# The Design and Implementation of the 4.4BSD Operating System



Marshall Kirk McKusick
Keith Bostic
Michael J. Karels
John S. Quarterman

*The Design and Implementation of the*

# 4.4BSD Operating System

*The Design and Implementation of the*

# 4.4BSD Operating System

*Marshall Kirk McKusick*
Consultant

*Keith Bostic*
Berkeley Software Design, Inc.

*Michael J. Karels*
Berkeley Software Design, Inc.

*John S. Quarterman*
Texas Internet Consulting

**ADDISON–WESLEY**

Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

This book is in the **Addison-Wesley UNIX and Open Systems Series**

Series Editors: *Marshall Kirk McKusick* and *John S. Quarterman*
Publishing Partner: *Peter S. Gordon*
Associate Editor: *Deborah R. Lafferty*
Associate Production Supervisor: *Patricia A. Oduor*
Marketing Manager: *Bob Donegan*
Senior Manufacturing Manager: *Roy E. Logan*
Cover Designer: *Barbara Atkinson*
Troff Macro Designer: *Jaap Akkerhuis*
Copy Editor: *Lyn Dupré*
Cover Art: *John Lasseter*



**Library of Congress Cataloging-in-Publication Data**

```
The design and implementation of the 4.4BSD operating system /
  Marshall Kirk McKusick ... [et al.].
    p. cm.
    Includes bibliographical references and index.
    ISBN 0-201-54979-4
    1. UNIX (Computer file) 2. Operating systems (Computers)
    I. McKusick, Marshall Kirk.
  QA76.76.063D4743 1996              96-2433
  005.4'3--dc20                     CIP
```



Text printed on recycled and acid-free paper.
ISBN 0201549794
1112131415 MA 03 02 01

This facility allows buffers in different parts of a process address space to be written atomically, without the need to copy them to a single contiguous buffer. Atomic writes are necessary in the case where the underlying abstraction is record based, such as tape drives that output a tape block on each write request. It is also convenient to be able to read a single request into several different buffers (such as a record header into one place and the data into another). Although an application can simulate the ability to scatter data by reading the data into a large buffer and then copying the pieces to their intended destinations, the cost of memory-to-memory copying in such cases often would more than double the running time of the affected application.

Just as *send* and *recv* could have been implemented as library interfaces to *sendto* and *recvfrom*, it also would have been possible to simulate *read* with *readv* and *write* with *writev*. However, *read* and *write* are used so much more frequently that the added cost of simulating them would not have been worthwhile.

### Multiple Filesystem Support

With the expansion of network computing, it became desirable to support both local and remote filesystems. To simplify the support of multiple filesystems, the developers added a new virtual node or *vnode* interface to the kernel. The set of operations exported from the vnode interface appear much like the filesystem operations previously supported by the local filesystem. However, they may be supported by a wide range of filesystem types:

- Local disk-based filesystems
- Files imported using a variety of remote filesystem protocols
- Read-only CD-ROM filesystems
- Filesystems providing special-purpose interfaces—for example, the **/proc** filesystem

A few variants of 4.4BSD, such as FreeBSD, allow filesystems to be loaded dynamically when the filesystems are first referenced by the *mount* system call. The vnode interface is described in Section 6.5; its ancillary support routines are described in Section 6.6; several of the special-purpose filesystems are described in Section 6.7.

## 2.7 Filesystems

A regular file is a linear array of bytes, and can be read and written starting at any byte in the file. The kernel distinguishes no record boundaries in regular files, although many programs recognize line-feed characters as distinguishing the ends of lines, and other programs may impose other structure. No system-related information about a file is kept in the file itself, but the filesystem stores a small amount of ownership, protection, and usage information with each file.

DOCKET ALARM

# Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

## Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

## Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

## Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

## API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

**LAW FIRMS**
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

**FINANCIAL INSTITUTIONS**
Litigation and bankruptcy checks for companies and debtors.

**E-DISCOVERY AND LEGAL VENDORS**
Sync your system to PACER to automate legal marketing.