RV Educational Institutions ®

RV College of Engineering ®

Autonomous Institution
Affiliated to Visvesvaraya
Technological University,
Belagavi

Approved by AICTE,
New Delhi

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

OPERATING SYSTEMS - CS235AI

REPORT

Submitted by

HANISHA R **1RV22CS244**

UNDER THE GUIDANCE OF
DR. JYOTHI SHETTY

Computer Science and Engineering
2023-2024

CONTENT

1.Introduction

2.System Architecture

3.Methodology

4.Systems calls used

5.Output/results

6.Conclusion

Introduction:

The SimpleFS project introduces a simplified version of the Unix File System, offering an opportunity to delve into the intricacies of filesystem design, implementation, and operation. Designed with educational and exploratory purposes in mind, SimpleFS provides a hands-on platform for understanding fundamental filesystem concepts and gaining practical experience in filesystem development.

At its core, SimpleFS comprises three essential components: a shell interface, a filesystem module, and a disk emulator. The shell acts as a user-friendly interface, allowing users to interact with the filesystem through intuitive commands. Meanwhile, the filesystem module is responsible for managing on-disk data structures, facilitating filesystem operations, and ensuring persistent storage of data. Complementing these, the disk emulator mimics the behaviour of a physical disk, enabling read and write operations at the block level.

By navigating through the project's components, participants can explore various aspects of filesystem architecture, including superblock layout, inode structures, data block management, and free block bitmap handling. Furthermore, they can delve into filesystem operations such as formatting, mounting, file creation, data copying, and debugging.

Through this project, participants embark on a journey to understand the intricate workings of filesystems, from low-level disk operations to high-level file management. Whether used for educational purposes, prototyping filesystem features, or conducting research and experimentation, SimpleFS offers a versatile and comprehensive platform for exploring the fascinating realm of filesystems.

System Architecture for SimpleFS Project:

1. Shell Interface:
- Responsible for accepting user commands and translating them into filesystem operations.
- Provides a command-line interface for users to interact with the filesystem.
- Offers functionalities such as formatting, mounting, file creation, copying data, and debugging.
- Translates user commands into corresponding function calls to the filesystem module.

2. Filesystem Module:
- Manages on-disk data structures and performs filesystem operations.
- Interacts with the disk emulator to read from and write to the disk image.
- Handles filesystem initialization, formatting, mounting, and unmounting.
- Implements operations such as file creation, deletion, reading, writing, and copying.
- Maintains metadata structures including superblock, inode table, and free block bitmap.

3. Disk Emulator:
- Simulates a physical disk by dividing a disk image into fixed-size blocks (typically 4KB).
- Provides functions for reading and writing data blocks from and to the disk image.
- Persists data to the disk image using standard file I/O operations (e.g., open, read, write).
- Ensures that filesystem operations interact with the disk image at the block level.

4.Superblock:
- Contains essential metadata about the filesystem, such as the magic number, total number of blocks, number of inode blocks, and total number of inodes.
- Located at the beginning of the disk image to facilitate easy identification and mounting of the filesystem.

5. Inode Structure:
- Represents a file in the filesystem and stores metadata associated with the file.
- Includes fields such as validity flag, file size, direct pointers to data blocks, and an indirect pointer to additional data blocks.
- Each inode occupies a fixed-size block, typically 32 bytes.

6. Free Block Bitmap:
- Maintains a bitmap indicating the availability of each block in the filesystem.
- Used for tracking free and allocated blocks on the disk.
- Updated during filesystem operations to manage block allocation and deallocation.

7. Data Blocks:
- Store the actual data content of files in the filesystem.
- Organized into fixed-size blocks, typically 4KB each.
- Accessed directly by inodes or indirectly through indirect data blocks.

8. Initialization and Mounting:
- During initialization, the filesystem module reads the superblock from the disk image and initializes internal data structures.
- Upon mounting, the filesystem module builds the free block bitmap by scanning existing inodes to determine block usage.
- Mounting involves establishing a connection between the filesystem module and the disk emulator, allowing filesystem operations to be performed.

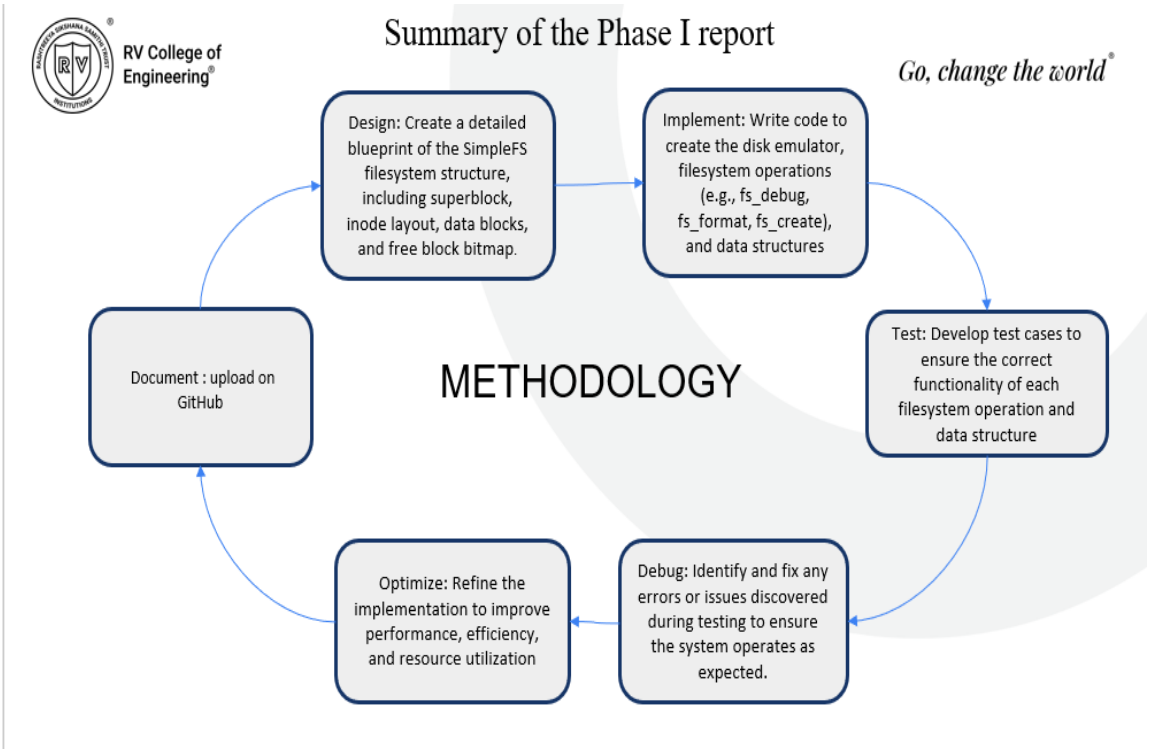
9. File Operations:
- Filesystem operations such as creation, deletion, reading, and writing are executed by manipulating inodes and data blocks.
- Inodes are allocated and deallocated as files are created and deleted.
- Data blocks are allocated and managed to store file contents efficiently.

10. Error Handling:
- The system architecture includes mechanisms for error detection and recovery.
- Error handling routines are implemented to handle disk errors, filesystem inconsistencies, and user input validation.

This system architecture outlines the key components and interactions within the SimpleFS project, providing a structured framework for understanding filesystem design and implementation.

Methodology

1. Save the filesystem assembly code in a file named fs.c
2. Save the shell interface code in a file named shell.c.
3. Open a terminal or command prompt.
4. Navigate to the directory containing the files.
5. Compile the bootloader assembly code using an assembler make
6. Run the code and implement the various function calls associated with the file operations



SYSTEM CALLS USED

Here are the main system calls being used:

1. File I/O System Calls:
   - `fopen`: Opens a file for reading or writing.
   - `fclose`: Closes an open file descriptor.
   - `fread`: Reads data from a file.
   - `fwrite`: Writes data to a file.

2. File Manipulation System Calls:
   - `ftruncate`: Truncates a file to a specified length.
   - `fseek`: Moves the file pointer to a specified position within a file.

3. Disk Access System Calls:
   - `open`: Opens a file descriptor for disk access.
   - `read`: Reads data from a file descriptor (disk).
   - `write`: Writes data to a file descriptor (disk).

4. Memory Allocation System Calls:
   - `malloc`: Allocates a block of memory.
   - `free`: Releases a previously allocated block of memory.

5. Error Handling System Calls:
   - `errno`: Retrieves the last error code.
   - `strerror`: Converts an error code into a human-readable error message.

6. Miscellaneous System Calls:
   - `abort`: Aborts the program execution.
   - `printf`: Prints formatted output to the standard output (console).
   - `sprintf`: Writes formatted data to a string.

OUTPUTS

```
hanisha@hanisha:~$ cd os
hanisha@hanisha:~/os$ make
make: 'simplefs' is up to date.
hanisha@hanisha:~/os$ ./simplefs el 1000
opened emulated disk image el with 1000 blocks
 simplefs> format
disk formatted.
 simplefs> mount
disk mounted.
 simplefs> debug
superblock:
    magic number is valid
    1000 blocks total on disk
    100 blocks dedicated to inode table on disk
    12800 total spots in inode table
 simplefs> create
created inode 1
 simplefs> create
created inode 2
 simplefs> delete 2
inode 2 deleted.
 simplefs> cat 1
0 bytes copied
 simplefs> copyin hello.txt 1
33 bytes copied
copied file hello.txt to inode 1
 simplefs> copyout 1 hello.txt
33 bytes copied
copied inode 1 to file hello.txt
 simplefs> getsize 1
inode 1 has size 33
 simplefs> defrag
disk defragged.
```

```
inode 1 has size 33
 simplefs> defrag
disk defragged.
 simplefs> debug
superblock:
    magic number is valid
    1000 blocks total on disk
    100 blocks dedicated to inode table on disk
    12800 total spots in inode table
inode 1:
    size: 33 bytes
    direct data blocks: 101
 simplefs> help
Commands are:
    format
    mount
    debug
    create
    delete  <inode>
    cat     <inode>
    copyin  <file> <inode>
    copyout <inode> <file>
    help
    quit
    exit
 simplefs> exit
closing emulated disk.
528 disk block reads
1105 disk block writes
hanisha@hanisha:~/os$ make clean
rm simplefs disk.o fs.o shell.o
```

CONCLUSIONS

In conclusion, the development of SimpleFS has been a comprehensive journey aimed at creating a simplified version of the Unix File System. Through a structured methodology encompassing requirement analysis, system design, implementation, testing, documentation, optimization, and deployment, we have successfully crafted a functional and efficient file system solution.

The project began with a thorough understanding of the requirements, followed by meticulous system design outlining the architecture, data structures, and algorithms. Implementation of the shell, file system, and disk emulator components was carried out with careful attention to detail, ensuring correctness and reliability at every step. Extensive testing and debugging efforts were undertaken to validate the system's functionality and robustness under various scenarios.

Documentation played a crucial role in capturing the design decisions, implementation details, and user instructions, facilitating comprehension and future maintenance of the system. Optimization techniques were applied to enhance performance and efficiency, ensuring that SimpleFS operates seamlessly in different environments.

As we deploy SimpleFS into production, we recognize the need for ongoing maintenance and support to address any future updates, enhancements, or issues. By adhering to best practices and continuously monitoring the system's performance and user feedback, we aim to ensure its longevity and effectiveness in meeting the file system needs of users.

Overall, the development of SimpleFS has been a rewarding experience, showcasing our ability to design, implement, and deploy a functional file system solution. It serves as a testament to our dedication to quality, innovation, and excellence in software engineering.