UNIVERSITY OF MANITOBA

THE DESIGN AND IMPLEMENTATION OF A STRUCTURED INDEXED FILE SYSTEM

by

Howard John Ferch

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF DOCTOR OF PHILOSOPHY

Department of Computer Science

University of Manitoba

Winnipeg, Manitoba

May, 1978

THE DESIGN AND IMPLEMENTATION OF A STRUCTURED INDEXED FILE SYSTEM

BY

HOWARD JOHN FERCH

A dissertation submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

This thesis discusses a file system specifically intended to manage source data files. The motivation for this file system was dissatisfaction with the facilities provided by existing file editing systems; the new file system is intended to provide a powerful set of file management facilities to allow for the construction of a more flexible editing system.

The two most important features of this file system are the ability to perform direct updating of data, and the management of hierarchies of files. Direct updating of data eliminates the copying of files required to perform editing functions on sequential files. Three major aspects to the hierarchical support are discussed. The use of a hierarchy allows for a subdivision of data in a way that matches the user's logical subdivision. In addition, the hierarchy is managed in such a way that an entire subsection of the hierarchy may be processed as a unit. The hierarchy provides a logical base for inherited attributes and password protection.

After the limitations of existing editors, and the facilities to be provided, are discussed, an actual implementation of such a file system is presented in three parts. First, the storage structures for providing the direct updating and file hierarchy capabilities are discussed. Secondly, the techniques used for maintaining integrity of the user data are presented. Finally, the performance level attained by such a file system is discussed and shown to be comparible to that of existing editors.

ACKNOWLEDGEMENTS

The author would like to express his gratitude to his advisor, Dr. C.R. Zarnke, for his many comments and suggestions during the production of this thesis and the software product upon which it is founded. Thanks are also due to the other examiners of this thesis, Dr. D.D. Cowan, Dr. R. J. Collens, and Dr. S. A. Bukhari, for the time they spent in reading the thesis, and for their suggestions for improvement.

Special thanks must be given to the many people who used the NAM file system during its development, and whose comments have helped to make it the successful product that it is.

The financial assistance of the National Research Council of Canada, through a postgraduate scholarship, is gratefully acknowledged.

TABLE OF CONTENTS

CHAPTER 1 - INTRODUCTION

With the advent of more powerful and cheaper computer hardware, there has been a correspondingly greater use of general purpose time-sharing systems. Such systems provide each user with a computer terminal from which he can interactively and in real time enter commands and data to the computer system and can interrogate the computer system. It is now common for many manufacturers to supply such a general time-sharing system as part of their major software offerings with a batch stream as an additional more minor component. This is even true for small minicomputer systems. For example, in Digital Equipment Corporation's RSX-11 [DEC 1] series of operating systems for the PDP-11 [DEC 3] computer system, which is a minicomputer system, only the largest version of the system even provides a batch capability at all.

One of the most widely used components of such a time-sharing system is the source editor, which provides support for the entry and editing of source data. The term "source data" is used to mean data in external character form produced directly by human effort, as opposed to

machine generated and manipulated data. Such data usually consists of program text, data for a program, or documentation (this thesis was entered and edited using such an editing system).

This source data is stored in files on secondary storage media such as magnetic disk units. The purpose of the editor is to provide a facility for editing these data files. Appropriate parts of the user's files are brought into the processor's main storage where they may be manipulated; the user employs editing commands to alter his data and the changes he makes are then stored back on the secondary storage media. Some of the best known of these editors include IBM's TSO [IBM 2], Digital Equipment Corporation's TECO [DEC 4], and Stanford's WYLBUR [Faiman 1973]. All general purpose time-sharing systems provide a built-in editor.

Most of these editing systems operate as a standard user program using the operating system facilities and are not actually built-in to the operating system. (i.e. they are treated simply as ordinary application programs). Consequently, they normally allow updating only of the types of files, or of some of the types of files, supported directly

by the operating system. Since the simplest type of file is a sequential series of records, the editor invariably supports the user's data only as a sequential file to which random editing operations cannot be applied directly. Thus, while editing, the editor normally copies the user's file to a specially organized (i.e., non-sequential) work file, edits the copy, and then rewrites the user's file when he is finished. If the computer system fails for some reason (for example, due to a power failure), the user may lose the changes performed during that session.

The editors in use provide a fairly standard set of editing operations. Records may be listed, inserted, deleted, altered, copied, moved and renumbered. Operations on entire files, such as listing, deleting, copying, and moving are also allowed, although these operations are often provided directly by the time-sharing system rather than by the editor itself, since they are of general application, and are not specifically used for editing purposes.

Unfortunately, while a great deal of study and effort has been devoted to operating systems design and implementation, particularly in the areas of process management, scheduling, and storage management, almost no mention is ever even made

of the editing system, although it is that part of the system that is most visible to the user. For example, an examination of the great amount of documentation on the MULTICS system [Organick 1972], which is probably the best known operating system implemented by someone other than a computer manufacturer, yields almost no information about the editing system's capabilities. As another example of the little attention given to editing, Ritchie and Thompson, in their description of the otherwise interesting UNIX system [Ritchie 1974], fail even to mention the editor even though (using their own figures in the same paper) the editor was the most heavily used command and also was one of the largest consumers of system resources.

The goal of this thesis, then, is to study the design, implementation, and performance of a file system providing much more sophisticated file manipulation facilities for the editing system than those currently found in use. The "file system" includes all that support software which provides data management services for the editor, in contradistinction to the editing function itself which includes the editor command language analysis and the actual altering and acquisition of data.

The major thrust of this thesis will be to demonstrate the practicality and cost of a more powerful file system by the construction of an actual file system implementing the new ideas. An actual file system, called the Network Access Method (NAM), has been implemented by the author and is in use providing facilities for the MANTES editing system [Zarnke 1978]. The entire system has not been implemented just to demonstrate the workability of the new ideas but is in production as a major editing system for a current user population of approximately 600 users at the University of Manitoba. The advanced features of the NAM file system have allowed the construction of an editing system possessing many new features over existing editors; the construction of NAM was motivated by the fact that the desired features of the editor could not be implemented using the existing operating system. Note, however, that it is not the intention of this thesis to discuss the actual features or commands supplied by the MANTES editor.

The thesis will begin in chapter 2 by discussing those features supplied for existing editors, and those additional features which are provided by NAM, as well as justification for the new facilities. Chapter 3 will discuss the external interface to NAM: that is, those features supplied to the editor by the file system. This is not to say that NAM

exists solely for use by editors; in fact it should be accessible to programs generally. However, most examples and justifications are based on the existence of an editor as the invoking program. In chapter 4 the storage structures used to implement the features provided will be discussed, while chapter 5 discusses a special aspect of file systems providing a direct updating capability; that of maintaining integrity of the user data across system failures. Chapter 6 discusses the performance and practicality of the NAM file system, using experimentally gathered data. Chapter 7 summarizes the results.

CHAPTER 2 - FILE SYSTEM FUNCTIONS

The first step in the construction of any software system is to acquire a knowledge of what the system is intended to do. This chapter, then, presents a general discussion of the actions of the file system. More specific details will follow in the next chapter but here is presented the background and motivation for the ideas which form its basis. The functions which are discussed here are crucial to the entire file system - the combination of functions discussed here provides the need and justification of the file system since the desired collection of features does not exist elsewhere. The remainder of the thesis will present the difficulties inherent in the implementation of the given functions.

Before starting the discussion proper it is important to distinguish the file system supplying the data storage facilities from the actual editor. The important concerns for this thesis are not the syntax and use of editor commands, but rather the facilities provided by the file system to the editor, used to manage the storage and retrieval of files and records. The function of the editor

is to interact with the user, usually through a series of commands. These commands are translated into requests that the file system perform certain data management functions. The actual editor is important only in that the functions it provides to its users dictate and motivate the functions provided by the file system - it is the desire to provide a sophisticated editor that forces the development of a similarly sophisticated file system to support it. It is not important for these discussions whether the file system is in fact part of, and unique to, the editor, or whether it is a general function supplied by the operating system, although it makes sense to separate it clearly. In general, to make the distinction in function between the editor and file system clear it can be said that the editor accepts commands from its users and performs operations upon sets of records - the actual location of the records on the secondary storage media and the transferring of data to and from the editor's work areas is the responsibility of the file system. The editor should have no knowledge of the techniques used to store the data - it only knows how to invoke the file system to retrieve and insert the particular data items it is concerned with.

2.1 BACKGROUND

A brief history of editing systems is useful to bring together those functions supplied by existing file systems. Source file editing systems have been in regular use since the early 1960's. In collecting a list of features desired in designing a new file system it thus would be expected to find a large variety of ideas and techniques in use by the fairly large number of timesharing systems in use or previously in use. Unfortunately, however, this is not the case. For example, there is a remarkable similarity between the features supported by the Compatible Time-Sharing System [Corbato 1962] in use in the early 1960's and those provided by modern day systems such as IBM's TSO [IBM 1,2], the MULTICS system [Organick 1972], or UNIX [Ritchie 1974]. While a substantial amount of work has been done in other areas of operating systems architecture, almost no original ideas have been presented in the areas of file systems design or source file editing.

Thus it is fairly easy to discern the important features supplied by current editing systems. Since the files operated upon by editors are read by other programs such as compilers, the files used are those generally provided by

the operating system as a whole. In fact, the features provided by editing systems are invariably linked to those of the host operating system; for example, in an operating system not providing sequence-numbered records in files it would be very unlikely to find an editor with sequence numbered records. Thus a discussion of editor features is to a large degree a discussion of the file systems provided by operating systems.

The most common types of files are sequential files: files containing an ordered set of records. Consequently, most editors are designed to operate on such files. The user may select a file and then perform a standard set of operations upon records in the file. These operations include listing, insertion, deletion, altering, moving and copying of records. Sometimes the records of a file are identified by unique sequence numbers; in other editors the records are not explicitly numbered. In either case, the sequential ordering of the records is important. No other structuring of the records is provided; the user must locate a record by its sequence number or by scanning a set of records in order to find one based on content. Only one file may be edited at a time, thus making quite ditficult inter-file operations such as making the same change to several related files.

Since sequential files as used by the editor are not suitable for the addition and deletion of records some effort must be made by the editor and its file system to make changes. Consequently, the file must first be copied to a specially organized internal work file maintained by the editor's file system. This internal work file is organized in such a way that insertions and deletions can be performed upon it. The editing changes are thus made against the work file; when all desired changes have been made the work file is copied back to the permanent copy kept in sequential order that can be accessed by all the other programs in the system. This copying and saving back of the file is slow and costly in machine resources and is especially annoying if the user wishes to make only a few changes (or even just to list parts of the file). This copying to a work file is also the reason that the user may edit only one file at a time - all files to be edited would have to be copied to internal work files first. Thus the user is discouraged (by the time taken) from switching from editing one file to editing another. If, for example, while editing one file he notices he needs to make some minor change to another file he cannot easily do this. In addition, if some problem such as a system failure or power failure occurs he may lose some or all of his changes since

the editor work file may be lost or may be difficult to retrieve.

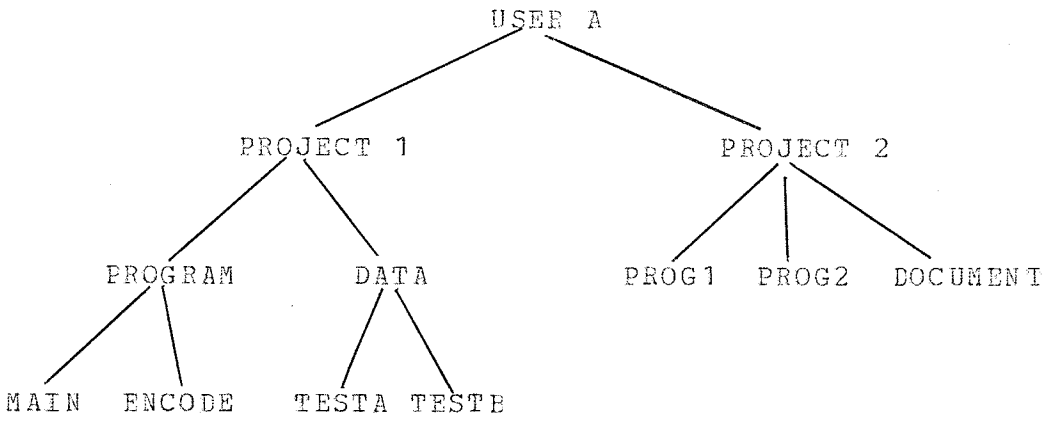
One feature very useful for editing systems is the ability to associate attributes with files. A system which supports, potentially, a diversified set of files, needs to allow files to differ in some respects; that is, the files are not necessarily completely uniform. For example, text files would contain both upper and lower case text while program source files might normally contain text in only one case. Some editors have the ability to associate such attributes with individual files in order that the attributes be applied automatically without extra user effort. In this way, the editor can make any adjustments to its operation (such as converting the user's input to a single case) automatically. These attributes have no intrinsic meaning for the file system, but since the attributes must be permanently associated with the file, the file system must be able to accomodate their addition and deletion.

Another facility that is sometimes offered is that to create several versions of a file. Quite often it is important to restore a file to its state at some earlier time. For example, a user may try some changes to an

algorithm, only to discover that the old one was better, and then he may wish to return to the old form. Having several versions of a file also allows the user to keep a production working version of a program and a current version being altered. This facility can easily (but not efficiently) be obtained by making a copy of a file before altering it. Editing systems which support the facility rely upon operating system facilities for keeping multiple copies of a file. For example, a user of IBM's OS/VS series of operating systems can use generation data groups [IBM 3] to keep track of versions of a file, while users of DEC operating systems such as RSX-11 [DEC 1] are given a direct version facility for all of their files. A more efficient technique for implementing versions will be described later in this thesis.

One of the most important concepts in file management systems, although one which is usually external to editing systems, is the ability to create hierarchical structures to relate files. A user of a computer system normally has sets of files which are related to each other; for example, a program source file, an input data file, and a documentation file, all pertaining to one single project. Virtually all operating systems allow the construction of a file hierarchy to express the natural relations between files. For

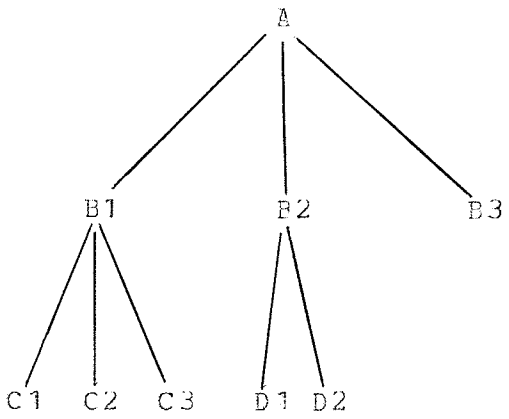
example, a user may create a hierarchical structure such as;



which defines files related to two projects. A hierarchical structure such as this has many advantages; it provides a logical structuring of data; it provides a natural mechanism for implementing naming conventions and protection rights; and it provides a means for referring to a related group of files as a unit. Unfortunately, existing editing systems are not capable of dealing with such a hierarchy and consequently permit the editing of only the bottom files. If a user wishes to alter all occurrences of a text string in several files, he must edit each individual file separately.

Before continuing on with the limitations of existing editors, however, a few definitions and descriptions of hierarchies in general are in order. Given a hierarchical

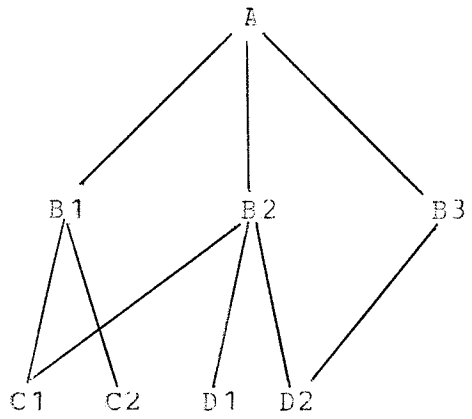
set of files, such as that depicted here, each participant in the tree structure is called a node.



Those nodes appearing at the bottom level and possessing no subdivisions are called leaves, (C1, C2, C3, D1, D2, B3) while the very top node is called the root (A). Starting with any node in the hierarchy and travelling upwards, all nodes passed on the way are called the ancestors of the given node. (A and B1 are the ancestors of C1), while the immediate ancestor of a node is called its father. (B1 is the father of C1). In a similar fashion, the nodes immediately under a given node are called its sons. (C1 is a son of B1). Different sons of the same father will be called brothers. (C1 and C3 are the brothers of C2).

In addition, hierarchies in which any node (except the

root node) may have several fathers will be discussed, as in;



Such structures will be referred to as networks; the nodes C1 and D2 in this example will be referred to as network nodes and they possess multiple fathers.

While the user is generally given the ability to create such hierarchies (although some systems do not allow the creation of networks), only the leaves themselves contain any data. The ancestors of the leaves exist only for the purpose of relating the leaves and assist in naming them. Each node in the hierarchy is named (by the user); the name by which a node may be referred to is that generated by concatenating the names of all its ancestors, beginning with the root node, down to the given node. Some delimiter, such