RV College of Engineering

Go, change the world

Title: Create your own file system using file system API, superblocks, inode and data block management.

Ruchitha M (1RV22CS165) Sanika Kamath(1RV22CS176)

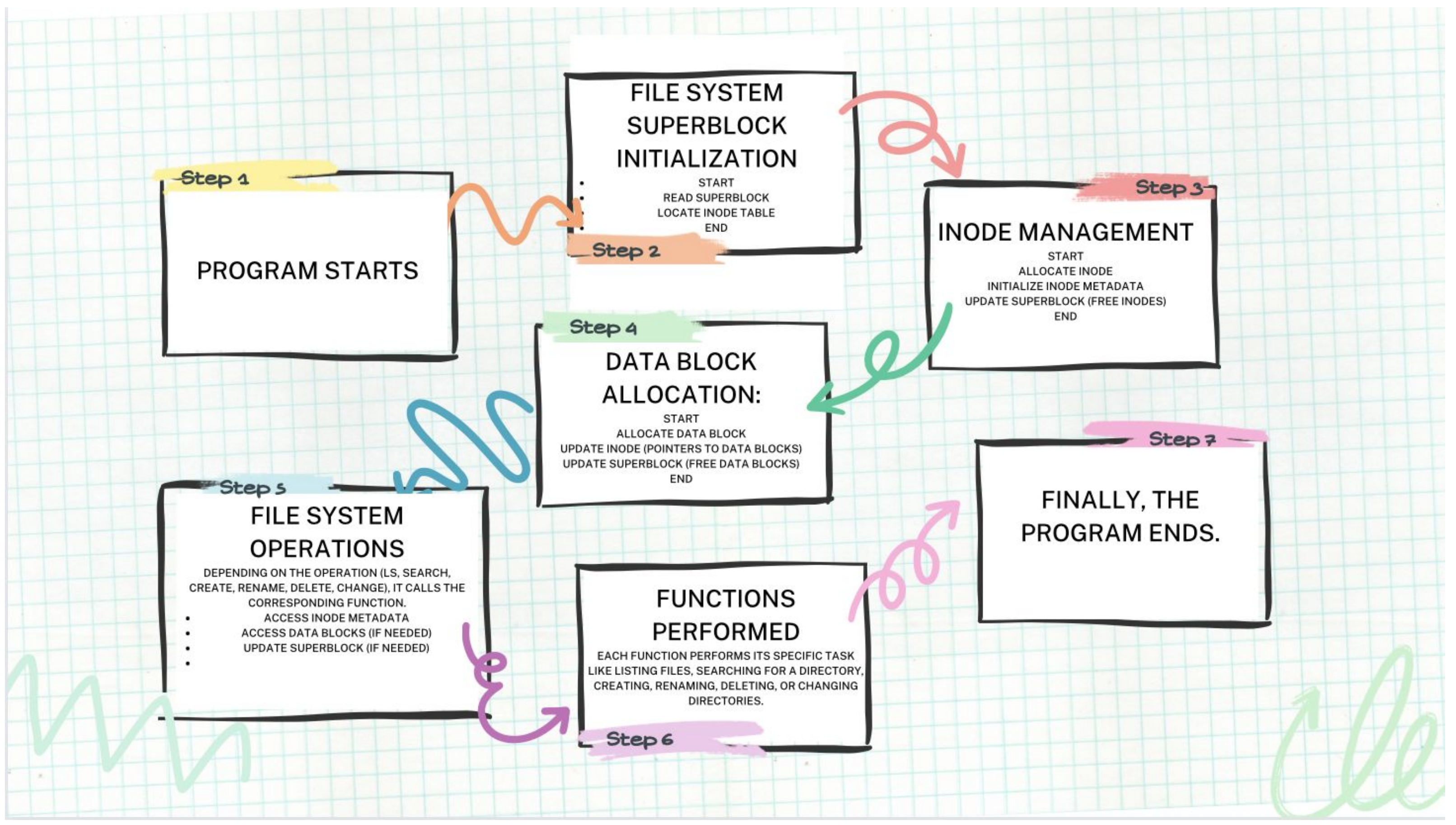
Department of Computer Science and Engineering

RV College of Engineering, Bangalore – 560059, INDIA

Motivation

Creating an advanced file system management solution designed to seamlessly handle diverse file operations, streamline navigation, and enhance organizational capabilities. Empowering users with a comprehensive array of functionalities including file listing, directory navigation, efficient file copying, moving, and deletion, robust file and directory creation, content reading and writing, dynamic directory path printing, and precise file or directory searching for unparalleled efficiency and productivity.

Flow Diagram


Step 1
PROGRAM STARTS
Step 2
FILE SYSTEM SUPERBLOCK INITIALIZATION
START
READ SUPERBLOCK
LOCATE INODE TABLE
END
Step 3
INODE MANAGEMENT
START
ALLOCATE INODE
INITIALIZE INODE METADATA
UPDATE SUPERBLOCK (FREE INODES)
END
Step 4
DATA BLOCK ALLOCATION:
START
ALLOCATE DATA BLOCK
UPDATE INODE (POINTERS TO DATA BLOCKS)
UPDATE SUPERBLOCK (FREE DATA BLOCKS)
END
Step 5
FILE SYSTEM OPERATIONS
DEPENDING ON THE OPERATION (LS, SEARCH, CREATE, RENAME, DELETE, CHANGE), IT CALLS THE CORRESPONDING FUNCTION.
ACCESS INODE METADATA
ACCESS DATA BLOCKS (IF NEEDED)
UPDATE SUPERBLOCK (IF NEEDED)
Step 6
FUNCTIONS PERFORMED
EACH FUNCTION PERFORMS ITS SPECIFIC TASK LIKE LISTING FILES, SEARCHING FOR A DIRECTORY, CREATING, RENAMING, DELETING, OR CHANGING DIRECTORIES.
Step 7
FINALLY, THE PROGRAM ENDS.

System Calls Made

Memory Allocation System Calls:

- `malloc()`: This system call is part of the C standard library and is used to dynamically allocate memory during runtime. In the mini filesystem, `malloc()` is utilized to allocate memory for `struct Inode` instances when creating files. Each file in the filesystem is represented by an inode, and dynamic memory allocation allows for flexibility in managing file metadata and content.

Input/Output System Calls:

- `printf()`, `scanf()`, and `puts()`: These system calls are fundamental for interacting with users through the command-line interface. `printf()` is used to display formatted output, providing information about filesystem operations, directory contents, and file contents to the user. `scanf()` enables the program to accept formatted input from the user, allowing them to enter commands and input data. `puts()` is similar to `printf()` but simpler, used for outputting strings to the standard output. Together, these system calls enable user interaction and feedback, making the filesystem accessible and usable.

Filesystem Management System Calls:

- `open()`, `read()`, `write()`, and `close()`: Although not directly used in the provided code, these system calls are fundamental for interacting with the filesystem of the underlying operating system. In a full-fledged filesystem implementation, these calls would be utilized for reading from and writing to files stored on disk. For instance, `open()` would be used to open files, `read()` for reading data from files, `write()` for writing data to files, and `close()` for closing files after operations are completed. While the provided code simulates filesystem operations using data structures and algorithms within the program's memory, these system calls would be essential for actual filesystem interactions in a real-world scenario.

Sleep System Call:

- `sleep()`: This system call is used to suspend the execution of the program for a specified number of seconds. In the provided code, `sleep()` is utilized within the main loop of the program to introduce a delay, pausing the execution and waiting for user input. This periodic waiting allows the program to remain responsive while waiting for user commands, enhancing user experience by preventing excessive CPU utilization and unnecessary resource consumption.

RELEVANCE TO OS COURSE

Implementing a file system in an operating system course is crucial for hands-on learning and understanding system components. It provides practical experience in coding, debugging, and problem-solving. Students grasp real-world applications, such as data storage management and access control, fostering critical thinking skills. They learn about trade-offs in design, performance, and security considerations. This practical exercise prepares them for advanced topics like distributed systems. Overall, it equips students with essential skills and knowledge for careers in software development and system administration.

Problem Statement

Creating an advanced file system management solution designed to seamlessly handle diverse file operations, streamline navigation, and enhance organizational capabilities. Empowering users with a comprehensive array of functionalities including file listing, directory navigation, efficient file copying, moving, and deletion, robust file and directory creation, content reading and writing, dynamic directory path printing, and precise file or directory searching for unparalleled efficiency and productivity.

Applications

- Storage (NAS) devices, or distributed file systems. Custom file systems can be optimized for specific workloads, scalability, fault tolerance, and data integrity.
- File System Interoperability: Developing file systems that support interoperability with existing file systems or storage protocols.
- Specialized Applications: Creating file systems tailored to specific application domains or industries, such as multimedia content delivery, scientific computing, financial services, or healthcare.
- Prototype Development: Rapidly prototyping and iterating file system designs for new product ideas or research projects.

Reference

[1.] E. Zadok, R. Iyer, N. Joukov, G. Sivathanu, and C. P. Wright (2006), "On Incremental File System Development", ACM Transaction on Storage, Vol. 2, No. 2, pp. 1-33.
[2.] M. A. Halcrow (2004), "Demands, Solutions, and Improvements for Linux Filesystem Security", in the proceedings of the Linux symposium, Ottawa, Canada, pp. 269-286.
[3.] E. Zadok et. al (1999), "Extending File Systems Using Stackable Templates", in proceedings of the annual USENIX Technical Conference, USENIX Association, Monterey, CA, pp. 57-70.

Acknowledgements

The authors thanks Principal and HoD, Department of Computer Science and Engineering, RVCE for the kind support received for completion of the project.