Datafair 69
The British Computer Society
Mancester, England
August 1969

ANALYSIS OF MODERN FILE SYSTEMS

by

Stuart E. Madnick
Massachusetts Institute of Technology
Project MAC
Cambridge, Massachusetts 02139
U.S.A.

Classification:

IV. Advances in programming, and
V. Data bases.

Abstract:

This paper presents a flexible strategy for the design of modern general purpose file systems. The model system developed is used as a basis for comparison among several of the most advanced file systems currently in use.

Analysis of Modern File Systems

by Stuart Madnick

The fields of computer applications and computer programming have grown at an impressive rate. Computer systems programming has probably outpaced most of the other areas of computer software technology. For many complex reasons, the computer software technology used in the development of modern operating systems remains more of an art than a science. It is difficult to build on the work of others and, as a result, many software projects have failed.

A typical physical computer system has a varied assortment of secondary storage devices (e.g. disks, drums, data cells, etc.) in addition to the primary storage (i.e. core memory). It is generally true that if primary memory size was limitless and very inexpensive, there would be no need for secondary storage (possible exceptions may be for backup requirements and transfer of data). In the framework of this paper, a file system will be defined as the software mechanism that extends the capacity of primary storage by handling and coordinating the transfer of information to and from the secondary storage devices (i.e. produces an "augmented computer" which is more flexible than the physical computer). This definition is somewhat more restrictive than other common interpretations which include as part of the file system definition the physical devices or the programs that operate upon the data (i.e. application programs or "general purpose data management packages"). In this interpretation the file system merely stores and transfers information but does not operate upon it.

This paper considers the role of sophisticated file systems for general purpose operating systems and presents a generalized model for these systems. The model is intended to serve two functions: (1) provide a basis for analysis and comparison of modern file systems, and (2) provide the foundations for the synthesis of new file systems.

The model developed is primarily based upon two principles: (1) hierarchical modularity, relating strongly to the work of Dijkstra, and (2) virtual memory, similar in concept to the endeavors of TSS/360 and Multics. The operation of a file system is described by a strict hierarchy of seven levels:

1. Access Methods and User Interface
2. Logical File System
3. Basic File System
4. File Organization Strategy Modules
5. Allocation Strategy Modules
6. Device Strategy Modules
7. Input/Output Control System

It is a basic premise of this paper that all modern file systems perform the functions embodied in the above seven levels. By carefully organizing the interdependencies of the levels to "build" on each other, it is claimed that powerful file systems can be effected easily and conveniently.

Although a precise description of a file system cannot be presented briefly, there are several general characteristics common to most file systems. In particular, a user specifies his request, such as read or write, by designating a file and an element within the file. Most advanced file systems allow considerable flexibility in the mechanism used to specify a file, it is typically described by means of a symbolic file name. Furthermore, the element within the file is specified in terms of the uniform logical representation of elements (i.e. device independent) in the particular file system which may, but usually does not, correspond to a precise physical specification of how and where the element is stored. For example, a typical request might be of the form:

"Read item 23 from file ALPHA into location 1564."

Realizing that information must usually be stored on devices in somewhat obscure ways, there must be some sequence of transformations required to convert the user's request into its final form that physically operates of the secondary storage device. Quite often the transformation is viewed as a single step, but that is a gross oversimplification that hides the fundamental mechanisms in use. In the diagram below the conversion process is illustrated in terms of a discrete sequence of logical transformations.

A simple analogy is presented in Figure 1 that loosely parallels the file system transformations. The analogy is only intended to provide some insight into the rationale behind each stage of the transformation. The transformation stages are briefly described below:

Logical File System (LFS):

The process starts from the user's request to "read item 23 from the file ALPHA into location 1564". The first step is to convert the symbolic file name into a unique numeric file identifier using a data base called the File Name Directory. In the analogy, this corresponds to looking up John Doe's identifier which is a Social Security Number in this illustration. The purpose for using an identifier is basically the same in both cases. It is usually more convenient to store information, manually or automatically, by means of a unique numeric "key" rather than a symbolic name which may, under certain circumstances, not even be

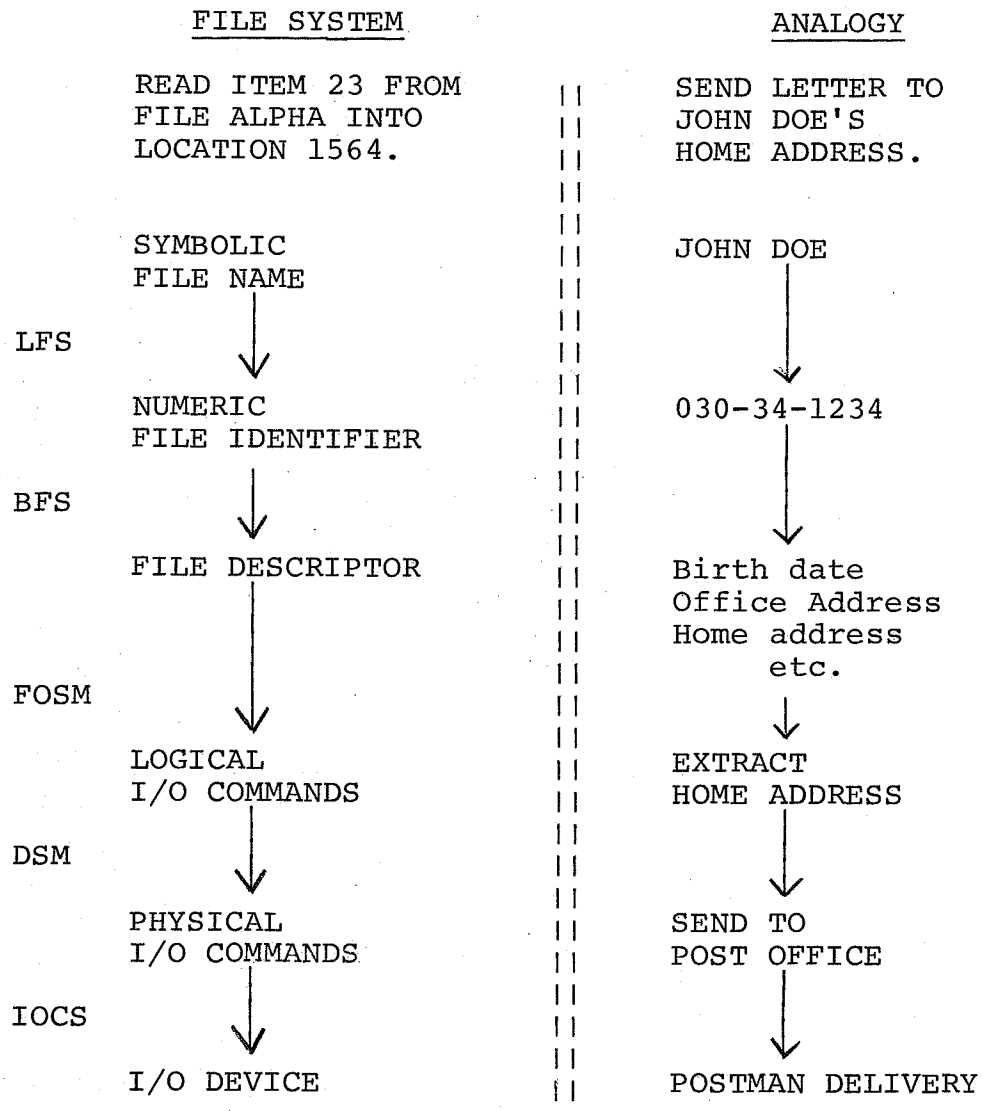

Figure 1.

Logical Transformations in File System

unique (i.e. there may be more than one John Doe in which case other factors must be considered in order to uniquely identify the person under consideration).

Basic File System (BFS):

The file identifier can then be used in conjunction with a data base called the File Descriptor Directory to conveniently access all the information known about a file, this information collectively is known as the file's descriptor. In the analogy, this would correspond to requesting all information in the social security records of 030-34-1234.

File Organization Strategy Modules (FOSM):

Now that everything is known about the file, it is necessary to consider the specific operation to be performed. Using the file descriptor and a data base called the File Map, a sequence of logical 1/0 commands can be produced. These are called logical 1/0 commands because they do not consider the specific physical characteristics of the secondary storage device to be used. This is analogous to putting an address on a letter which can be done without considering the physical destination nor the route to be taken.

Device Strategy Modules (DSM):

In order to complete the transformation, the logical 1/0 commands must be converted into the appropriate sequence of physical 1/0 commands using a data base called the Device Map. This conversion may be trivial or complex depending upon the peculiarities of the device and 1/0 interfaces to the devices. In the analogy this process is performed at the Post Office where the address is used to determine the physical routing needed to get the letter to its destination.

Input/Output Control System (IOCS):

The final step in the process is the physical transfer of information. This is usually performed by means of software/hardware interactions to activate the appropriate device and confirm the successful completion of the request. Of course, in the analogy this transfer is accomplished by the postman assisted by trucks, planes, trains, and other automation.

In this paper, the manner in which several diverse modern file system perform these logical transformations is discussed. The IBM Operating System/360 Data Management facility is probably one of the most comprehensive file systems incorporated in a batch-oriented operating system.

On the other hand, the demands of general-purpose time-sharing forced the MIT CTSS (Compatible Time Sharing System) and SDS 940 systems to adopt a more flexible file system structure. In general, none of these systems have taken full advantage of the inherent modularity of the transformation levels.

Recently, the modular design has been used to develop two new file systems for very dissimilar computer system, a small IBM 1130 with 8,000 16-bit words of memory and a large IBM 360/67 with 512,000 8-bit bytes of memory.

Finally, MIT's Multics (Multiplexed Information and Computing Service) illustrates a design that exploits the effectiveness of utilizing the logical hierarchy, augmented by elaborate hardware features, to build an extremely powerful and flexible file system capability.

BIBLIOGRAPHY

Barrow, D. W., Fraser, A. G., Hartley, D. F., Landy, B., and Needham, R. M., File Handling at Cambridge University, Proceedings Spring Joint Computer Conference, pp. 163-167, 1967.

Bleier, R. E., Treating hierarchical data structures in the SDC time-shared data management system (TDMS), ACM National Conference Proceedings, 1967.

Corbato, F. J., et al, The Compatible Time-Sharing System, MIT Press, Cambridge, 1962.

Daley, R. C., and Dennis, J. B., Virtual memory, processes and sharing in Multics, Communications of the ACM, May 1968.

Daley, R. C., and Neumann, P. G., A general purpose file system for secondary storage, Proceedings Fall Joint Computer Conference, 1965.

Dennis, J. B., Segmentation and the design of multi-programmed computer systems, Journal of the ACM, October 1965.

Dijkstra, E. W., The structure of the 'THE' multiprogamming system, ACM Symposium on Operating Systems Principles, Gatlinburg, Tennessee, October 1967.

Dijkstra, E. W., Complexity controlled by hierarchical ordering of function and variability, Working Paper for the NATO Conference on Computer Software Engineering, Garmisch, Germany, October 7-11, 1968.

Dixon, P. J., and Sable, D. J., DM-1 - A generalized data management system, Proceedings Spring Joint Computer Conference, 1967.

Henry, W. R., Hierarchical structure for data management, IBM Systems Journal, Volume 8, No. 1, 1969.

IBM Cambridge Scientific Center, CP-67/CMS Program Logic Manual, Cambridge, Massachusetts, April 1968.

IBM Corporation, IBM System/360 Time Sharing System Access Methods, Form Y28-2016-1, 1968.

Lett, Alexander S., and Konisgsford, William L., TSS/360: a time-shared operating system, Proceedings Fall Joint Computer Conference, 1968.

Lockemann, Peter C., and Knutsen, W. Dale, Recovery of disk contents after system failure, Communications of the ACM, 1968.

Madnick, Stuart E., Multi-processor software lockout, ACM National Conference Proceedings, August 1968.

Madnick, Stuart E., Design strategies for file systems: a working model, FILE/68 International Seminar on File Organization, Helsingør Denmark, November 1968.

Madnick, Stuart E., Modular approach to file system design, Proceedings Spring Joint Computer Conference, 1969.

Nelson, T. H., A file structure for the complex, the changing and the indeterminate, ACM National Conference Proceedings, August 1965.

Nelson, D. B., Pick, R. A., and Andrews, K. B., GIM-1 - A generalized information management language and computer system, Proceedings Spring Joint Computer Conference, 1967.

O'Neill, R. W., Experience using a time-shared multi-programming system with dynamic address relocation hardware, Proceedings Spring Joint Computer Conference, 1967.

Randell, B., Towards a methodology of computer system design, Working Paper for the NATO Conference on Computer Software Engineering, Garmisch, Germany, October 7-11, 1968.

Rappaport, R. L., Implementing multi-process primitives in a multiplexed computer system, S. M. Thesis MIT Department of Electrical Engineering, August 1968.

Rosen, Saul, Programming Systems and Languages, McGraw-Hill, New York, 1967.

Rosen, Saul, Electronic computers: a historical survey, ACM Computing Surveys, Volume 1, Nol, 1, p. 24, March 1969.

Rosin, Robert F., Supervisory and monitor systems, ACM Computing Surveys, Volume 1, No. 1, pp. 37-54, March 1969.

Saltzer, J. H., CTSS technical notes, MIT Project MAC Report MAC-TR-16, August 1965.

Saltzer, J. H., Traffic control in a multiplexed computer system, Sc.D Thesis, MIT Department of Electrical Engineering, August 1968.

Scherr, A. L., An analysis of time-shared computer systems, MIT Project MAC Report MAC-tr-18, June 1965.

Schwartz, Jules I., Coffman, Edward G., and Weissman, Clark, A general-purpose time-sharing system, Proceedings Spring Joint Computer Conference, 1964.

Schwartz, Jules I., and Weissman, Clark, The SDC time-sharing system revisited, ACM National Conference Proceedings, 1967.

Scientific Data Systems, SDS 940 Time-Sharing System Technical Manual, Santa Monica, California, August 1968.

Seawright, L. H., and Kelch, J. A., An introduction to CP-67/CMS, IBM Cambridge Scientific Center Report 320-2032, Cambridge, Massachusetts, September 1968.

Wilkes, M. V., Time-Sharing Computer Systems, pp. 75-90, American Elsevier Publishing Company, Inc., New York, 1968.