file system, which is stored remotely, and accessed in a carefully controlled manner to protect it from unintentional mishaps.

The improvements in capabilities for accessing and managing backup storage afforded by a random access network facility permit greater control by users over the state of backup storage. Database consistency can be guaranteed, and the existence of only one copy of a file in backup storage greatly simplifies the task of determining what state newly retrieved files are in after completing recovery from a failure.

The use of the network for remote storage of backup copies of information presents certain problems not present in a local tape system. The lack of local control over the storage and access of information requires some form of protection mechanism. Encryption was chosen as a solution since it affords a technique to protect remote information to a degree that can be specified locally, in the implementation of the encryption strategy.

6.1 Summary of Results

A basic assumption was made early in the thesis that catastrophic failures are rare, and the solution presented is therefore oriented toward the more frequent but less severe form of failure that accounts for most instances of system unavailability. Some form of high bandwidth mass storage facility, such as a tape system, could be used to recover from a total loss of file storage. Since network bandwidth is limited, it would not be appropriate to use this facility to recover from such a situation. However, a queueing analysis of the proposed mechanisms has shown that even with a network bandwidth that is low in the context of current technology, satisfactory backup service can be provided for both backup and retrieval needs.

The possibility that the primary backup storage facility may not be available leads to a scheme to provide backup for the backup facility. Since the network permits access to many storage facilities, the use of an alternate allows backup operations to continue in the face of a temporary failure of the primary facility. The resulting management of multiple distributed copies can be a difficult problem. However, the special nature of access needed for backup copies eases the burden. By consolidating distributed copies made during a temporary failure, copies can be centralized at the primary storage facility when recovery is completed. This operation is similar to the management of multiple copies on tapes during the maintenance of complete and catchup dumps, but is simpler to accomplish because access to backup storage state information is much easier. The fact that file storage recovery is prevented only if two independent systems fail at about the same time lessens

the chances that a backup copy will be unavailable for retrieving when the local system fails.

The drawbacks and limitations discussed above require that certain tradeoffs be made, for example between frequency of backup and acceptable amount of unrecoverable loss of information. Even so, the mechanisms provide an improvement in reliability enhancement over tape systems from the two standpoints of system availability and degree of recovery possible. In addition, the basic mechanisms used by the backup facility are meant to be simple, secure, and flexible. Several generalizations and extensions of these basic mechanisms for other uses are suggested in the next section.

6.2 Other Applications Using the Backup Mechanisms

A discussion of the operation of the backup mechanism has already been given. Rather than reiterating that discussion here, we will merely suggest some possible extensions for providing other services that might be of interest. An investigation of all the issues that might arise is beyond the scope of this thesis, and as such, is suggested as an area of future research.

The basic facility that the backup storage mechanism provides is a form of file system. Basic assumptions were made about the contents and access needs of this file system for purposes of backup, in order to simplify the

design and illustrate the essential issues. Some of the restrictions can be removed to provide more general file system services for users. One such service has already been mentioned several times: a file archiving facility. By allowing multiple copies of a file to exist, several backup versions could be retained for an indefinite amount of time. A carefully defined interface is required to produce the desired effect in backup storage when a local file is deleted. The maintenance of distributed copies is more complicated, since writing a new version does not make the old one obsolete. A user interface is needed for managing archive storage (e.g. for operations such as recalling and deleting archived files), and an access control mechanism for retrieval of archived copies is required. It might be desirable to archive catalog and access control information with the file, instead of keeping it local. The tradeoffs between access cost and storage cost need to be investigated.

Archiving is one example of what might be called a user service. The organization of computer systems in a distributed computing network environment motivates the development and delivery of services to a wide distribution of users [Cos 75, Cro 75, Far 73, Rob 70, Row 75, Th 73]. The essence of distributed computing is the semi-autonomous tight coupling between a set of local resources dedicated to providing an efficient, cost effective set of services, and an additional need and ability to share other services, on an occasional basis, using a network communication facility. New services can be provided using extensions of the mechanisms presented in the thesis. It will also be of interest to investigate the nature of requirements for backup service in a distributed computing environment.

6.2 Other Applications Using the Backup Mechanisms

The design presented has been oriented toward the needs of a large, shared computing utility. The nature of the needs of smaller systems, such as personal computers, intelligent terminals, special purpose dedicated systems, etc., that might also be part of the computer network is an area that will be interesting to investigate. The tradeoffs between recovery time, system availability, and a specific policy implementation will vary with the application and the size of the system. Protection and alternate reliability enhancement schemes may also vary. While the design presented in the thesis is believed to be flexible and general enough to be adaptable to specific needs in all these situations, only one environment was illustrated. Especially with the distribution of many services in the network, the specific nature of backup requirements for each system will be a matter of local concern. Further investigations into requirements of other applications can be done by the reader wishing to implement a backup facility for that application.

6.3 The Distributed File System and Beyond

The backup service provides a very restricted set of file system functions. Data can be stored and later retrieved. Access is restricted to a privileged backup I/O procedure. The basic mechanisms used to store and retrieve files can be useful for implementing a general purpose distributed file system. A short summary of the issues will be given in this section.

6.3 The Distributed File System and Beyond

A file system provides a facility for managing file names for users. Names provide a handle for accessing the file. The backup system provides a primitive naming and access facility, but will not be sufficient for a general purpose distributed file system [Boo 72, FaH 72, PeM 75]. A technique employing uniform use of network-wide unique file names that are more flexible than UID's should help.

The mechanism used for retrieving files that are not in the cache can be adopted to access files in a distributed file system. A distributed file system can be considered to be the union of the individual file systems that comprise its implementation. The cache mechanism provides a technique for efficient access to distributed files. Maintenance of the distributed file system is a cache management problem. However, if distributed files are shared, this problem becomes a very difficult one. Some of the ideas presented are applicable for naming, accessing, and updating distributed files, but are not sufficient for keeping shared files consistent or handling a general file migration strategy. Unavailability of part of the distributed file system introduces problems, but so does a multiple copy scheme that might be used as a solution. Controlling and authorizing access, and improving reliability for such a file system are other problems.

With the advent of larger and cheaper computer networks, there are two tendencies for storing data. One tendency is to centralize data for personal use, so that access and control are efficient and localized. An opposing

tendency is to distribute data for sharing, for reliability, and for economy. If reliability is a concern to the user, it is hoped that the ideas in this thesis will lead to a practical, flexible, and successful backup facility. If other issues are a concern, it is hoped that the ideas in the thesis are a step in the right direction toward solutions to harder problems. Whether the end application is a private, dedicated file system, or a large, shared distributed file system, needs for communication will make networks useful, and needs for reliable services can use networks to enhance reliability of file storage.

BIBLIOGRAPHY

[ARPA 74] Advanced Research Projects Agency, "ARPA Current Network Protocols," (December 1974), NIC Document Number 7104, Network Information Center, Stanford Research Institute, Menlo Park, Ca.

[BCD 72] Bensoussan, A., Clingen, C. T., and Daley, R. C., "The Multics Virtual Memory: Concepts and Design," CACM 15, 5 (May 1972), pp. 308-318.

[Bob 72] Bobrow, D, et. al., "TENEX - A Paged Time Sharing System for the PDP-10," CACM 15, 3 (March 1972), pp. 135-143.

[Boo 72] Booth, G. M., "The Use of Distributed Databases in Information Networks," Proceedings First International Conference on Computer Communication (October 1972), p. 371, Washington, D.C.

[CCA 75] Computer Corporation of America, Datacomputer Version 1 User Manual, Datacomputer Project Working Paper No. 11 (August 1, 1975), Cambridge, Ma.

[CCA 73] Computer Corporation of America, Further Datalanguage Design Concepts, Datacomputer Project Working Paper No. 8 (December 15, 1973), Cambridge, Ma.

[Cor 65] Corbató, F. J., and Vyssotsky, V. A., "Introduction and Overview of the Multics System," AFIPS FJCC Conference Proceedings 27 (1965), pp. 185-196, Spartan Books, Washington, D.C.

[Cos 75] Cosell, B. P., et al., "An Operational System for Computer Resource Sharing," Proceedings Fifth Symposium on Operating Systems Principles (November, 1975), pp. 75-81, Austin, Texas.

[Cro 72] Crocker, S. D., et al., "Function-Oriented Protocols for the ARPA Computer Network," AFIPS SJCC Conference Proceedings 40 (May 1972), pp. 271-279, AFIPS Press, Montvale, N.J.

[Cro 75] Crocker, S. D., "The National Software Works: A New Method for Providing Software Development Tools Using the ARPANET," Proceedings Meeting on 20 Years of Computer Science (July 1975), Pisa, Italy.

[Fab 73] Fabry, R. S., "Dynamic Verification of Operating System Decisions," CACM 16, 11 (November 1973), pp. 659-668.

[FaH 72] Farber, D. J., and Heinrich, "The Structure of a Distributed Computer System: The Distributed File System," Proceedings First International Conference on Computer Communication (October 1972), p. 364, Washington, D.C.

[FaL 72] Farber, D. J., and Larsen, K. C., "The System Architecture of the Distributed Computer System: The Communications System," Proceedings Symposium on Computer Networks (1972), Brooklyn Polytechnic Institute.

[Far 73] Farber, D. J., et al., "The Distributed Computing System," Proceedings of the 7th Annual IEEE Computer Society International Conference (February, 1973), New York, N.Y.

[Fei 73] Feistel, H., "Cryptography and Computer Privacy," Scientific American 228, 5 (May 1973), pp. 15-23.

[Fei 75] Feistel, H., et al., "Some Cryptographic Techniques for Machine-to-Machine Data Communications," Proceedings IEEE 63, 11 (November 1975), pp. 1545-1554.

[Fr 69] Fraser, A. G., "Integrity of a Mass Storage Filing System," The Computer Journal 12, 1 (February 1969), pp. 1-5.

[Ke 76] Kent, S. T., "Encryption-Based Protection Protocols for Interactive User-Computer Communication," S.M. Thesis, M.I.T. Department of Electrical Engineering and Computer Science, published as M.I.T. Laboratory for Computer Science Technical Report No. 162 (May 1976).

[Mar 75] Marill, T. and Stern, D., "The Datacomputer: A Network Data Utility," AFIPS NCC Conference Proceedings 44 (May 1975), p. 389, AFIPS Press, Montvale, N.J.

[MB 76] Metcalfe, R. M., and Boggs, D. R., "Ethernet: Distributed Packet Switching for Local Computer Networks," CACM 19, 7 (July 1976), pp. 395-404.

[MIT 74] M. I. T. Laboratory for Computer Science, "An Introduction to Multics," Technical Report No. 123 (February 1974).

[NBS 75] National Bureau of Standards, "Computer Data Protection," Federal Register 40, 52 (March 1975), pp. 12067-12250.

[Org 72] Organick, E. I., The Multics System: An Examination of Its Structure (1972), M.I.T. Press, Cambridge, Ma.

[Orn 75] Ornstein, S. M., et al., "Pluribus -- A Reliable Multiprocessor," AFIPS NCC Conference Proceedings 44 (May 1975), p. 551, AFIPS Press, Montvale, N.J.

[Par 72] Parnas, D., "On the Criteria to be Used in Decomposing Systems into Modules," CACM 15, 12 (December 1972), pp. 1053-1058.

[Pe 71] Peck, P. L., "Survey of Applicable Safeguards for Insuring the Integrity of Information in the Data Processing Environment," MITRE Corporation Technical Report (Washington Operations), MTP-356 (June 1971).

[PeM 75] Peebles, R., and Manning, E., "A Computer Architecture for Large (Distributed) Databases," Proceedings International Conference on Very Large Databases (September 1975), p. 405, New York, N.Y.

[Rob 70] Roberts, L., and Wessler, B., "Computer Network Development to Achieve Resource Sharing," AFIPS SJCC Conference Proceedings 36 (May 1970), pp. 543-549, AFIPS Press, Montvale, N.J.

[Row 75] Rowe, L. A., "The Distributed Computing Operating System," University of California at Irvine, Department of Information and Computer Science, Technical Report No. 66 (June 1975).

[Row 73] Rowe, L. A., Hopwood, M. D., and Farber, D. J., "Software Methods for Achieving Fail-Safe Behavior in the Distributed Computing System," Proceedings IEEE Symposium on Computer Software Reliability (May 1973), p. 7, New York, N.Y.

[Sal 75] Saltzer, J. H., and Schroeder, M. D., "The Protection of Information in Computer Systems," Proceedings IEEE 63, 9 (September 1975), pp. 1278-1308.

[Sh 71] Schell, R., "Dynamic Reconfiguration in a Modular Computer System," Ph.D. Thesis, M.I.T. Department of Electrical Engineering, published as M.I.T. Laboratory for Computer Science Technical Report No. 86 (June 1971).

[Shr 72] Schroeder, M. D., "Cooperation of Mutually Suspicious Subsystems in a Computer Utility," Ph.D. Thesis, M.I.T. Department of Electrical Engineering, published as M.I.T. Laboratory for Computer Science Technical Report No. 104 (September 1972).

[Sm 72] Smith, J. L., Notz, W. A., and Osseck, P. R., "An Experimental Application of Cryptography to a Remotely Accessed Data System," Proceedings ACM 25th National Conference (August 1972), pp. 282-297.

[St 74] Stern, J. A., "Backup and Recovery of On-Line Information in a Computer Utility," S.M. and E.E. Thesis, M.I.T. Department of Electrical Engineering, published as M.I.T. Laboratory for Computer Science Technical Report No. 116 (January 1974).

[Th 73] Thomas, R. H., "A Resource Sharing Executive for the ARPANET," AFIPS NCC Conference Proceedings 42 (June 1973), pp. 155-163, AFIPS Press, Montvale, N.J.

[Tu 73] Turn, R., "Privacy Transformations for Databank Systems," AFIPS NCC Conference Proceedings 42 (June 1973), pp. 589-601, AFIPS Press, Montvale, N.J.

[We 72] Wensley, J. H., "SIFT -- Software Implemented Fault Tolerence," AFIPS FJCC Conference Proceedings 41 Part 1 (December 1972), pp. 243-253, AFIPS Press, Montvale, N.J.

[Wo 75] Wood, D. C., "A Survey of the Capabilities of 8 Packet Switching Networks," Proceedings 1975 Symposium on Computer Networks: Trends and Applications (1975).