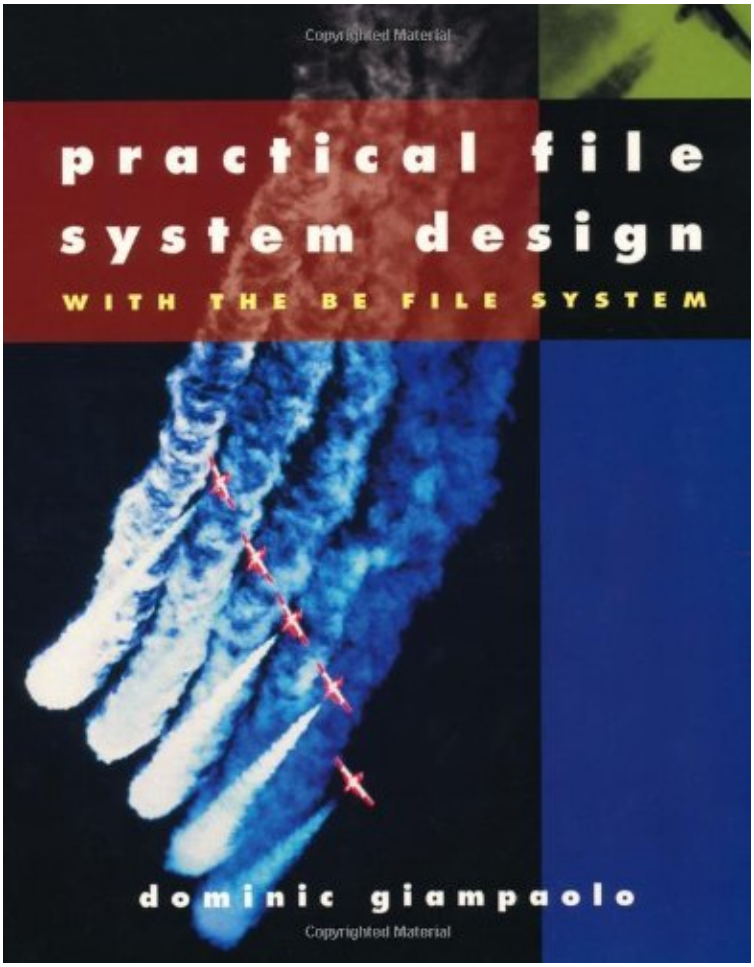
Practical File System Design

Von Dominic Giampaolo
*DOC | *audiobook | ebooks | Download PDF | ePub*



Produktinformation -Verkaufsrang: #1194755 in eBooksVerffentlicht am: 2013-08-29Erscheinungsdatum: 2013-08-29File Name: B008I3ARBI | File size: 51.Mb

Von Dominic Giampaolo : Practical File System Design before purchasing it in order to gage whether or not it would be worth my time, and all praised Practical File System Design:

KundenrezensionenHilfreichste Kundenrezensionen0 von 0 Kunden fanden die folgende Rezension hilfreich. Worth reading, but not the last word in file system designVon Ein KundeThis book may be slightly over-sold on its jacket ("guide to the design and implementation of file systems in general ... covers all topics related to file systems") but that's likely not the author's fault. It does provide intermediate levels of detail regarding many, perhaps most, areas of concern to file system designers and deserves a place in the library of anyone embarking on such a project - though people expecting a cookbook rather than a source of detailed ideas will be disappointed.The ideas are in general sound and representative of the current state of file system practice. The historical view is a bit Unix-centric - to state that the Berkeley Fast File System is the ancestor of most modern file systems is to ignore arguably superior and significantly earlier implementations from IBM, DEC, and others. This bias carries over into aspects of implementation as well,

such as use of the Unix direct/indirect/double-indirect mapping mechanism to manage contiguous 'block runs' without adding file address information to the mapping blocks to eliminate the need to scan them sequentially (save for the double-indirect blocks, which avoid the scan by establishing a standard run-length of 4 blocks - arrgh!) when positioning within the file - and the unbalanced Unix-style tree itself would almost certainly be better implemented as a b-tree variant (with its root in-line in the i-node) indexed on file address. And the text occasionally blurs the distinction between what the BFS chose to implement (a journal system that forced meta-data update transactions to be serialized) and what is possible (a multi-threaded journal supporting concurrent transactions simply by allowing each transaction to submit a log record for each individual change it makes - which would also support staged execution of extremely large transactions eliminating the log size as a constraint on them).Some of the choices made in BFS can be questioned, even in its particular use context. The 'allocation group' mechanism interacts in subtle ways with the basic file system block size, and given the relative and on-going improvement of disk seek time vs. rotational latency the value of locating related structures relatively near each other (though not actually adjacent) on disk may no longer justify the added complexity (though the effort to place file inodes immediately following the parent directory inode is likely worthwhile if a read-ahead facility exists to take advantage of it). The discussion of on-disk placement also ignores 'disks' that may in fact be composed of multiple striped units, which would further dilute the benefits of allocation groups; note that this would also complicate the read-ahead facility just mentioned, as would a shared-disk environment unless the disk unit itself performed the read-ahead and replication if present was taken into account (as in the Veritas file system, as I remember).Even the fundamental decision to make attributes indexable deserves closer examination, given the costs of indexing. Current hardware can perform a complete inode scan on a single-user workstation fast enough to satisfy the occasional random query and can scan the inodes for files within some limited sub-tree of the directory structure (e.g., a cluster of e-mail directories) relatively quickly for more common queries, and in a multi-user environment indexing individual attributes across all users is frequently not the behavior desired. Placing index management under explicit application control may be a better approach, perhaps by allowing the application to specify on attribute creation the index, if any, in which its value should be entered (thus preserving the ability to encapsulate the operation within a system-controlled transaction without the need for user-level transaction support) - and storing the index (perhaps by its inode) with the attribute for later change or deletion.Conspicuous by their omission are any mentions of how to manage very large allocation bit-maps (which one really must expect when other parts of the system are carefully crafted to handle 2**58-byte files) or the impact of a shared-disk environment (if BFS was intended to be limited to desk-top use this may be more understandable, but even desk-tops may soon have high-availability configurations). Security is mentioned briefly as a concern to be addressed later - but BFS's dynamic allocation of inodes from the general space pool makes this impossible, given that directory inode addresses can apparently be fed in from user-mode (the author does note this near the book's end, but fails to discuss possible remedies).The author also expresses regret in the introduction at not having had time to include more comparative information on other file systems, both current and historical. Perhaps he is leaving himself room to write a second book. I hope so: despite my comments above, this one was worthwhile - both on its own merits, and because of the lack of competition in this subject area.0 von 0 Kunden fanden die folgende Rezension hilfreich. I wish every technical writer were this good.Von Ein KundeI had wanted to buy this book for some time, butas a Unix Admin, I couldn't justify the money nor thestudy time. Well, now that I've bought it, I'mkicking myself for not doing so earlier. I havegained a much greater understanding of hashes, trees,filesystems, and databases. The book is an epitomeof clarity of thought and presentation. It's not often(never?) that I find a technical book that I wantto read cover to cover in one sitting! I only wishthat the author had more time to revisit the BeFSshort-comings that he mentions, and then GPL the endresult.0 von 0 Kunden fanden die folgende Rezension hilfreich. Excellent coverage of advanced file system topics.Von miguel_vaca@hotmail.comI found this book a useful insight into the mechanisms at work in modern file systems. I especially found the performance comparisons of popular file systems useful, giving the reader the ability to analyze the trade-offs of the differing implementations, and how those design decisions are based on the requirements of the file-system.

KurzbeschreibungThis is the new guide to the design and implementation of file systems in general, and the Be File System (BFS) in particular. This book covers all topics related to file systems, going into considerable depth where traditional operating systems books often stop. Advanced topics are covered in detail such as journaling, attributes, indexing and query processing. Built from scratch as a modern 64 bit, journaled file system, BFS is the primary file system for the Be Operating System (BeOS), which was designed for high performance multimedia applications. You do not have to be a kernel architect or file system engineer to use Practical File System Design. Neither do you have to be a BeOS developer or user. Only basic knowledge of C is required. If you have ever wondered about how file systems work, how to implement one, or want to learn more about the Be File System, this book is all you will need.* of other file systems, including Linux ext2, BSD FFS, Macintosh HFS, NTFS and SGI's XFS.* Allocation policies for

placing data on disks and discussion of on-disk data structures used by BFS * How to implement journaling* How a disk cache works, including cache interactions with the file system journal* File system performance tuning and benchmarks comparing BFS, NTFS, XFS, and ext2* A file system construction kit that allows the user to experiment and create their own file systemsKurzbeschreibungThis is the new guide to the design and implementation of file systems in general, and the Be File System (BFS) in particular. This book covers all topics related to file systems, going into considerable depth where traditional operating systems books often stop. Advanced topics are covered in detail such as journaling, attributes, indexing and query processing. Built from scratch as a modern 64 bit, journaled file system, BFS is the primary file system for the Be Operating System (BeOS), which was designed for high performance multimedia applications. You do not have to be a kernel architect or file system engineer to use Practical File System Design. Neither do you have to be a BeOS developer or user. Only basic knowledge of C is required. If you have ever wondered about how file systems work, how to implement one, or want to learn more about the Be File System, this book is all you will need.* of other file systems, including Linux ext2, BSD FFS, Macintosh HFS, NTFS and SGI's XFS.* Allocation policies for placing data on disks and discussion of on-disk data structures used by BFS * How to implement journaling* How a disk cache works, including cache interactions with the file system journal* File system performance tuning and benchmarks comparing BFS, NTFS, XFS, and ext2* A file system construction kit that allows the user to experiment and create their own file systemsSynopsis This work provides an in-depth look at the BeOS from the vantage point of the operating sustem's file system design and implementation issues in general, and looks at how those issues were handled in the development of the BeOS. It provides a guide for progammers and developers to the system specific feature of the BeOS file system that will enable easier and higher performance application development of the BeOS.