

A Fast File System for UNIX*

*Marshall Kirk McKusick, William N. Joy†,
Samuel J. Leffler‡, Robert S. Fabry*

Computer Systems Research Group
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA 94720

ABSTRACT

A reimplementaion of the UNIX file system is described. The reimplementaion provides substantially higher throughput rates by using more flexible allocation policies that allow better locality of reference and can be adapted to a wide range of peripheral and processor characteristics. The new file system clusters data that is sequentially accessed and provides two block sizes to allow fast access to large files while not wasting large amounts of space for small files. File access rates of up to ten times faster than the traditional UNIX file system are experienced. Long needed enhancements to the programmers' interface are discussed. These include a mechanism to place advisory locks on files, extensions of the name space across file systems, the ability to use long file names, and provisions for administrative control of resource usage.

Revised February 18, 1984

CR Categories and Subject Descriptors: D.4.3 [Operating Systems]: File Systems Management – *file organization, directory structures, access methods*; D.4.2 [Operating Systems]: Storage Management – *allocation/deallocation strategies, secondary storage devices*; D.4.8 [Operating Systems]: Performance – *measurements, operational analysis*; H.3.2 [Information Systems]: Information Storage – *file organization*

Additional Keywords and Phrases: UNIX, file system organization, file system performance, file system design, application program interface.

General Terms: file system, measurement, performance.

* UNIX is a trademark of Bell Laboratories.

† William N. Joy is currently employed by: Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, CA 94043

‡ Samuel J. Leffler is currently employed by: Lucasfilm Ltd., PO Box 2009, San Rafael, CA 94912

This work was done under grants from the National Science Foundation under grant MCS80-05144, and the Defense Advance Research Projects Agency (DoD) under ARPA Order No. 4031 monitored by Naval Electronic System Command under Contract No. N00039-82-C-0235.

TABLE OF CONTENTS

1. Introduction
2. Old file system
3. New file system organization
 - 3.1. Optimizing storage utilization
 - 3.2. File system parameterization
 - 3.3. Layout policies
4. Performance
5. File system functional enhancements
 - 5.1. Long file names
 - 5.2. File locking
 - 5.3. Symbolic links
 - 5.4. Rename
 - 5.5. Quotas

Acknowledgements

References

1. Introduction

This paper describes the changes from the original 512 byte UNIX file system to the new one released with the 4.2 Berkeley Software Distribution. It presents the motivations for the changes, the methods used to effect these changes, the rationale behind the design decisions, and a description of the new implementation. This discussion is followed by a summary of the results that have been obtained, directions for future work, and the additions and changes that have been made to the facilities that are available to programmers.

The original UNIX system that runs on the PDP-11[†] has simple and elegant file system facilities. File system input/output is buffered by the kernel; there are no alignment constraints on data transfers and all operations are made to appear synchronous. All transfers to the disk are in 512 byte blocks, which can be placed arbitrarily within the data area of the file system. Virtually no constraints other than available disk space are placed on file growth [Ritchie74], [Thompson78].*

When used on the VAX-11 together with other UNIX enhancements, the original 512 byte UNIX file system is incapable of providing the data throughput rates that many applications require. For example, applications such as VLSI design and image processing do a small amount of processing on a large quantities of data and need to have a high throughput from the file system. High throughput rates are also needed by programs that map files from the file system into large virtual address spaces. Paging data in and out of the file system is likely to occur frequently [Ferrin82b]. This requires a file system providing higher bandwidth than the original 512 byte UNIX one that provides only about two percent of the maximum disk bandwidth or about 20 kilobytes per second per arm [White80], [Smith81b].

Modifications have been made to the UNIX file system to improve its performance. Since the UNIX file system interface is well understood and not inherently slow, this development retained the abstraction and simply changed the underlying implementation to increase its throughput. Consequently, users of the system have not been faced with massive software conversion.

Problems with file system performance have been dealt with extensively in the literature; see [Smith81a] for a survey. Previous work to improve the UNIX file system performance has been done by [Ferrin82a]. The UNIX operating system drew many of its ideas from Multics, a large, high performance

[†] DEC, PDP, VAX, MASSEBUS, and UNIBUS are trademarks of Digital Equipment Corporation.

* In practice, a file's size is constrained to be less than about one gigabyte.

operating system [Feiertag71]. Other work includes Hydra [Almes78], Spice [Thompson80], and a file system for a LISP environment [Symbolics81]. A good introduction to the physical latencies of disks is described in [Pechura83].

2. Old File System

In the file system developed at Bell Laboratories (the “traditional” file system), each disk drive is divided into one or more partitions. Each of these disk partitions may contain one file system. A file system never spans multiple partitions.† A file system is described by its super-block, which contains the basic parameters of the file system. These include the number of data blocks in the file system, a count of the maximum number of files, and a pointer to the *free list*, a linked list of all the free blocks in the file system.

Within the file system are files. Certain files are distinguished as directories and contain pointers to files that may themselves be directories. Every file has a descriptor associated with it called an *inode*. An inode contains information describing ownership of the file, time stamps marking last modification and access times for the file, and an array of indices that point to the data blocks for the file. For the purposes of this section, we assume that the first 8 blocks of the file are directly referenced by values stored in an inode itself*. An inode may also contain references to indirect blocks containing further data block indices. In a file system with a 512 byte block size, a singly indirect block contains 128 further block addresses, a doubly indirect block contains 128 addresses of further singly indirect blocks, and a triply indirect block contains 128 addresses of further doubly indirect blocks.

A 150 megabyte traditional UNIX file system consists of 4 megabytes of inodes followed by 146 megabytes of data. This organization segregates the inode information from the data; thus accessing a file normally incurs a long seek from the file’s inode to its data. Files in a single directory are not typically allocated consecutive slots in the 4 megabytes of inodes, causing many non-consecutive blocks of inodes to be accessed when executing operations on the inodes of several files in a directory.

The allocation of data blocks to files is also suboptimum. The traditional file system never transfers more than 512 bytes per disk transaction and often finds that the next sequential data block is not on the same cylinder, forcing seeks between 512 byte transfers. The combination of the small block size, limited read-ahead in the system, and many seeks severely limits file system throughput.

The first work at Berkeley on the UNIX file system attempted to improve both reliability and throughput. The reliability was improved by staging modifications to critical file system information so that they could either be completed or repaired cleanly by a program after a crash [Kowalski78]. The file system performance was improved by a factor of more than two by changing the basic block size from 512 to 1024 bytes. The increase was because of two factors: each disk transfer accessed twice as much data, and most files could be described without need to access indirect blocks since the direct blocks contained twice as much data. The file system with these changes will henceforth be referred to as the *old file system*.

This performance improvement gave a strong indication that increasing the block size was a good method for improving throughput. Although the throughput had doubled, the old file system was still using only about four percent of the disk bandwidth. The main problem was that although the free list was initially ordered for optimal access, it quickly became scrambled as files were created and removed. Eventually the free list became entirely random, causing files to have their blocks allocated randomly over the disk. This forced a seek before every block access. Although old file systems provided transfer rates of up to 175 kilobytes per second when they were first created, this rate deteriorated to 30 kilobytes per second after a few weeks of moderate use because of this randomization of data block placement. There was no way of restoring the performance of an old file system except to dump, rebuild, and restore the file system. Another possibility, as suggested by [Maruyama76], would be to have a process that periodically

† By “partition” here we refer to the subdivision of physical space on a disk drive. In the traditional file system, as in the new file system, file systems are really located in logical disk partitions that may overlap. This overlapping is made available, for example, to allow programs to copy entire disk drives containing multiple file systems.

* The actual number may vary from system to system, but is usually in the range 5-13.