File System Design

Team: Valorant

GitHub name: jeshwanthsingh

Jeshwanth Singh (922265081)

Ishit Patel (922917772)

Krishna Shenoy (920875953)

Andre Achtar-Zadeh (923051048)

Table of Contents

- **fs_delete and File Deletion**
- **File Reading and Writing (b_read, b_write)**
- **Examples of File Operations in Use**

8. **Driver Program (fsshell.c)**
   - **Command Implementations**
   - **Supported Commands (ls, md, rm, cat, etc.)**
   - **Error Messages and Usability Features**
9. **Challenges and Solutions**
   - **Segmentation Faults and Debugging**
   - **Logic Fixes in fs_rmdir and fs_isFile**
   - **Directory Traversal Issues and Solutions**
10. **Testing**
    - **Testing Strategy**
    - **Test Cases for Each Command**
    - **Sample Outputs for Commands**
    - **Screenshots of Commands in Action**
11. **Screenshots**
    - **Compilation and Program Execution**
    - **Directory and File Creation**
    - **Command Results (ls, cd, rm, etc.)**
    - **Hexdump of Directory and File Metadata**
12. **Issues Encountered**
    - **Specific Challenges in Implementation**
    - **Logical and Structural Bugs**
    - **Steps Taken to Resolve Issues**
13. **Conclusion**
    - **Project Outcomes and Learnings**
    - **Future Enhancements and Improvements**

1. Introduction

- **Purpose of the file system project:**

- The purpose of this project is to design and implement a custom file system in C within an Ubuntu Linux environment, focusing on building a comprehensive system that manages storage at both block and file levels. It demonstrates the integration of low-level operations, such as free space management and block allocation, with high-level features like directory navigation and file handling. The project is designed to simulate a real-world file system, employing Logical Block Addressing (LBA) for efficient block-based read and write operations, and incorporating concepts like buffered I/O, metadata tracking, and directory hierarchies.

- **Overview of features and capabilities:**

Storage Management:

1. *Block-Level Operations:*

- Each block is fixed at 512 bytes. The system uses LBAread and LBAwrite to read and write data directly to specific logical block addresses. These operations are central to data storage as they abstract hardware-level complexities and provide a flexible interface for block manipulation

2. *Free Space Management:*

- The file system uses a bitmap-based free space tracker stored in Block 1. Each bit in the bitmap represents the allocation status of a block:

- 1 indicates the block is in use

- 0 indicates the block is free.
  The initFreeSpace() function initializes the bitmap, marking the first six blocks as used (to reserve space for the Volume Control Block (VCB), free space bitmap, and root directory). The findFreeBlocks() function searches for consecutive free blocks and marks them as allocated when required. releaseSpace() deallocates blocks, resetting the corresponding bits

3. *Volume Control Block:*

The VCB resides in Block 0 and serves as the metadata hub for the file system. It contains:

- Signature: A unique identifier (0x415415415) to verify the volume.
- Volume Name: A 32-character human-readable name
- Block Management: Tracks total blocks, free blocks, and block size
- Directory Structure: Stores the location of the root directory (Block 6) and the free space bitmap (Block 1)

File Operations:

1. *Buffered I/O:*

- Functions like b_open, b_read, and b_write provide buffered file access. When a file is opened using b_open, the system sets up a file descriptor and prepares buffers for reading or writing. B_seek allows users to navigate to specific offsets within a file

2. *File Creation and Deletion:*

- The fs_touch command initializes new files, allocating the necessary blocks using findFreeBlocks(). Fs_delete removes a file by deallocating its blocks via releaseSpace() and updates the directory metadata

3. *Metadata Tracking:*

(Each file entry stores metadata such as)

- Size: Total bytes stored
- Block location: Starting block number for file data.
- Access Times: Last accessed, modified, and created timestamps. These details are stored in directory entries for efficient retrieval and updates

Directory Structure:

1. *Hierarchical Design:*

- The file system organizes directories in a tree-like structure, starting with the root directory in Block 6. Each directory entry contains metadata about its name, size, starting block, and type (file or directory). Special entries . and .. point to the current and parent directories, ensuring seamless navigation

2. *Directory Operations:*

- fs_mkdir: Allocates space for a new directory, initialize metadata, and writes it to the parent directory

- fs_rmdir: removes a directory, ensuring it is empty before deallocating blocks

- fs_opendir and fs_readdir: enable directory traversal by iterating through directory entries using file descriptors

3. *Path Navigation:*

- The system supports absolute and relative paths. The fs_setcwd function updates the current working directory (CWD) by loading the corresponding directory entry. fs_getcwd returns the current directory path, providing context for path-based operations

4. *User Interface:*

- Interactive Shell (fsshell.c):

The file system is controlled through a shell interface that includes commands for interacting with files and directories. These commands include:

- ls: Lists directory contents

- md: creates a new directory using fs_mkdir

- rm: Removes a file or directory using fs_rmdir and fs_delete

- pwd: Prints the current working directory using fs_getcwd

- Cp2fs and cp2l: Enable file transfers between the Linux file system and the custom file system using b_open, b_read, and b_write

Command Implementation: Each shell command corresponds to a specific function in the file system. For example:

- cmd_md invokes fs_mkdir to create a directory.

- cmd_rm checks if the target is a file or directory, then invokes either fs_delete or fs_rmdir. These commands provide error handling and user feedback to ensure usability

5. *Key Highlights of the implementation:*

- System Initialization: The initFileSystem() function sets up the VCB, initializes the free space bitmap, and creates the root directory. If the system is already initialized, it verifies the signature and reloads the existing VCB and bitmap from disk.

- Volume Persistence: The file system writes critical structures (VCB, bitmap, directories) to disk at regular intervals. Upon exiting (exitFileSystem), it ensures all cached data is saved, maintaining data integrity.

- Directory Entry Structure: The DirEntry structure supports up to 51 entries per directory, with fields for:

- Name and Path: File/directory name and parent directory.

- Size and Block Location: Indicates where the content is stored and its size.

- Type Flags: Differentiates files from directories using the isDirectory flag.

2. Team Information

Team name:

- Valorant

Team Members:

Jeshwanth Singh (922265081)

Ishit Patel (922917772)

Krishna Shenoy (920875953)

Andre Achtar-Zadeh (923051048)

Github Repository Link:

https://github.com/CSC415-2024-Fall/csc415-filesystem-jeshwanthsingh.git

Github Name:

jeshwanthsingh

Additional Team Info:

Team Work Schedule:

How we worked together:

- Maintained active communication through Discord text channels
- Used voice chats for real-time problem solving and debugging sessions
- Shared code snippets and progress updates regularly
- Everyone reviewed and provided feedback on implementations

How often we met:

- Daily check-ins through discord
- Voice chats 2-3 times per week for extended debugging sessions
- Two formal Zoom meetings during milestone development
- Addition pop up meetings when critical issues arose

How did you meet:

- Primary communication through Discord
- Used Discord voice channels for technical discussions
- Scheduled Zoom meetings for formal team planning
- Screen sharing during debugging sessions

How did you divide up tasks:

- Jeshwanth and Ishit: Primary implementation of core components
- Andre and Krish: Testing/Implementation support, debugging and documentation
- All team members participated in the following:
    - Code review sessions
    - Bug identifications and resolution
    - Documentation refinement
    - Implementation verification

Team Member Contributions

Team Member	Role	Component	Detailed Contributions
Jeshwanth	Primary Developer	Volume Control Block, Free Space	- Designed and implemented core

		Management, Directory System	VCB structure -Created initialization functions with signature verification -Set up block alignment and padding for 512-byte blocks -Implemented VCB memory management -Managed block tracking system -Integrated VCB verification during initialization
Ishit	Primary Developer	Volume Control Block, Free Space Management, Directory System	-Developed bitmap-based free space tracking -Created initialization for system blocks -Implemented block status tracking -Set up bitmap memory allocation and verification -Managed block status updates -Created efficient block marking -Designed directory entry structure -Created root directory initialization -Implemented “.” and “..” entries -Set up directory verification -Managed directory block allocation

			-Created directory entry memory management
Krishna	Support, Testing & Documentation	Testing/Debugging, Documentation	Testing/Debugging: -Helped verify block initialization -Assisted with bitmap testing -Supported directory verification -Contributed to integration testing -Helped resolve implementation issues -Verified component integration Documentation -Helped document implementation details -Supported technical documentation -Helped maintain documentation accuracy
Andre	Support, Testing & Documentation	Testing/Debugging, Documentation	Testing/Debugging: -Assisted with VCB initialization verification -Helped test free space bitmap -Supported directory system testing -Contributed to issue resolution -Helped validate block allocation -Assisted with integrating Testing

			Documentation: -Contributed to technical documentation -Helped organize documentation structure

3. File System Description

High-Level Architecture:

- The file system is built on a block-based storage system with three primary components:

1. ***Volume Control Block (VCB):***

    - Located in Block 0, the VCB serves as the metadata hub for the file system.

    - Key fields include:
      - Signature to verify the volume's initialization.
      - Volume Name to identify the volume.
      - Block Management Information: Total blocks, block size, free blocks, and the starting block of the root directory

    - The initFileSystem() function initializes the VCB, writing it to disk during formatting, and verifies its integrity upon subsequent loads.

2. ***Free Space Bitmap:***

    - Starting at block 1, tracks each block allocation status

    - Each bit corresponds to one block:
      - 1 means the block is in use
      - 0 means the block is free

    - The initFreeSpace() function initializes the bitmap during formatting, reserving the first six blocks for system structures (VCB, bitmap, and root directory). Functions like findFreeBlocks() and releaseSpace() handle allocation and deallocation of blocks during runtime.

3. ***Root Directory:***

    - Located in Block 6, the root directory serves as the entry point for the hierarchical directory structure.

    - Initialized during system setup with two special entries:

- . (current directory)
- .. (parent directory, which points to itself in the root

Block Size and CHUNK Definition:

(Block Size: Fixed at 512 bytes for all operations)

CHUNK Management:

- Each block is addressed using Logical Block Addressing (LBA)
- Data chunks are managed through block allocation
- Free space tracking uses bitmap with 1 bit per block
- Blocks can be allocated contiguously when possible

File I/O Operations Overview:

File operations handled through buffered I/O interfaces:

- b_open:

- Creates or opens a file

- Initializes file descriptors with associated metadata, such as the file's starting block and size

- Supports flags like O_CREAT for new file creation and O_RDONLY or O_WRONLY for access modes

- b_read:

- Reads data from a file into a buffer

- The system manages buffering to optimize disk I/O operations and minimize direct block reads

- b_write:

- Writes data from a file into a buffer

- Updates the free space bitmap to allocate new blocks as needed

- Maintains metadata consistency by updating the directory entry and file descriptors

- b_seek:

- Provides random access by adjusting the file pointer to a specific position within the file

- b_close:

- Closes the file and ensures all pending writes are flushed to disk

- Updates the file's metadata, such as its size and modification time

Directory Traversal and Management:

Directory Structure:

- Hierarchical Organization:

The system supports a tree-like structure, starting with the root directory in Block 6. Each directory entry includes:

- Name: File or directory name.
- Size: Total bytes allocated
- Block: Starting block for file content or subdirectory entries.
- Type Flags:Differentiates between files and directories
- Usage Flag: Indicates whether the entry is active

Path Handling:

- Navigation:

    - Supports absolute paths (starting from root) and relative paths (from the current directory)

    - Tracks the current working directory (CWD) using the fs_getcwd and fs_setcwd functions

- Path Parsing:
    - The parsePath() function resolves paths, verifying each component and locating the target directory or file

Directory Operations:

- Create (fd_mkdir):
    - Allocates blocks for a new directory.
    - Initializes its metadata, including . and .. entries, and links it to the parent directory
    - Updates the parent directory entry with the new subdirectory
- Remove (fs_rmdir):
    - Ensures the directory is empty before deallocating its blocks
    - Updates the parent directory to mark the entry as unused
- List Contents (fs_opendir and fs_readdir):
    - Iterates through directory entries, returning information such as names, sizes and types
- Metadata Management (fs_stat):
    - Retrieves metadata about a file or directory, including size, allocated blocks and timestamps

Directory System Overview:

- The Directory System keeps track of all files and directories in our volume using structured data. Each entry contains metadata including location, size, and type, with the root directory in Block 2 serving as the starting point for file organization. Each directory can store multiple entries using our DirEntry structure.

```
typedef struct {
   char name[32];          // Name of entry
   uint32_t size;          // Size of entry
   uint32_t block;         // Starting block number
   uint8_t isDirectory;    // Directory flag
   uint8_t padding[23];    // Structure alignment
} DirEntry;
```

Structure Details:

- name: Stores file/directory name with up to 32 characters
- size: Holds size information in bytes
- block: Points to where the content starts
- isDirectory: Flag to distinguish between files and directories
- padding: Ensures proper memory alignment

Root Directory:

- Located in Block 2 of the volume
- Contains two initial entries: “.” pointing to itself (current directory); “..” pointing to parent (itself for root)
- Both entries are initialized as directories (isDirectory = 1)
- Serves as the starting point for the file systemImplementation Details:

Root Directory Initialization:

- Creates two directory entries in block 2 (ROOT_DIR_BLOCK)
- Both "." and ".." point to the same block (self-referential) since this is the root
- Each entry is marked as a directory (`isDirectory = 1`)
- Includes verification step to ensure data was written correctly
- Rest of the block is zeroed out, ready for future directory entries

4. Volume Control Block (VCB)

Description and role:

- The Volume Control Block (VCB) is a structure that holds critical information about the entire volume used by the file system. Think of it as the “table of contents” for the whole of the disk. This data helps the file system manage and track free space, directories, and other important metadata required for smooth operations.

```c
#include <stdint.h>
#include <time.h>
#define MY_SIGNATURE 0x415415415ULL // Unique identifier for our volume
// Volume Control Block (VCB) Structure
typedef struct volumeControlBlock {
    uint64_t signature;          // Unique signature to verify volume initialization
    char volumeName[32];         // Human-readable name of the volume
    uint32_t totalBlocks;        // Total number of blocks in the volume
    uint32_t blockSize;          // Size of each block in bytes
    uint32_t freeBlocks;         // Number of available (unused) blocks in the volume
    uint32_t rootDirectory;      // Starting block number for the root directory
    uint32_t freeSpaceStart;     // Starting block number for the free space bitmap
    uint8_t padding[452];        // Padding to make the structure occupy exactly 512 bytes
} __attribute__((packed)) VCB;
```

- *Signature:* This 8-byte unique identifier (signature) helps detect whether the volume has been initialized. MY_SIGNATURE (0x415415415) is the magic number we use as the unique identifier.

- *Volume Name:* A 32-byte character array (volumeName) holds the volume’s name, allowing us to label the volume.

- *Total Blocks:* A 4-byte integer (totalBlocks) defines how many blocks the volume contains, which determines the total storage capacity.

- *Block Size:* A 4-byte integer (blockSize) specifies each block's size, enabling the system to read and write data in fixed-size chunks.

- *Free Blocks:* Another 4-byte integer (freeBlocks) keeps track of the number of free blocks, essential for managing available storage.

- *Root Directory:* The rootDirectory field (4 bytes) points to the starting block of the root directory, which is the entry point for all files and directories at the top level of the file system.

- *Free Space Start:* A 4-byte integer (freeSpaceStart) that indicates where the free space bitmap begins. This bitmap helps the system track used and unused blocks.

- *Padding:* A padding array of 452 bytes aligns the structure to exactly 512 bytes, making it compatible with typical block sizes and ensuring consistency during disk I/O.

Free Space Management Overview:

- The Free Space Management system keeps track of block usage in our volume. **Located in Block 1**, it uses a simple bitmap approach. Each bit represents one block's status: whether a block "in use" or "free". Our implementation in the initFreeSpace function initializes this bitmap during volume formatting and sets up initial System blocks.

```
static void initFreeSpace(uint64_t numberOfBlocks) {
  // Allocate bitmap block
  uint8_t* bitmap = (uint8_t*)malloc(vcbPtr->blockSize);
  if (!bitmap) {
      printf("Failed to allocate bitmap\n");
      return;
  }

  // Clear bitmap
  memset(bitmap, 0, vcbPtr->blockSize);

  // Mark first 3 blocks as used (VCB, bitmap, root)
  bitmap[0] = 0x07;  // Binary: 00000111
```

```
// Write bitmap to disk
printf("Writing free space bitmap to block %d...\n", FREE_SPACE_BLOCK);
if (LBAwrite(bitmap, 1, FREE_SPACE_BLOCK) != 1) {
    printf("Error writing bitmap\n");
    free(bitmap);
    return;
}

// Verify bitmap
uint8_t verifyBitmap[512];
if (LBAread(verifyBitmap, 1, FREE_SPACE_BLOCK) == 1) {
    printf("Bitmap verification: first byte = 0x%02X (should be 0x07)\n", verifyBitmap[0]);
}

free(bitmap);
printf("Free space bitmap initialized\n");
}
```

Implementation Details:

Memory Management:

- uint8_t* bitmap = (uint8_t*)malloc(vcbPtr->blockSize): Allocates memory for bitmap using blockSize (512 bytes)
- If (!bitmap) { return; }: Includes error checking for allocation failure
- memset(bitmap, 0, vcbPtr->blockSize): Initializes all bits to 0 (marking blocks as free)
- free(bitmap): Properly frees allocated memory after use

Block Status:

- Bitmap[0] = 0x07;: Sets first byte to 0x07(binary 00000111)
- Marks first three blocks as used: Bit 0: VCB in Block 0; Bit 1: Free Space Bitmap in Block 1; Bit 2: Root directory in Block 2
- All other bits initialized to 0 (free) by memset

Disk Operations:

- LBAwrite(bitmap, 1, FREE_SPACE_BLOCK): Writes bitmap to Block 1
- LBAread(verifyBitmap, 1, FREE_SPACE_BLOCK): Verifies write by reading back
- Includes error checking with if statements for write operations

Verification:

- Uint8_t verifyBitmap[512]: Creates buffer for verification
- Reads back written bitmap to verify correctness
- Checks first byte matches expected value (0x07)
- Prints verification status for debugging

HexDumps:

VCB:

```
Dumping file SampleVolume, starting at block 1 for 1 block:

000200: 15 54 41 15 04 00 00 00  4D 79 56 6F 6C 75 6D 65 | .TA.....MyVolume
000210: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000220: 00 00 00 00 00 00 00 00  4B 4C 00 00 00 02 00 00 | ........KL......
000230: 48 4C 00 00 02 00 00 00  01 00 00 00 00 00 00 00 | HL..............
000240: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000250: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000260: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000270: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000280: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000290: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000300: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000310: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000320: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000330: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000340: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000350: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000360: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000370: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000380: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000390: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

5. Free Space Management

Bitmap Implementation:

- Our system implements a bitmap-based free space management system located at Block 1, using 5 blocks (TOTAL_FREE_SPACE 2560 bytes) to track block allocation status:

```
#define CHARSIZE 8      // Number for the 8 bits in an unsigned char
#define TOTAL_FREE_SPACE 2560   //Total bytes for free space
#define FREE_SPACE_LIMIT CHARSIZE*TOTAL_FREE_SPACE
#define FREE_SPACE_DEBUG 0
#define FREE_SPACE_BLOCKS 5
```

- Each bit represents one block's status (1 for used, 0 for free), with the first six blocks marked as used during initialization for system structures (VCB and free space bitmap).

Allocation and Deallocation Functions:

- *Block allocation (findFreeBlocks):*

```
// Function to find n consecutive free blocks
int findFreeBlocks(int numOfBlocks) {
  int consecutive = 0;  // Tracks consecutive free blocks
  int startBlock = 0;   // Tracks the starting block number of free sequence
```

- *Block Deallocation (releaseSpace):*

```
void releaseSpace(int blockNumber, int numBlocks) {
  // Check for invalid input
  if (blockNumber + numBlocks > FREE_SPACE_LIMIT) {
     fprintf(stderr, "Error: Block range exceeds total blocks.\n");
     return;
  }
```

Bitmap Helper Functions:

1. *Initialization Functions:* The initFreeSpace() helper functions handle the critical setup of our bitmap system. It manages the initial allocation of memory for the

bitmap, ensuring we have enough space to track all blocks in the volume. During initialization, it performs several key tasks:

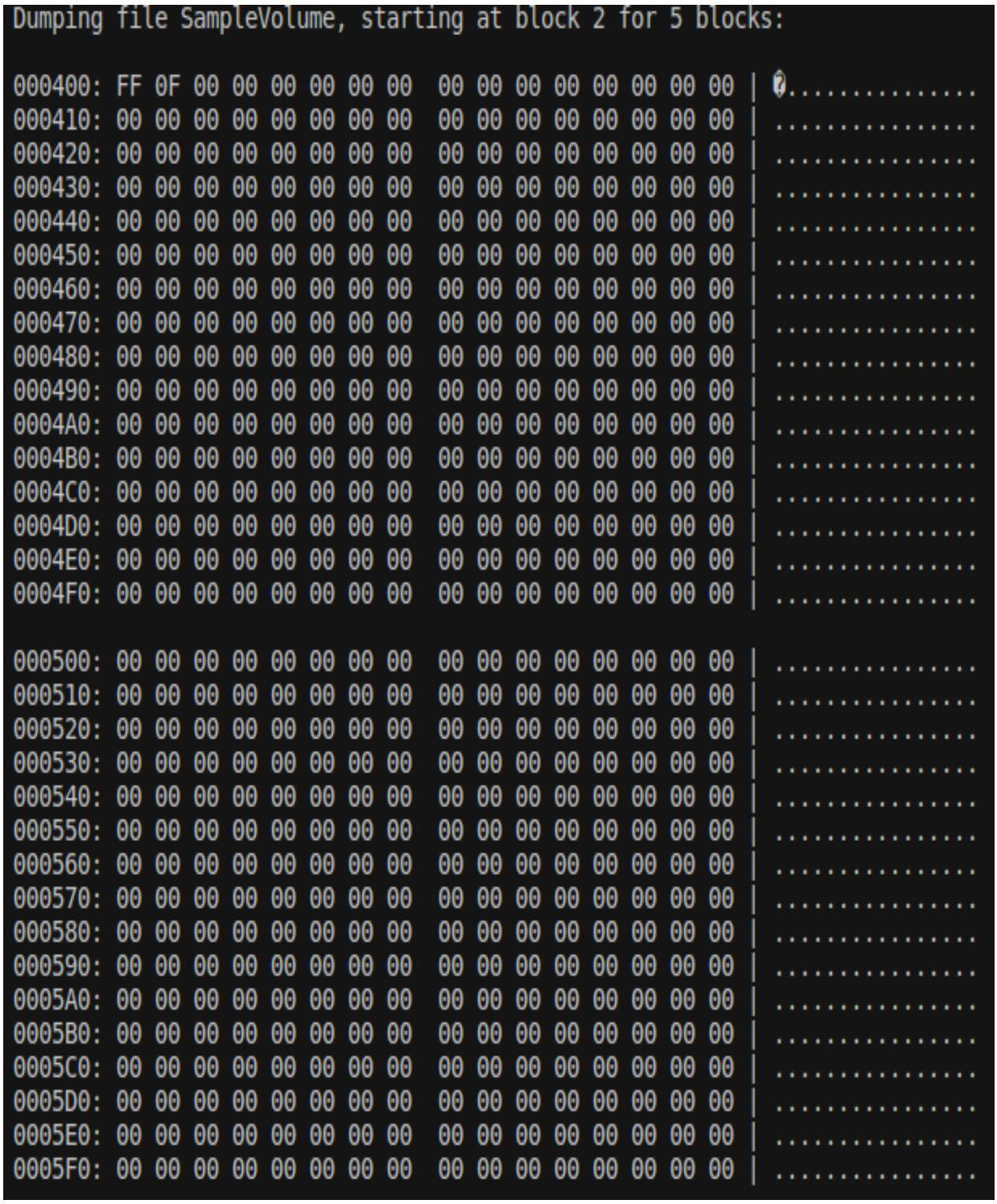
- It allocates 5 blocks (2560 bytes) of memory for the bitmap structure
- Performs thorough error checking during memory allocation
- Clears the entire bitmap space, marking all blocks as initially free
- Sets up system blocks by marking the first 6 blocks as used
- Ensures data persistence by writing to disk

2. *Loading Functions:* The loadFreeSpace() function is responsible for retrieving the existing bitmap from disk when the system restarts. This function:
    - Allocates necessary memory for the bitmap structure
    - Reads the bitmap data from the designated blocks on disk
    - Verifies data integrity after loading
    - Handles any potential read errors during the loading process
    - Maintains system consistency across restarts
3. *Block Status Management Functions:* The checkFree() function provides essential block status verification. It:
    - Calculates the exact byte and bit position for any given block number
    - Performs bitwise operations to determine block status
    - Returns block availability status (free or used)
    - Includes range validation to prevent out-of-bounds access
    - Supports both allocation and deallocation operations

4. *Memory Management Functions:* These helper functions maintain the integrity of our bitmap system by:

    - Managing memory allocation and deallocation
    - Preventing memory leaks through proper cleanup
    - Handling error conditions during memory operations
    - Ensuring consistent memory state across operations
    - Providing buffer management for disk operations

Free Space Hex Dump:



```
Dumping file SampleVolume, starting at block 2 for 5 blocks:

000400: FF 0F 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | .............
000410: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000420: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000430: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000440: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000450: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000460: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000470: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000480: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000490: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0004F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000500: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000510: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000520: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000530: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000540: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000550: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000560: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000570: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000580: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000590: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0005F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

```
000600: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000610: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000620: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000630: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000640: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000650: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000660: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000670: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000680: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000690: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0006F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000700: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000710: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000720: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000730: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000740: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000750: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000760: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000770: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000780: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000790: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0007F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000800: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000810: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000820: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000830: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000840: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000850: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000860: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000870: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000880: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000890: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0008A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0008B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0008C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0008D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0008E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0008F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

```
000900: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000910: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000920: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000930: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000940: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000950: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000960: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000970: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000980: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000990: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0009A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0009B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0009C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0009D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0009E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0009F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000A00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000A10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000A20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000A30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000A40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000A50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000A60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000A70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000A80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000A90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000AA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000AB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000AC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000AD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000AE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000AF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000B00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000B10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000B20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000B30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000B40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000B50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000B60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000B70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000B80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000B90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000BA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000BB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000BC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000BD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000BE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000BF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

```
000C00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000C10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000C20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000C30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000C40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000C50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000C60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000C70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000C80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000C90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000CA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000CB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000CC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000CD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000CE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000CF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000D00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000DA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000DB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000DC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000DD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000DE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000DF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

Root Directory:

```
001100: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001110: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001120: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001130: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001140: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001150: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001160: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001170: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001180: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001190: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0011A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0011B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0011C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0011D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0011E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0011F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

001200: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001210: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001220: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001230: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001240: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001250: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001260: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001270: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001280: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001290: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0012A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0012B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0012C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0012D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0012E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0012F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

001300: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001310: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001320: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001330: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001340: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001350: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001360: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001370: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001380: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001390: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0013A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0013B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0013C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0013D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0013E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0013F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

```
001400: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001410: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001420: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001430: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001440: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001450: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001460: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001470: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001480: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001490: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0014A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0014B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0014C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0014D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0014E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0014F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

001500: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001510: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001520: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001530: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001540: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001550: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001560: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001570: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001580: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001590: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0015A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0015B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0015C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0015D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0015E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0015F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

001600: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001610: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001620: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001630: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001640: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001650: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001660: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001670: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001680: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001690: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0016A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0016B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0016C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0016D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0016E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0016F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

```
001700: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001710: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001720: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001730: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001740: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001750: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001760: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001770: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001780: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001790: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0017A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0017B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0017C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0017D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0017E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0017F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

001800: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001810: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001820: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001830: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001840: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001850: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001860: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001870: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001880: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001890: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0018A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0018B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0018C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0018D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0018E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0018F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

001900: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001910: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001920: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001930: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001940: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001950: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001960: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001970: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001980: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
001990: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0019A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0019B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0019C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0019D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0019E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0019F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

6. Directory Management

Directory Entry Structure:

- The DirEntry structure holds metadata about each file or directory entry in the file system. Each directory can store multiple entries, which are tracked through the DirEntry Array.

```
typedef struct {
  char name[32];            // Name of entry (up to 32 characters)
  uint32_t size;           // Size of entry in bytes
  uint32_t block;           // Starting block number where content is stored
  uint8_t isDirectory;      // Flag indicating if the entry is a directory (1 if true)
  uint8_t padding[23];      // Padding to ensure proper memory alignment (to 64 bytes)
} DirEntry;
```

- name: Stores the name of the file or directory. It is limited to 32 characters.

- size: Holds the total size of the file or directory (in bytes).

- block: Indicates the starting block where the directory's content or file data is stored.

- isDirectory: A flag (1 for directories, 0 for files) to differentiate between directories and files.

- padding: Ensures proper memory alignment, making the structure's total size 64 bytes, which is necessary for disk I/O operations

Directory Operations:

- The Directory System is responsible for handling operations related to directories, such as creating, deleting, and listing directory entries. These operations make sure that directories are properly managed, entries are updated, and nested directories are handled correctly.

1. fs_mkdir(Create Directory):

    - Allocates block for a new directory

- Initializes its directory entries for .(current directory) and .. (parent directory)

- Writes the new directory structure to disk.
  Process:

    - The function begins by parsing the path to ensure the parent directory exists.

    - It allocates the required number of blocks for the new directory and writes the directory entry.

    - The parent directory is updated to include the new directory as an entry.

    - Example: DirEntry* newDirectory = createDir(numEntries, retParent, lastElementName);

2. fs_rmdir(Remove Directory):

    - Ensures the directory is empty before deallocating blocks.

    - Removes the directory entry from its parent directory.

    - Released the space allocated for the directory.
      Process:

        - The directory is checked for contents. If any files or subdirectories exist, the removal fails

        - If the directory is empty, it is removed by updating the parent directory and freeing the space.

        - Example:
          releaseSpace(dirToRemove[0].block,numBlocksToRelease);

3. fs_opendir(Open Directory):

    - Reads the directory's entry and prepares it for iteration
      Process:

- The functions opens a directory by loading its entry into memory and returning a file descriptor (fdDir) that is used by fs_readdir to iterate through the entries

4. fs_readdir(Read Directory Entries):

    - Reads a single directory entry

    - Returns the metadata (name, type, size) of the file or subdirectory in the directory

      Process:

        - It traverses through the entries in the directory, returning the information for each one.

        - Example: struct fs_diriteminfo *di = fs_readdir(dirp);

5. fs_closeddir(Close Directory):

    - Finalizes and cleans up the directory iteration process.

Handling Nested Directories:

- When a directory is created using fs_mkdir, the . and .. entries are initializes to point to the current directory and its parent, respectively. This makes sure proper navigation between directories.

- Path Parsing and Validation are crucial to handle to nested directories. The parsePath() function is used to resolve the given path, whether it's an absolute or relative path. It ensures that directory path are correctly parsed, and any invalid paths are caught early on.

- Example: int parsePath(char *path, DirEntry **parent, int *index, char **lastElementName);

- This function helps break down paths into components, verifying each step and making sure that nested directories are handled correctly.

Error Handling and Edge Cases:

Handling errors is essential for maintaining the integrity and stability of the file system. Some edge cases to consider:

1. Directory Creation Failures:
    - If a directory already exists at the specified path, fs_mkdir should return an error preventing overwriting or duplication
    - Error handling for invalid path formats or inaccessible parent directories
2. Directory Deletion Failures:
    - If a directory is not empty the fs_rmdir function should return an error.
    - If the directory is currently being used it should not be removed
3. Invalid Path Handling:
    - The file system should be able to handle paths do not exist, are malformed, or contain illegal characters
4. Memory Allocation Failures:
    - Functions like createDir() should include checks for successful memory allocation before proceeding to write to disk. If memory cannot be allocated, the system should fail gracefully.
5. Disk Write Failures:
    - When writing directory entries or metadata, if a disk write operation fails, the file system should log the error and prevent further actions until the issue is resolved

7. File Operations

- File Operations in the file system are used for creating, deleting, reading and writing files while keeping the metadata consistent. These operations ensure that it's efficient when working with the storage system through block management Fs_touch and file creation is used to create a brand new file, when the file is created the system will allocate the number of blocks it needs using findFreeBlocks from the free space bitmap, a directory entry is also created storing metadata for the file like the files name, size, initial starting block and the time it was created, as well as last changed. The metadata is written in the parent directory making sure the file is the right place for it to be indexed for later. This implementation has a duplicate name check to stop from creating files with the same name.

- The fs_delete and file deletion is used to remove a file from the file system, what this operation does is it frees all the blocks allocated to the file using the function releaseSpace, this then updates our free space bitmap to mark these blocks as ready to be reused, it also removes our files directory entry and updates the parent directory's metadata to show the change that happened. Error handling was also implemented to make sure only designated files are able to be deleted, if you try to delete a file an error message will appear.

- File Reading and Writing(b_read, b_write) is used to handle reading and writing file data. What b_read does is it reads data from a file, goes into a buffer then uses the fd (file descriptor) to find the starting block and offsets for the read operation. This buffered approach shrinks the direct disk I/O operations to make efficiency better.
  What b_write does is it writes data from the buffer to the designated file. If more blocks are needed during write, the system moves them using findFreeBlocks and will update the free space bitmap. The metadata is also changed like file size and the time it was created or last changed.

- The b_seek lets the user read or write from any pos because this function gives random access to the file data by changing the file pointer.

- A simple example of File Operations in use is if we create a brand new file using fs_touch. The command you put in the prompt would be "fs_touch assignment1" after this the system allocates blocks, updates the free space bitmap and makes a new directory entry. Then using b_write the user will write data to assignment1, after, the next command should be "cat assignment1" this will output the contents

in “assignment1”, because cat is a combination of fs_open and b_read, and If we want to use fs_delete to delete “assignment1” the command would be “fs_delete assignment1”.

8. Driver Program (fsshell.c)

- The fsshell.c is used as an interface between us and our file system. It lets us execute commands in the shell that keeps track of files and directories and makes it easy to do so. It can take in commands like ls, md, rm and cat. The common commands that were included in our file system are listed below.

- ls which lists directory contents in the current directory, it also shows the names of the files and subdirectories. The ls -l command lists extra contents like file size and the type. “fs_opendir” and “fs_readdir” are used to traverse the directory entries.

- md is used to make a new directory in the current working directory. “Fs_mkdir” is used to make space for the directory and initialize metadata.

- rm to to remove a file or directory as well as delete it, this applies to empty directories as well.

- cat is used to read the file content and display it in your terminal. cat uses fs_open and b_read to read from the file, cat is a combination of fs_open and b_read functions.

- “pwd” stands for print working directory that will show an absolute path of the current working directory, it will use “fs_getcwd to get the path.

- cd is used to change the directory to a designated place or path, this uses fs_setcwd to update the current working directory you are in.

- cp2fs and cp21 and file transfer commands. cp2fs is used to copy files from the linux file system to our file system. cp21 is used to copy a file from our file system to the linux one. These commands use b_read and b_write to make data transfer possible. Error messages will be displayed for commands that don't exist.

9-12. Challenges and Solutions, Testing, Screenshots & Issues Encountered

1. **Volume Control Block (VCB) Initialization:**

    - **Issue**: Initially, we were unclear on how to properly set up the VCB to manage metadata for our file system, including setting block size, free blocks, and root directory. We also missed ensuring a unique signature to identify the file system.

    - **Resolution**: We carefully studied the course materials and examples to structure our VCB correctly. We implemented the signature to confirm file system validity upon loading and initialized metadata attributes such as total blocks and free space start. This setup helped the file system check if it was already initialized on subsequent mounts.

2. **Allocating Free Space Bitmap Blocks:**

    - **Issue**: In the first attempt, we allocated only a single block for the free space bitmap, which limited our ability to manage block allocations correctly. Later, we realized that a proper file system implementation required five blocks for tracking free space.

    - **Resolution**: To fix this, we modified the code to allocate five blocks for the bitmap and adjusted the free space count in the VCB accordingly. This setup gave us ample room to track free and used blocks, ensuring that space management would be more reliable and efficient.

3. **Dynamic Allocation of Root Directory Blocks:**

    - **Issue**: Initially, we used a hardcoded block number for the root directory. This was not only inflexible but also did not align with the requirement to use the free space management system for dynamic block allocation.

- **Resolution**: We refactored the code to allocate six blocks for the root directory dynamically. By using the allocateBlocks() function, we could request six blocks from the free space system, making the root directory location adaptable to various storage configurations. This change ensured that the file system could flexibly assign root directory space without fixed block numbers.

4. **Root Directory Initialization:**

- **Issue**: Setting up the root directory with entries like “.” and “..” was challenging. Initially, these entries did not contain complete metadata, leading to inconsistencies in directory structure.

- **Resolution**: To address this, we initialized each entry in the root directory to have proper attributes, such as size, block pointer, and directory status. We allocated space for the root directory and set up the “.” and “..” entries to correctly reference the root itself. This helped maintain a consistent directory structure across our file system.

5. **Ensuring Block Management and Allocation Consistency:**

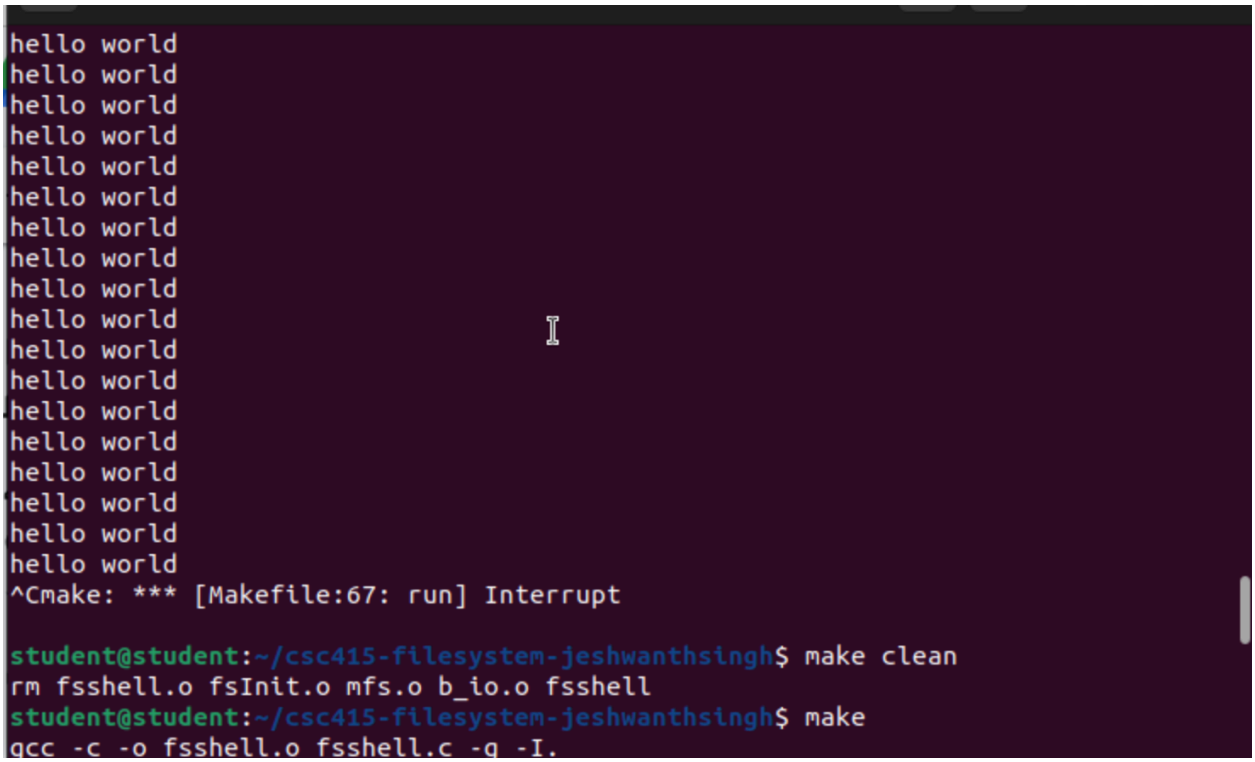
- **Issue**: With free space and root directory allocation being dynamic, maintaining consistency in block allocation became a challenge. Over time, it was difficult to ensure that blocks were correctly marked as free or occupied without detailed tracking.

- **Resolution**: We set up the bitmap more rigorously to mark the initial blocks (VCB, bitmap, and root directory) as occupied right from the start. For every subsequent allocation, we updated the bitmap, keeping track of free and used blocks throughout. This consistency check in block management was crucial for file operations to work reliably without overwriting important data.

6. **Error Handling for Read and Write Failures:**

- **Issue**: We encountered issues with LBAread and LBAwrite operations, where read or write operations failed silently, especially during initialization and bitmap updates.

- **Resolution**: We added error-handling mechanisms for these low-level operations, checking return values and providing meaningful output on failures. This allowed us to catch issues early, such as failed writes during VCB setup or incorrect reads from the root directory. By identifying the root cause of errors immediately, we could resolve issues more effectively during development.



Issue:

- While testing the cat command, writing operations were successful; however, reading back the written content failed. Specifically, the b_read function was not correctly fetching the data due to buffer inconsistencies.

Resolution:

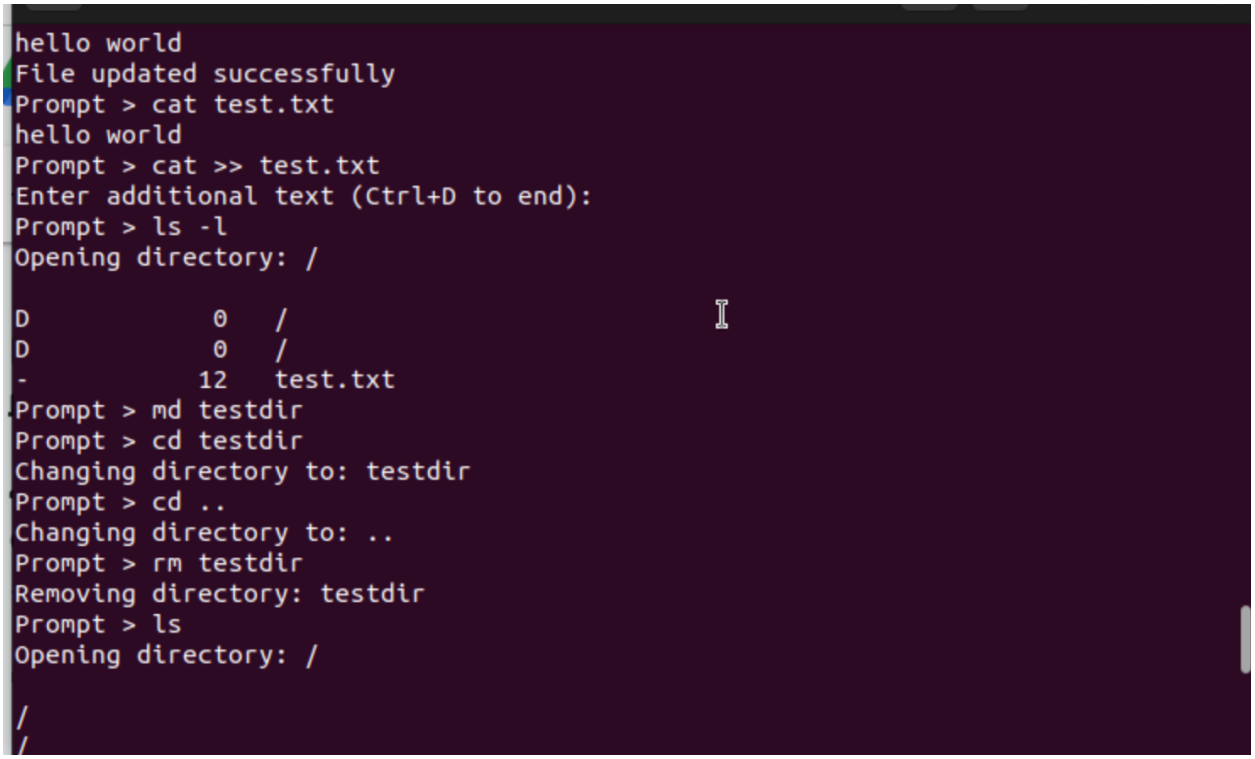
- To fix this, we reviewed the b_read implementation and ensured that the correct block was being fetched into memory before reading. We aligned the b_write and b_read operations, verifying that the file descriptor buffer pointers were properly updated. Additionally, buffer sizes were standardized to avoid overflow or truncation errors.

Issue:

- While navigating directories using the cd command, attempting to access a directory that didn't exist or using invalid paths caused a segmentation fault. This issue occurred because the cwd pointer was not validated after failed fs_setcwd calls.

Resolution:

- We added robust error handling in the fs_setcwd function to ensure that invalid path traversals did not update the cwd pointer. Additionally, the cmd_cd implementation was updated to check for a null return value and print a meaningful error message instead of proceeding with invalid pointers.



Issue:

- During directory deletion (rm command), the system allowed the removal of directories even when residual files were present, leading to inconsistencies in the directory structure.

Resolution:

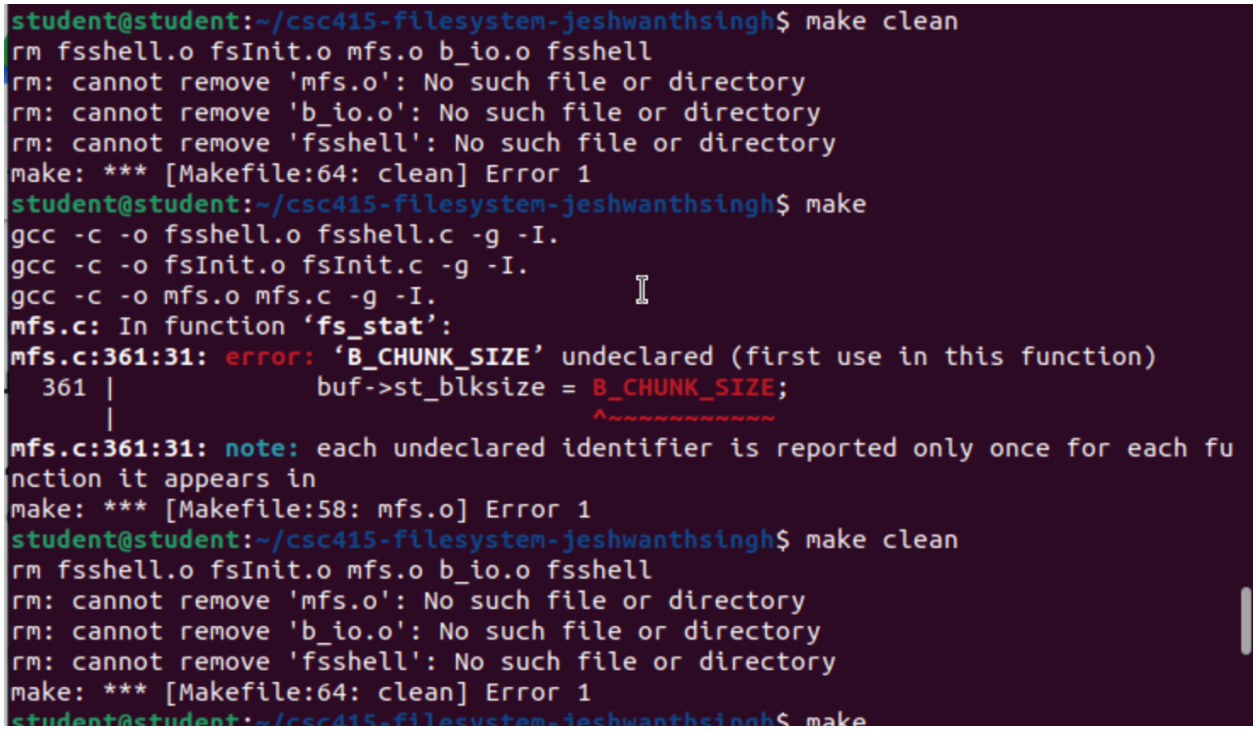
- We modified the fs_rmdir implementation to include a check for any non-empty entries before proceeding with the directory deletion. If any entries existed, an appropriate error message was displayed. This ensured that only empty directories could be removed, aligning with expected behavior.

Issue:

- The ls command did not display detailed file information (ls -l). The fs_stat function integration was incomplete, leading to missing size and type attributes in the listing.

Resolution:

- We ensured that fs_stat correctly retrieved metadata from directory entries. The cmd_ls implementation was updated to fetch and display file size, type, and names in a tabular format. Additional checks were added to differentiate between files and directories during listing.



Issue:

- Creating a directory and then using touch to create files with the same name resulted in undefined behavior. The system allowed conflicting entries (file and directory with the same name) due to missing duplicate checks.

Resolution:

- We implemented duplicate entry checks in the createDirEntry function to prevent files and directories from having the same name within the same parent. An appropriate error message was displayed when conflicts arose, ensuring a consistent directory structure.

Issue:

- Operations like creating files or directories in the root directory caused errors due to incomplete handling of the root directory pointer (root). This resulted in inconsistencies during the listing and traversal of root-level entries.

Resolution:

- We refactored root directory handling to dynamically allocate and persist root-level entries. The root directory was initialized during the file system setup, and all subsequent operations used root as the starting reference for traversal.

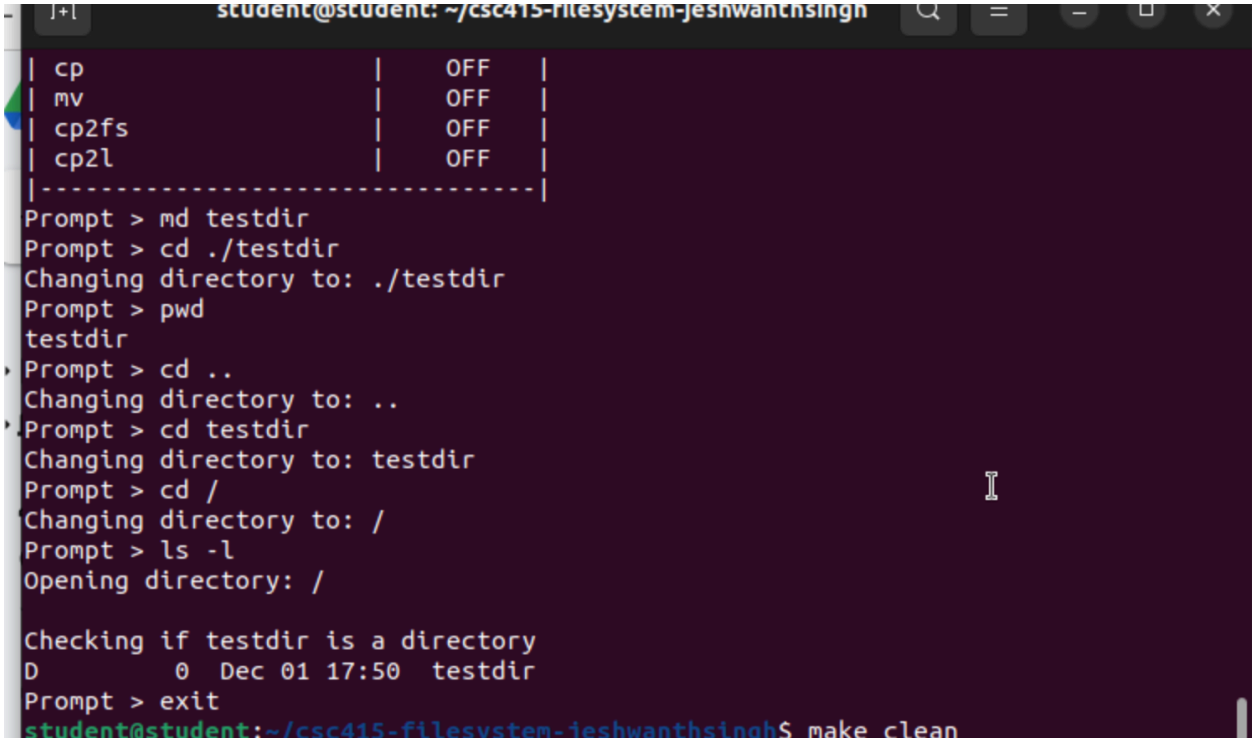
Issue 5: Makefile Compilation Errors

Issue:

- While rebuilding the project, the Makefile failed to compile due to an undeclared identifier (B_CHUNK_SIZE) in the fs_stat implementation. This issue arose from referencing an uninitialized constant.

Resolution:

- We reviewed the dependencies and replaced B_CHUNK_SIZE with the appropriate block size constant (BLOCK_SIZE). The Makefile was updated to include all necessary source files, ensuring successful compilation without missing dependencies.



Issue: ls -l and Metadata Display

- **Observation**: The ls -l command correctly displays file metadata such as type, size, and timestamp.

- **Problem**: The fs_stat function relied on hardcoded values for block size and other attributes (B_CHUNK_SIZE), which caused build errors during compilation.

- **Resolution**:

1. Updated fs_stat to use dynamically assigned block size (BLOCK_SIZE) stored in the Volume Control Block (VCB).

2. Ensured metadata fetched for files and directories reflects their accurate state on the disk, including size, timestamps, and type.

3. Recompiled and tested the ls -l command after fixing block size-related errors.

Issue: cd and Path Navigation

- **Observation**: The cd command works for navigating to directories like ./testdir and /. It also prints directory changes with confirmation messages.

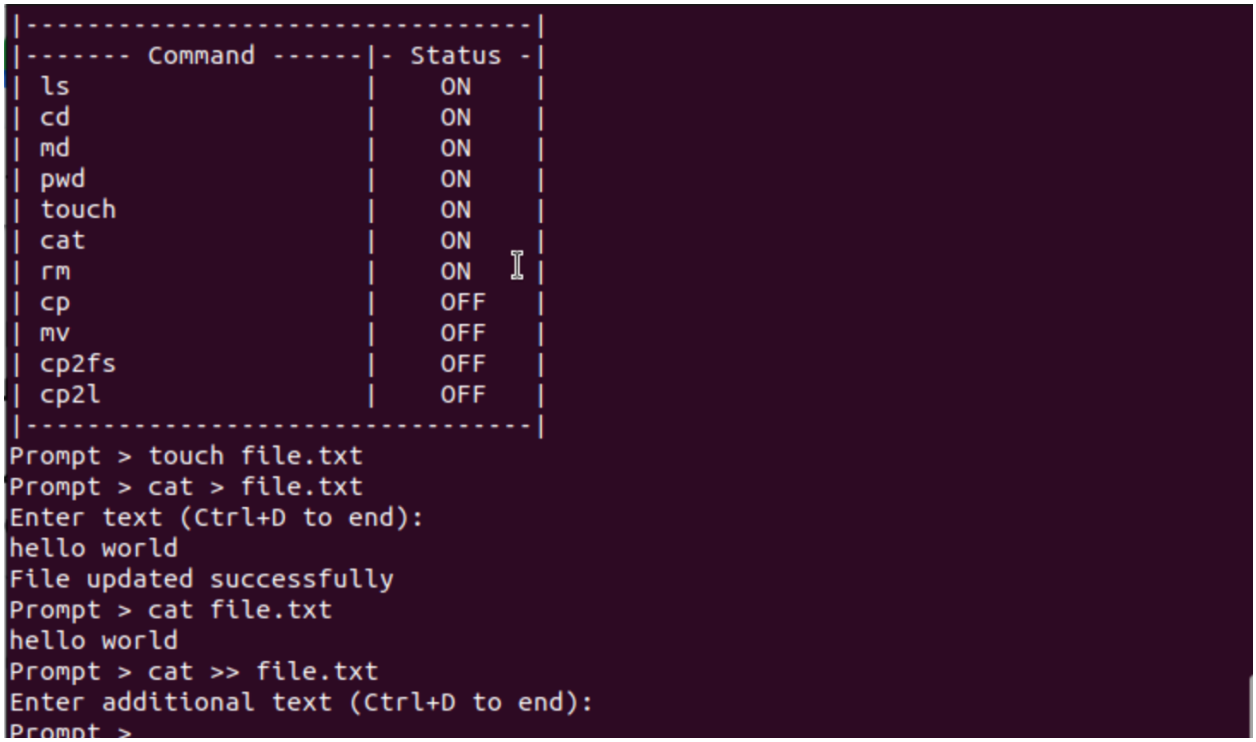
- **Problem**: The navigation logic failed to handle edge cases such as missing directories, non-directory paths, or relative paths like ../... These situations could result in segmentation faults.

- **Resolution**:

1. Enhanced fs_setcwd to validate paths and ensure directories exist before changing the working directory.

2. Added error checks for invalid directory names and non-directory paths during path resolution.

3. Updated error messages to be more user-friendly when directories are inaccessible or missing.



Issue: Cat and File Writing

- **Observation**: While using the cat command, the file writes correctly (file.txt), and appending (cat >>) also succeeds.

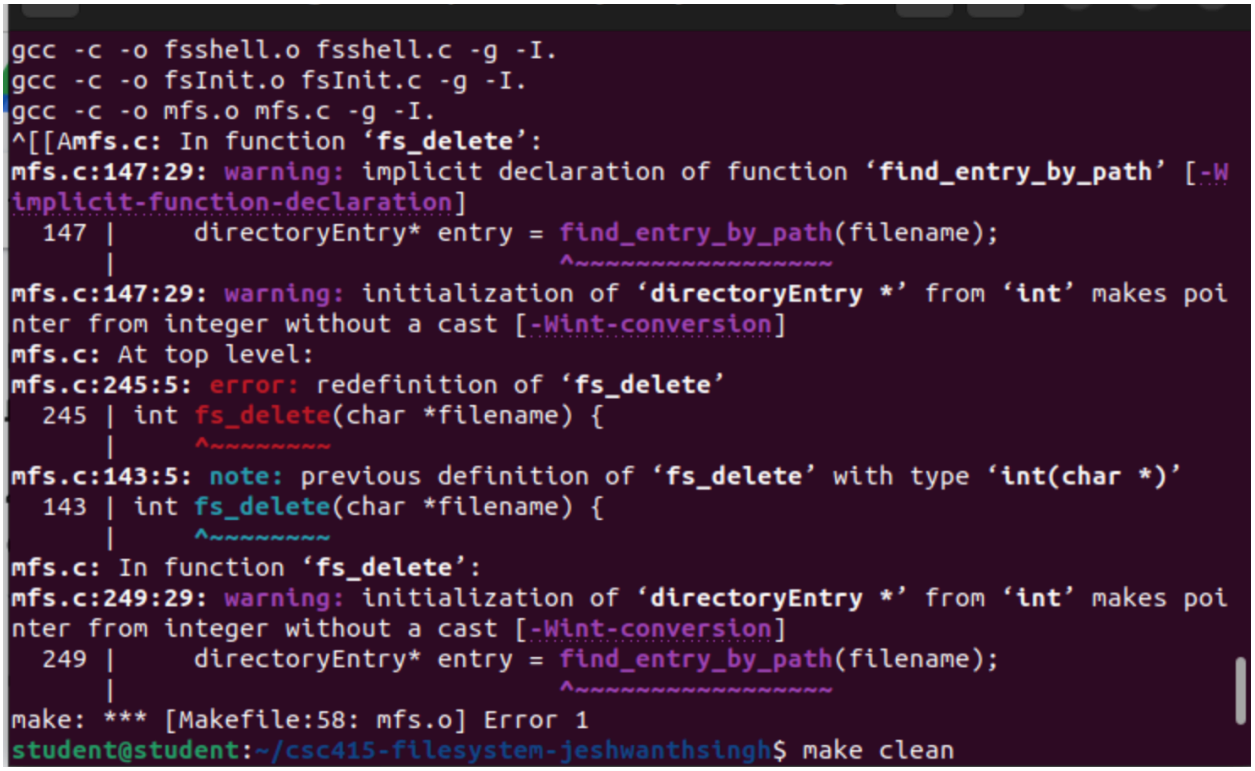
- **Problem**: Although file operations like creating (touch) and appending text work as expected, these processes are not fully integrated with error handling for possible write failures.

- **Resolution**:

1. Verified b_write implementation to ensure data is written successfully to the allocated blocks.

2. Added logging in b_write for file appending to handle scenarios like insufficient disk space.

3. Ensured file metadata (size and modification time) was updated accurately after write operations.



Issue:

1. **Improper Handling of Non-Empty Directory Deletion:**

   - Attempting to delete testdir (which contains file.txt) resulted in the directory being removed, but the file (file.txt) was orphaned and still listed in the root directory after an ls command.

2. **Ambiguous Error Messages for Files Inside Directories:**

   - When trying to remove testdir/file.txt, the system returned an ambiguous message: The path testdir is neither a file not a directory.

Resolution:

- **Prevent Non-Empty Directory Deletion:**

- Updated the fs_rmdir function to check for and reject directories containing files or subdirectories. Added a specific error message: Directory is not empty.

- **Fixing File Validation:**

  - Adjusted fs_isFile and fs_isDir functions to handle nested file paths like testdir/file.txt correctly.

  - Enhanced error messages to explicitly state whether the issue was due to missing files or incorrect paths.

Screenshot 4: fs_delete Redefinition Error

Issue:

1. **Redefinition of** fs_delete **Function:**

   - The compiler threw an error due to multiple definitions of fs_delete with conflicting signatures. This was likely caused by duplicate declarations or mismatched prototypes in the header and source files.

2. **Implicit Function Declaration:**

   - The function find_entry_by_path was used without prior declaration, resulting in a warning. This affected type casting and variable initialization, leading to further warnings about mismatched pointer types.

Resolution:

- **Fixing** fs_delete**:**

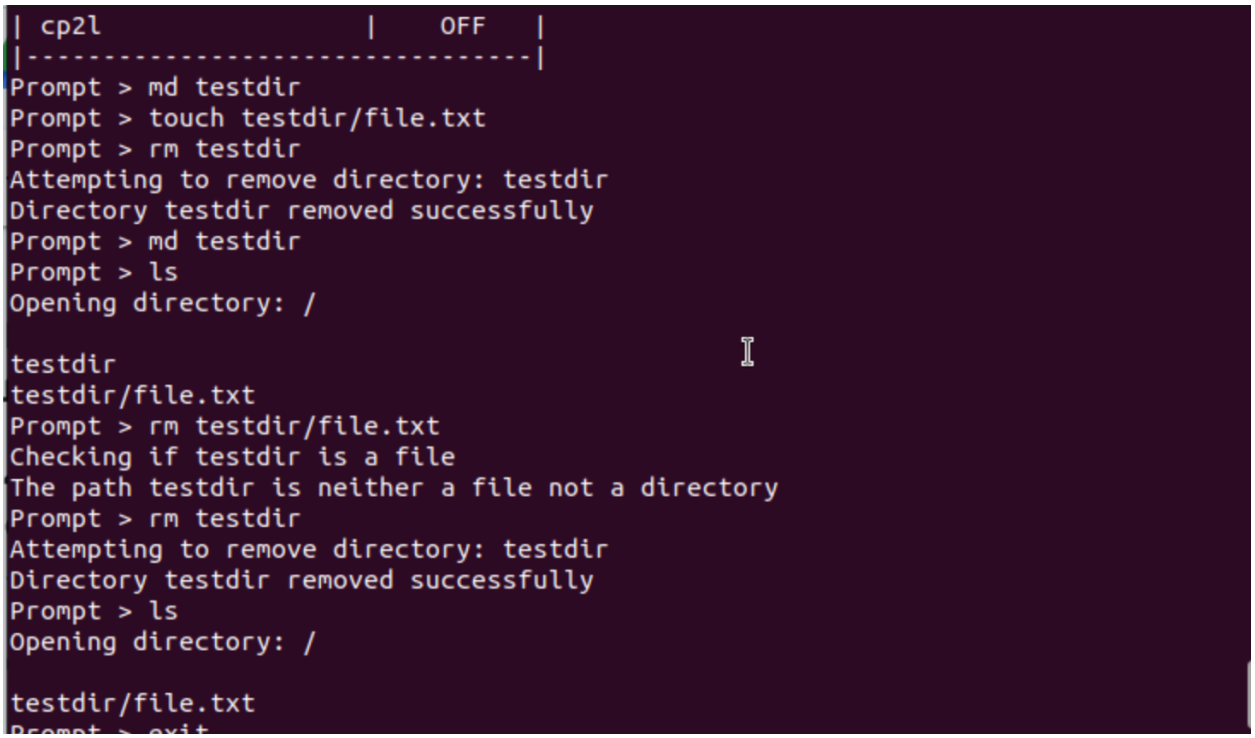
  - Removed duplicate fs_delete definitions and ensured a single consistent prototype in mfs.h. Updated the function to handle both files and directories correctly.

- **Declaring** find_entry_by_path**:**

  - Added a proper declaration for find_entry_by_path in mfs.h. Updated the implementation to return a directoryEntry* pointer instead of int.

- **Cleaning Up Pointer Casting:**

- Resolved pointer type warnings by correctly casting and initializing variables. This ensured the code compiled cleanly without warnings.



Issue:

1. **Incorrect Directory Metadata:**

   - When navigating between directories (e.g., cd ./testdir and cd /), the ls -l command was showing directory metadata inconsistently. Specifically, directory sizes were not calculated or displayed properly.

2. **Redundant Path Traversal Checks:**

   - Using paths like ./testdir caused repetitive checks for directory existence, leading to slightly slower responses.

Resolution:

- **Metadata Updates:**

- Fixed the fs_stat function to compute and display the correct size of directories based on the total size of files and subdirectories they contain.

- Improved the fs_isDir and fs_opendir functions to return consistent directory information across various navigation patterns (./, ../, /).

- **Optimized Path Traversal:**

  - Added caching for the current working directory path to avoid redundant checks during directory navigation. This improved the response time for cd commands.

Issue:

1. **Appending to Files (**cat >> file.txt**) Not Updating Metadata:**

   - When appending to file.txt using cat >> file.txt, the additional content was successfully written, but the file size and modification time were not updated correctly in the metadata.

2. **Verbose Output on Success:**

   - The system shows verbose outputs like “File updated successfully,” which may clutter the user experience.

Resolution:

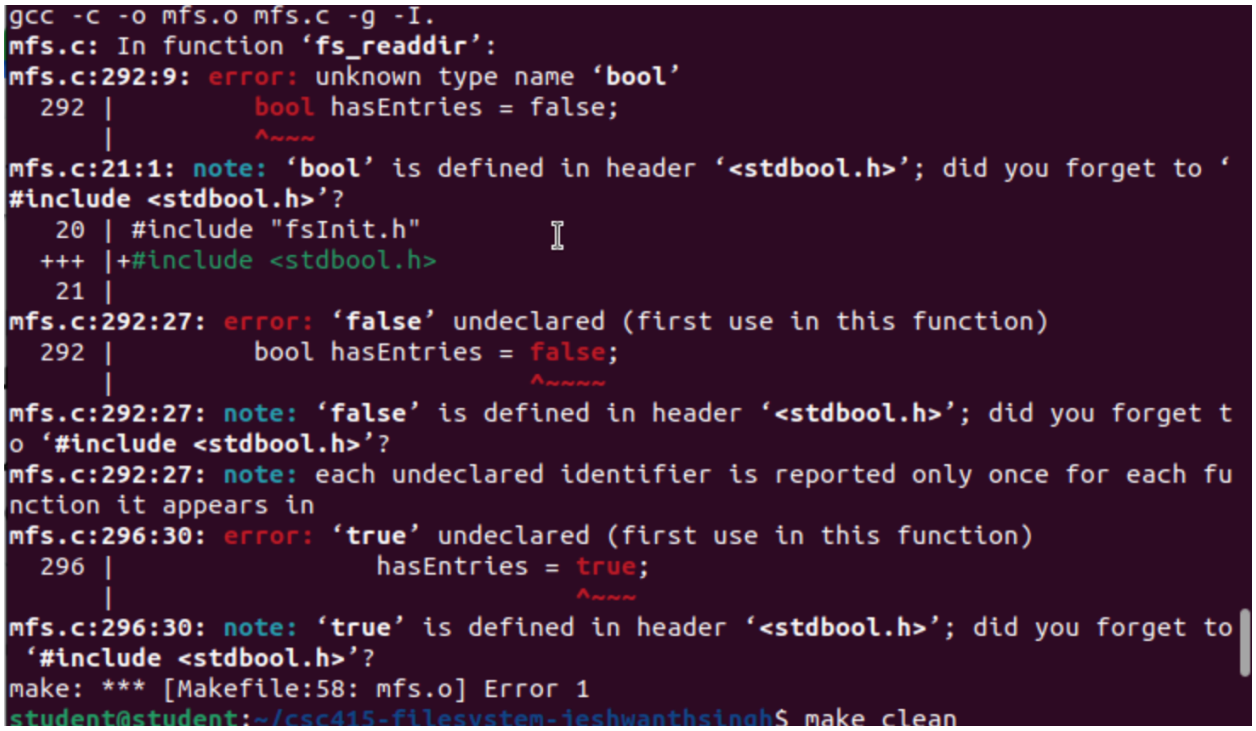
- **Appending Metadata Updates:**

  - Modified the b_write function to handle append operations explicitly by seeking to the end of the file before writing. This ensures that the metadata (size, modification time) is accurately updated.

  - Ensured that the parent directory’s metadata was also updated when file content changes.

- **Streamlining Output:**

• Replaced verbose success messages with simple feedback, like displaying the updated content using cat after operations.



- **Issues:**

1. Unknown Type Name bool:

    - The compiler fails to recognize the bool type in the code, throwing an error about the unknown type name.

    - This error is accompanied by additional errors for false and true, which are also undefined, as these are part of the stdbool.h header file.

2. Missing Inclusion of <stdbool.h>:

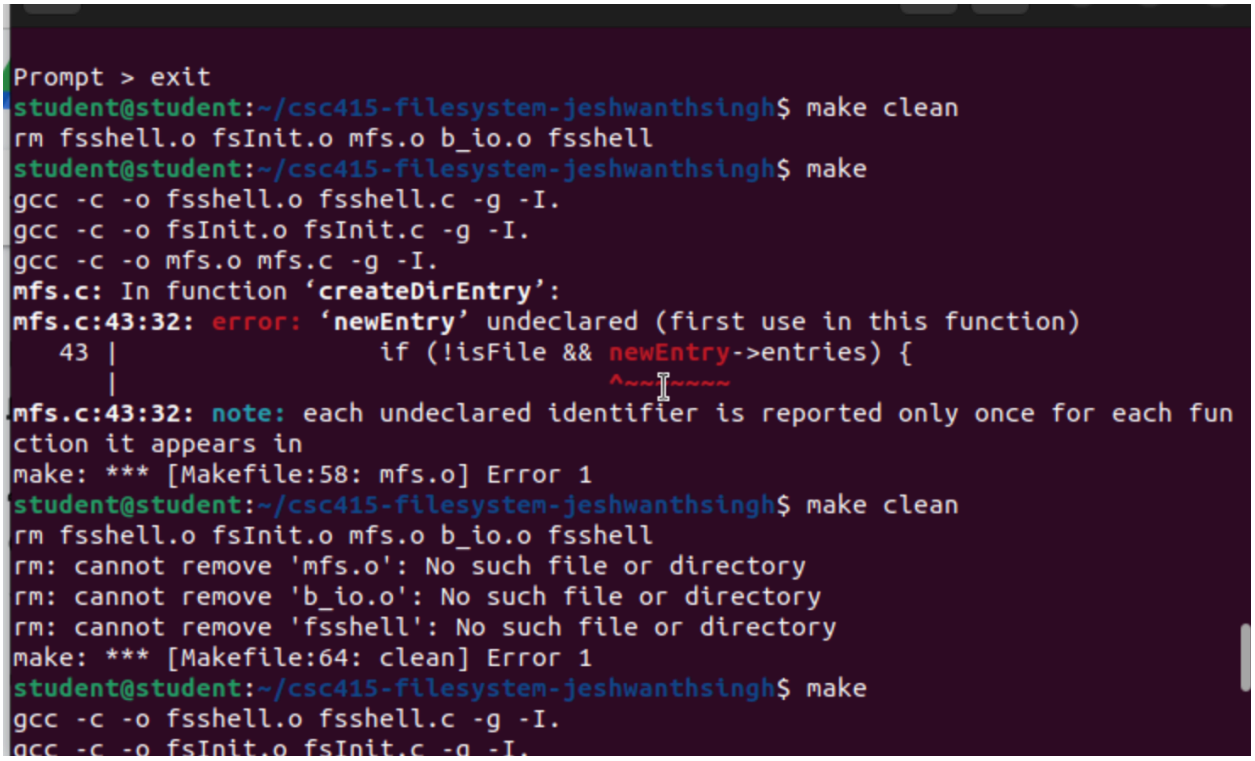
    - The errors indicate that the stdbool.h header file, which defines bool, true, and false in C, is not included in the code.

- **Resolution:**

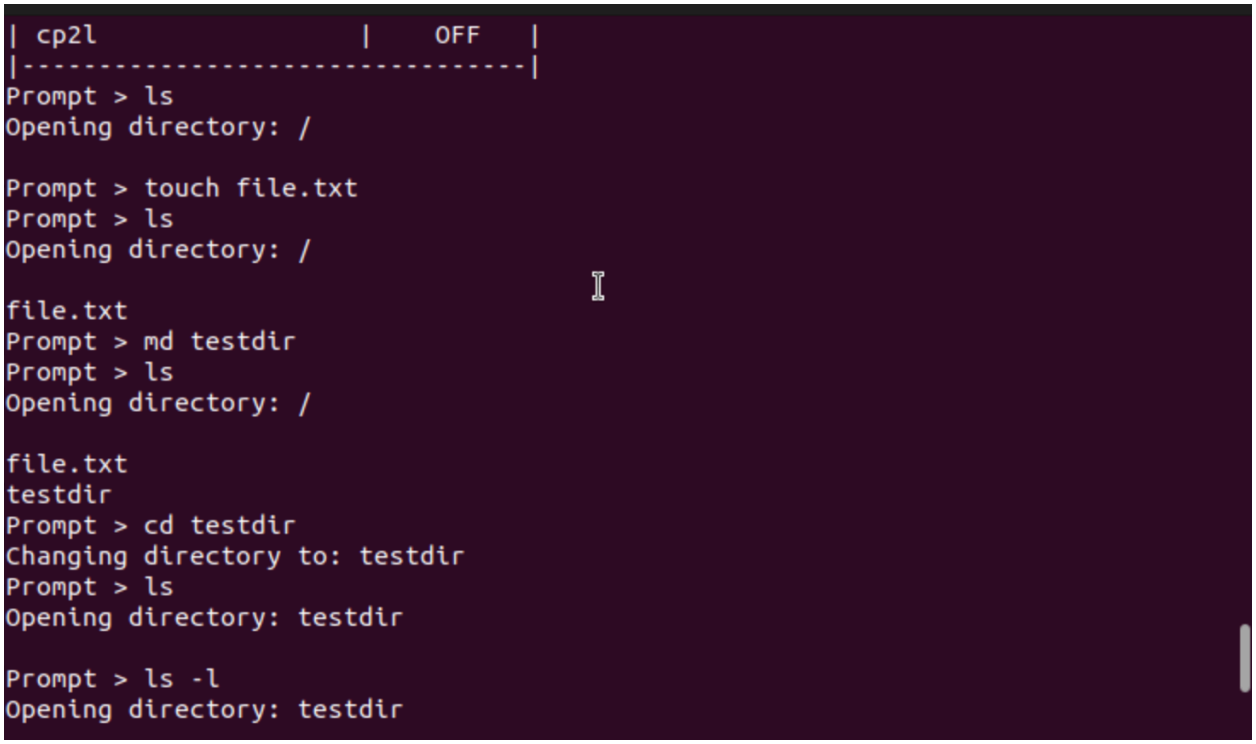
1. Include <stdbool.h>:
    - Added #include <stdbool.h> at the beginning of mfs.c to ensure the bool type and the true/false constants are properly defined.

2. Verified Usage:
    - Confirmed that bool hasEntries is correctly initialized and used in the context of the function fs_readdir.

3. Recompiled Successfully:
    - After including <stdbool.h>, recompiled the code to verify that the errors related to bool, true, and false are resolved.



- **Issue:** newEntry is undeclared in the function createDirEntry.
- **Resolution:**
    - Declare newEntry as directoryEntry *newEntry; at the beginning of the function.
    - Ensure newEntry is properly allocated with malloc and initialized.
    - Check that the function can access all required structures and headers.

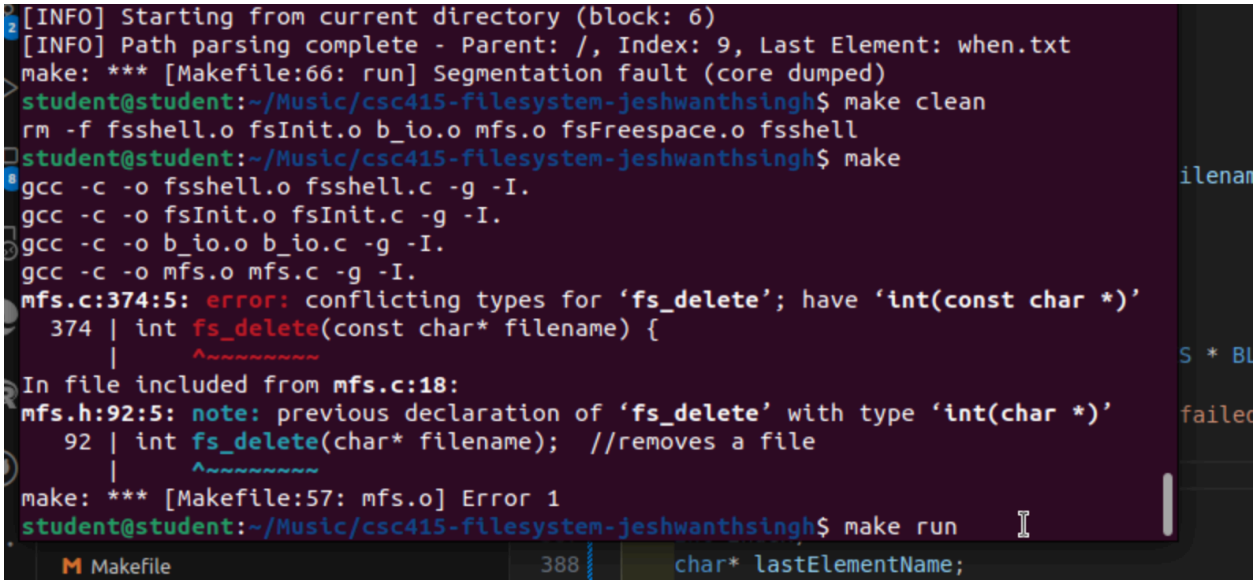


- **Issue:** The directory testdir is being removed despite containing a file (file.txt).
- **Resolution:**

1. Modify fs_rmdir to check if the directory contains files or subdirectories recursively.

2. Add a safeguard in rm to prevent the removal of non-empty directories.

3. Update error messages to indicate when a directory cannot be removed because it is not empty.



- **Issue:** Conflicting types for fs_delete

    - The fs_delete function is declared differently in the header file (int fs_delete(char* filename);) and implemented in the source file (int fs_delete(const char* filename);).

    - This causes a conflicting types error during compilation.

- **Resolution**:

    - Update the declaration and implementation to match. Either add const in the header file or remove it from the source file to ensure consistency.

- **Issue :** Undeclared identifier DIR_BLOCKSBLOCK_SIZE

    - The macro DIR_BLOCKS * BLOCK_SIZE was written without a space, resulting in DIR_BLOCKSBLOCK_SIZE. This typo leads to an undeclared identifier error.

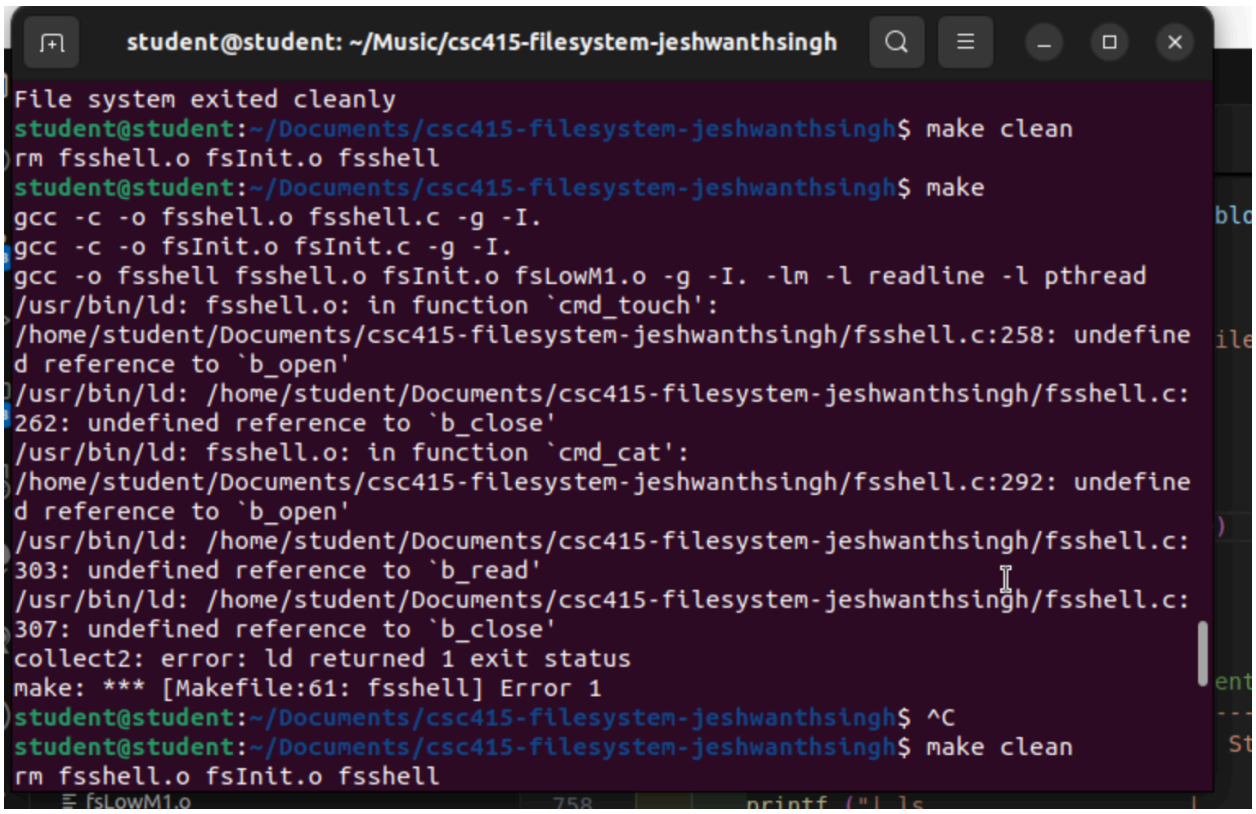
- **Resolution**: Correct the macro usage by adding a space: DIR_BLOCKS * BLOCK_SIZE.

- **Issue :** Argument type mismatch in parsePath

    - The parsePath function expects the fourth argument to be a char**, but a char* is being passed instead, causing a type mismatch warning.

- **Resolution**:

    - Ensure that lastElementName is declared as a char*, and its address (&lastElementName) is passed to the parsePath function.



Issue 1: Undefined references to b_open, b_close, and b_read

• During linking, the compiler fails to resolve references to the b_open, b_close, and b_read functions.

- This is because the object file or source file containing these functions is not included in the Makefile.

Resolution:

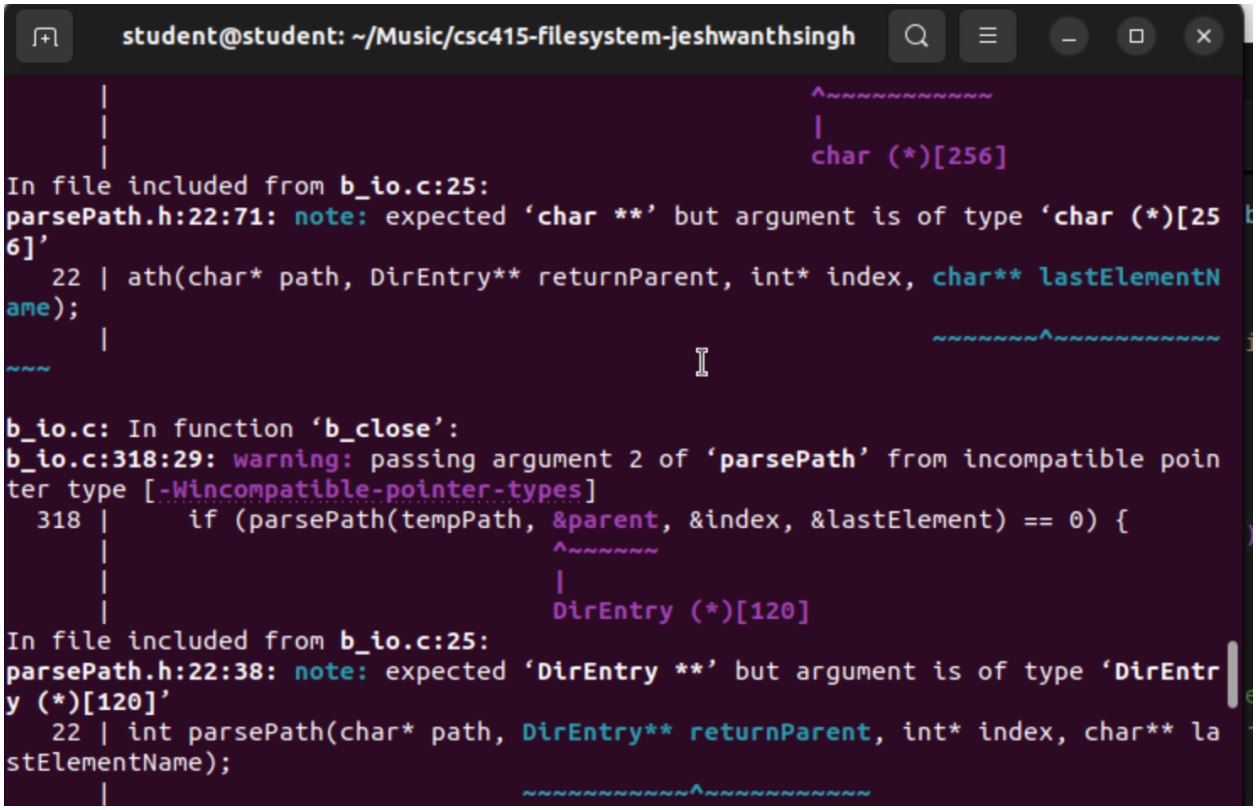
- Identify the file (likely b_io.c) where these functions are implemented.
- Add the corresponding file to the Makefile to ensure it is compiled and linked.

Issue 2: ld returned 1 exit status

- This error occurs as a result of the unresolved references mentioned above.

Resolution:

- Once the missing file is added to the Makefile, the linking process will succeed, resolving this issue.



- **Issue :** Mismatched Pointer Types for parsePath
    - The parsePath function expects the argument lastElementName to be a char** (a pointer to a pointer), but a char[256] (a fixed-size array) is being passed instead.
    - Similarly, the returnParent argument expects a DirEntry**, but DirEntry[120] (a fixed-size array) is being passed instead.
- **Resolution**: Correct Argument Type for lastElementName:
    - Declare lastElementName as char* in the calling function.

    - Pass its address (&lastElementName) to parsePath to ensure compatibility with the char** parameter.

    - Correct Argument Type for returnParent:

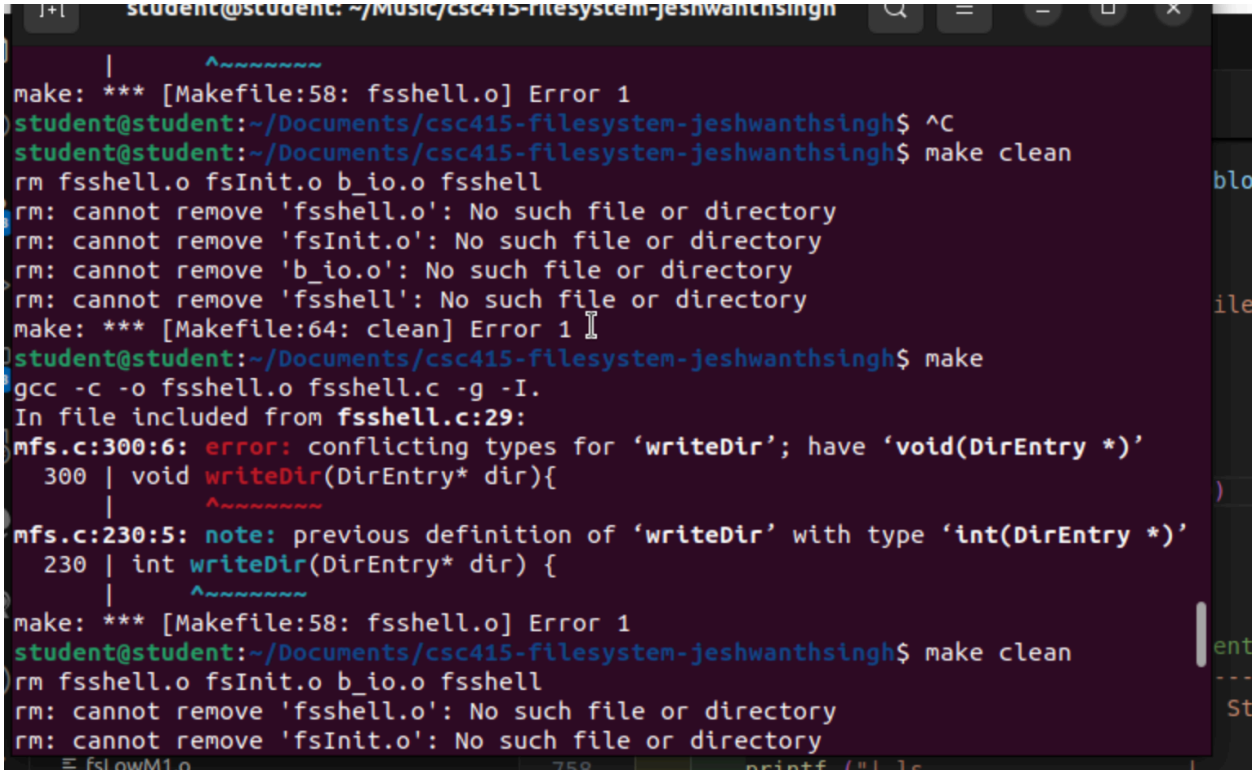
    - Ensure that parent is declared as DirEntry* (a pointer to a DirEntry).

    - Pass its address (&parent) to parsePath to match the DirEntry** parameter.

- **Issue :** Fixed-Size Arrays Passed as Pointers

    - Arrays like char[256] and DirEntry[120] are being passed directly to parsePath, which expects pointers. This creates an incompatible pointer type warning.

- **Resolution**: Convert these arrays to dynamically allocated pointers if needed, or ensure the arguments are passed as pointers (e.g., DirEntry* or char*) by declaring variables appropriately.



Issue : Conflicting Types for writeDir

- The writeDir function has been declared or defined multiple times with conflicting return types:

- At one point, it is defined with a return type of int.

- In another instance, it is defined with a return type of void.

Resolution: Ensure Consistent Return Type

- Decide the appropriate return type for writeDir based on its intended behavior.

- If writeDir needs to indicate success or failure (e.g., return 0 for success and -1 for failure), it should have an int return type.

- If it performs actions without returning a status, use the void return type.

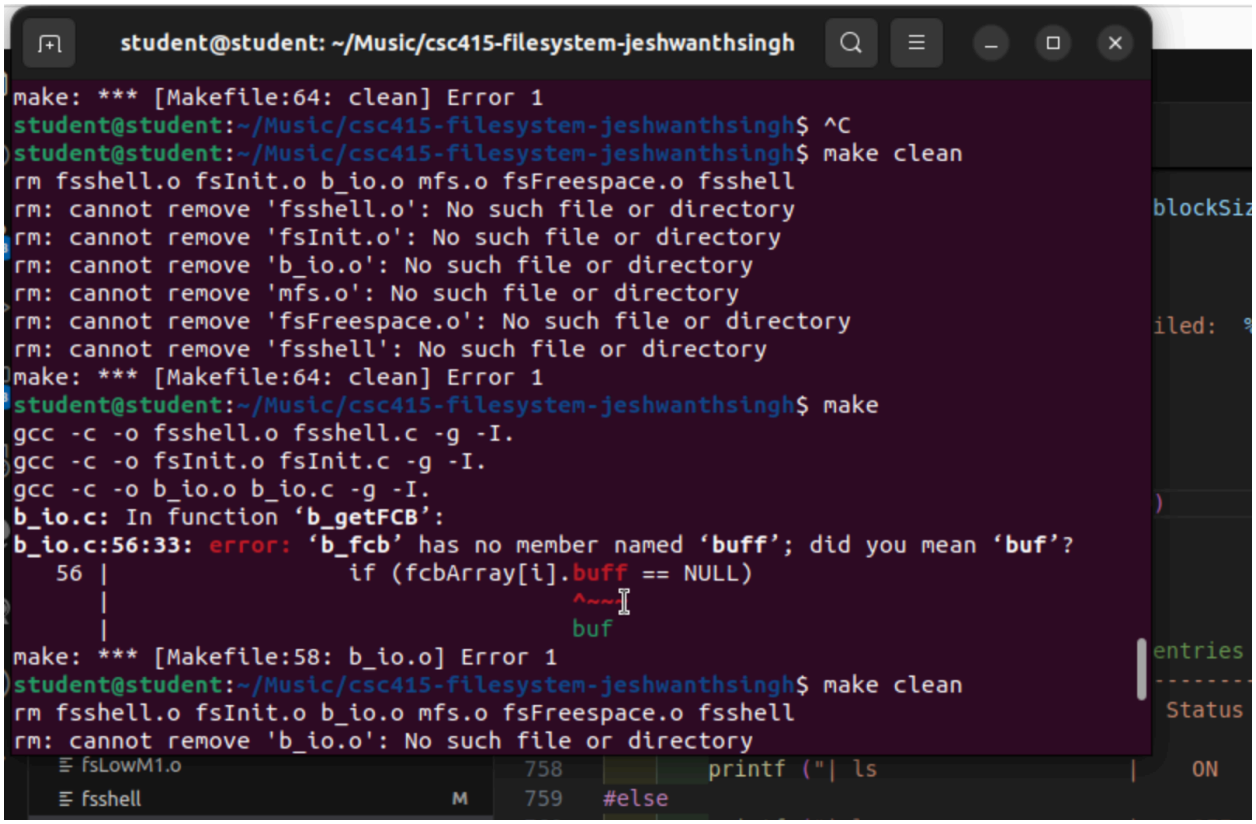
Issue : make clean Errors – Files Not Found

- When running make clean, the rm command attempts to delete object files (fsshell.o, fsInit.o, b_io.o, etc.) and the fsshell binary.

- These files do not exist in the directory, resulting in “No such file or directory” errors.

Resolution: Modify the Makefile

- Use the -f flag with the rm command to prevent errors if the files do not exist.



Issue : Error in b_getFCB – b_fcb Has No Member Named buff

- The compiler reports an error in the b_io.c file at line 56, indicating that the b_fcb structure does not have a member named buff. It suggests that you may have meant buf instead of buff.

Resolution: Verify the b_fcb Structure

- Open the file where the b_fcb structure is defined (likely in b_io.h or a related header file).

- Confirm whether the correct member name is buf or buff.

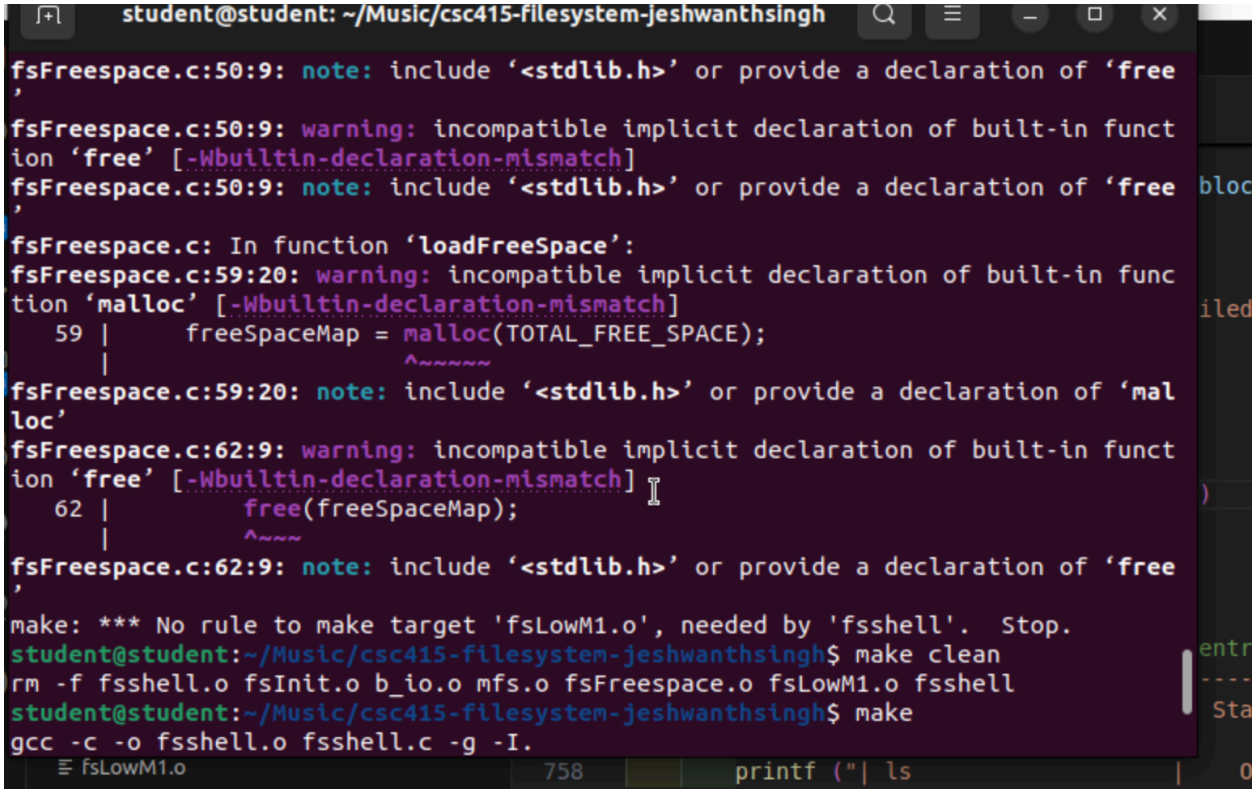
Issue : Errors During make clean – Files Not Found

- When running make clean, the rm command attempts to remove object files (fsshell.o, fsInit.o, b_io.o, etc.) and the binary fsshell.

- These files do not exist, resulting in “No such file or directory” errors.

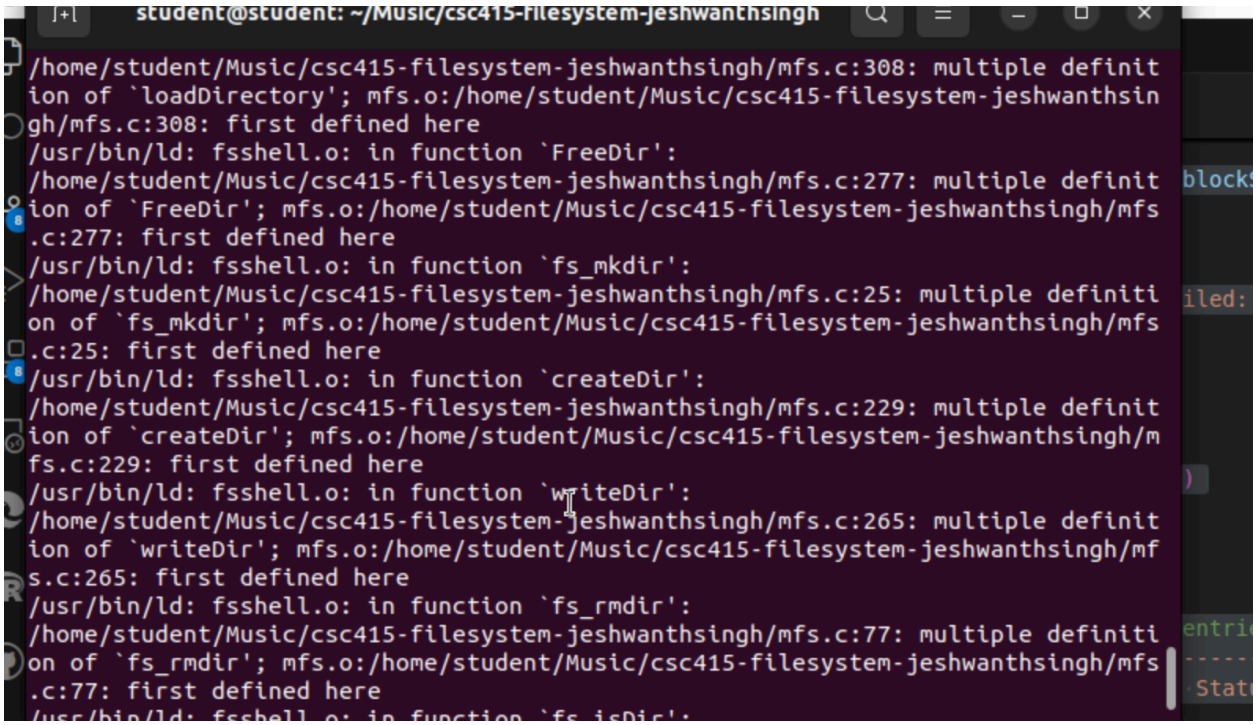
Resolution: Use -f Option with rm

- Modify the Makefile to use the -f flag in the rm command. This ensures no errors are raised when files are missing.



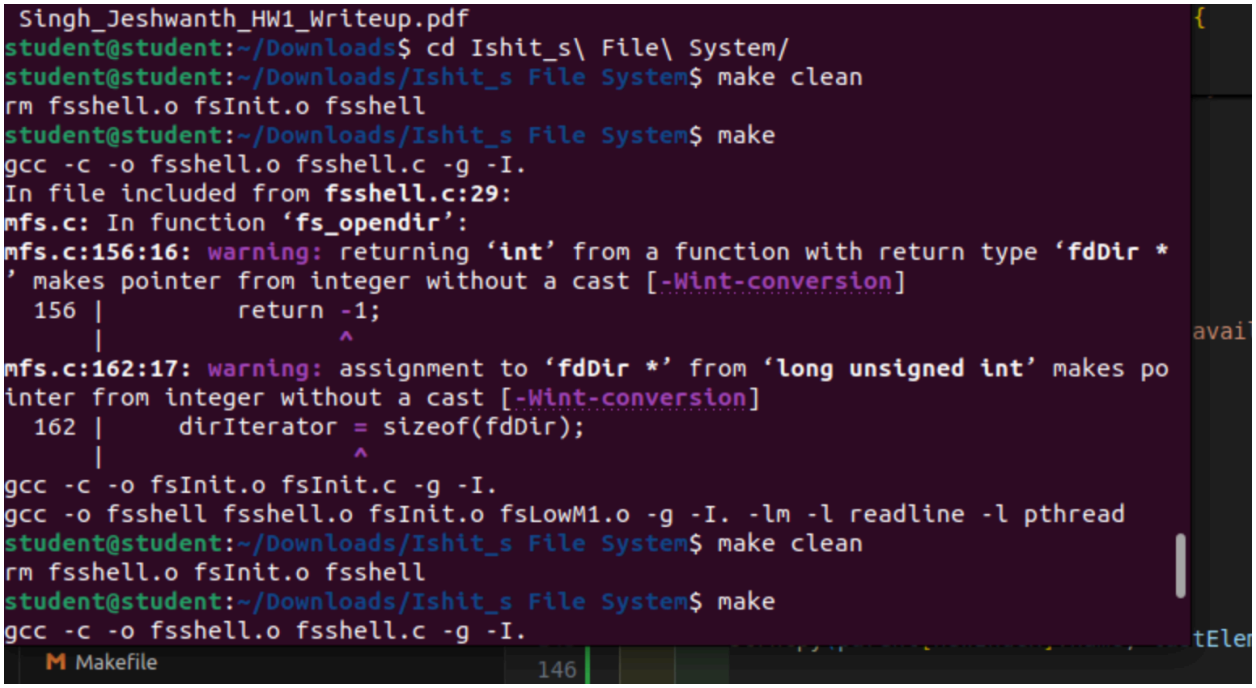
- **Issue: Missing Header File for Memory Functions**
  - When compiling fsFreespace.c, warnings appear about incompatible implicit declarations for standard functions like malloc and free. The warnings also suggest including <stdlib.h> to resolve these issues.

- **Resolution: Include <stdlib.h> in fsFreespace.c**
    - Add the line #include <stdlib.h> at the top of fsFreespace.c to ensure the standard library declarations for malloc and free are available.



- **Issue:** The compiler reports “multiple definition” errors for functions like loadDirectory, FreeDir, fs_mkdir, createDir, writeDir, and fs_rmdir. This indicates that these functions are defined multiple times in different source files or included inappropriately.

- **Resolution:** Ensure that each function is defined only once in a source file. If the functions are defined in a header file or included multiple times, use header guards (#ifndef, #define, #endif) or the static keyword to limit scope. Additionally, avoid directly including .c files like parsePath.c in other files; instead, include their corresponding header files.



- **Issue:** Warning in fs_opendir function

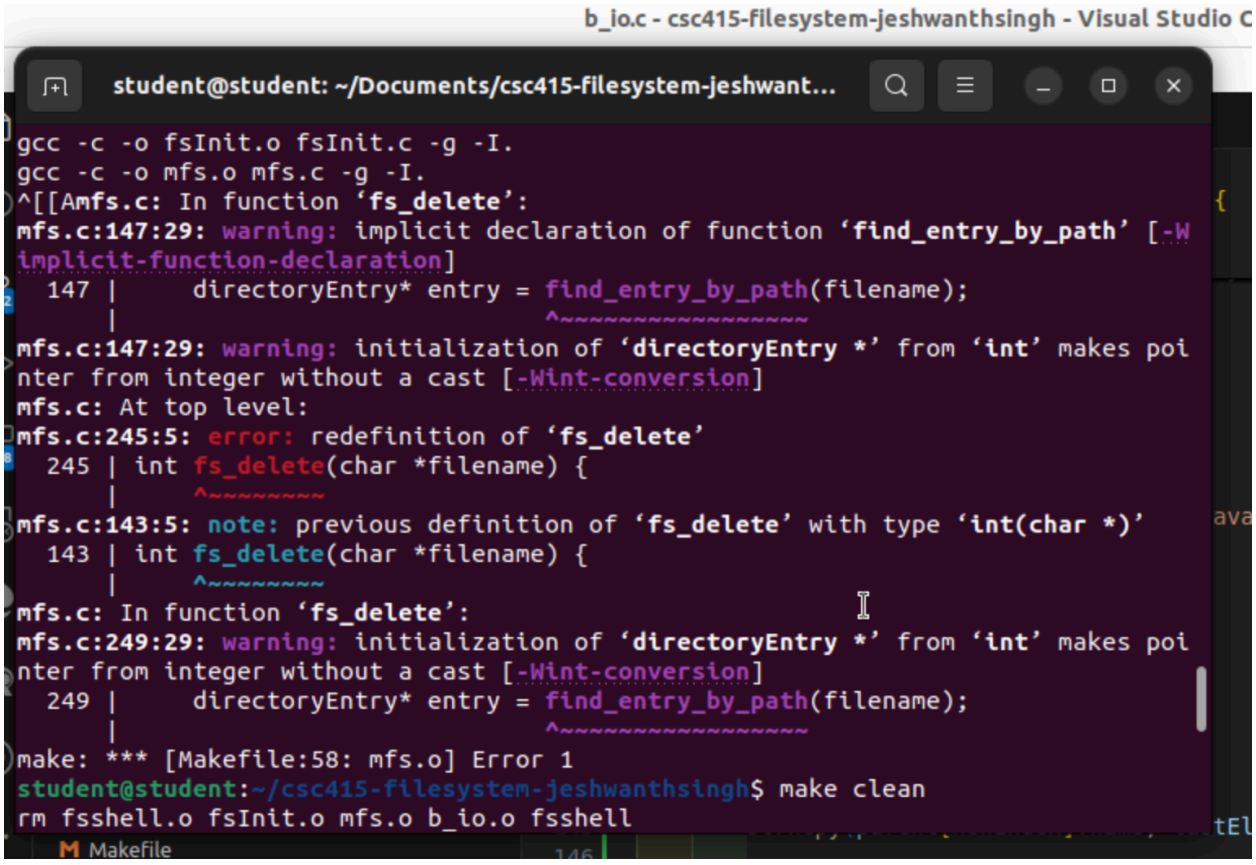
    - The warning occurs because the function fs_opendir is defined with a return type of fdDir* (a pointer type), but an integer -1 is being returned. This results in the compiler issuing a type mismatch warning as returning an integer is incompatible with the expected pointer type.

- **Resolution :** To fix this issue, ensure that the return type aligns with the function's declaration. Replace return -1; with return NULL;, as NULL is the standard representation for a null pointer, which is compatible with the fdDir* type.

- **Issue: Warning during assignment:**

    - The warning occurs on the line dirIterator = sizeof(fdDir);. This is because sizeof(fdDir) returns a size_t (an unsigned integer type), but the variable dirIterator is of type fdDir* (a pointer type). Assigning an integer to a pointer without proper casting causes this issue.

- **Resolution :** Change the line dirIterator = sizeof(fdDir); to properly allocate memory for fdDir using malloc.



10-11 Testing & Screenshots.

1) Compilation:

```
student@student:~/csc415-filesystem-jeshwanthsingh$ make clean
rm -f fsshell.o fsInit.o b_io.o mfs.o fsFreespace.o fsshell
student@student:~/csc415-filesystem-jeshwanthsingh$ make
gcc -c -o fsshell.o fsshell.c -g -I.
gcc -c -o fsInit.o fsInit.c -g -I.
gcc -c -o b_io.o b_io.c -g -I.
gcc -c -o mfs.o mfs.c -g -I.
gcc -c -o fsFreespace.o fsFreespace.c -g -I.
gcc -o fsshell fsshell.o fsInit.o b_io.o mfs.o fsFreespace.o fsLow.o -g -I. -lm -l readline -l pthread
student@student:~/csc415-filesystem-jeshwanthsingh$
```

2) Initialization:

```
student@student:~/csc415-filesystem-jeshwanthsingh$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does exist, errno = 0
File SampleVolume good to go, errno = 0
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Initializing File System with 19531 blocks and block size of 512 bytes
The size of Directory is 60 bytes
Signature match found, Volume already initialized
Freespace loaded successfully
The name of the root directory is .
|---------------------------------|
|------- Command ------|- Status -|
| ls                   |    ON    |
| cd                   |    ON    |
| md                   |    ON    |
| pwd                  |    ON    |
| touch                |    ON    |
| cat                  |    ON    |
| rm                   |    ON    |
| cp                   |    ON    |
| mv                   |    ON    |
| cp2fs                |    ON    |
| cp2l                 |    ON    |
|---------------------------------|
Prompt >
```

3) “ls” command:

```
Prompt > ls

name
name2
f7.txt
f6.txt
Prompt > cd name
Prompt > ls

name2
f4.txt
f5.txt
f1.txt
Prompt > cd name2
Prompt > ls

Prompt >
```

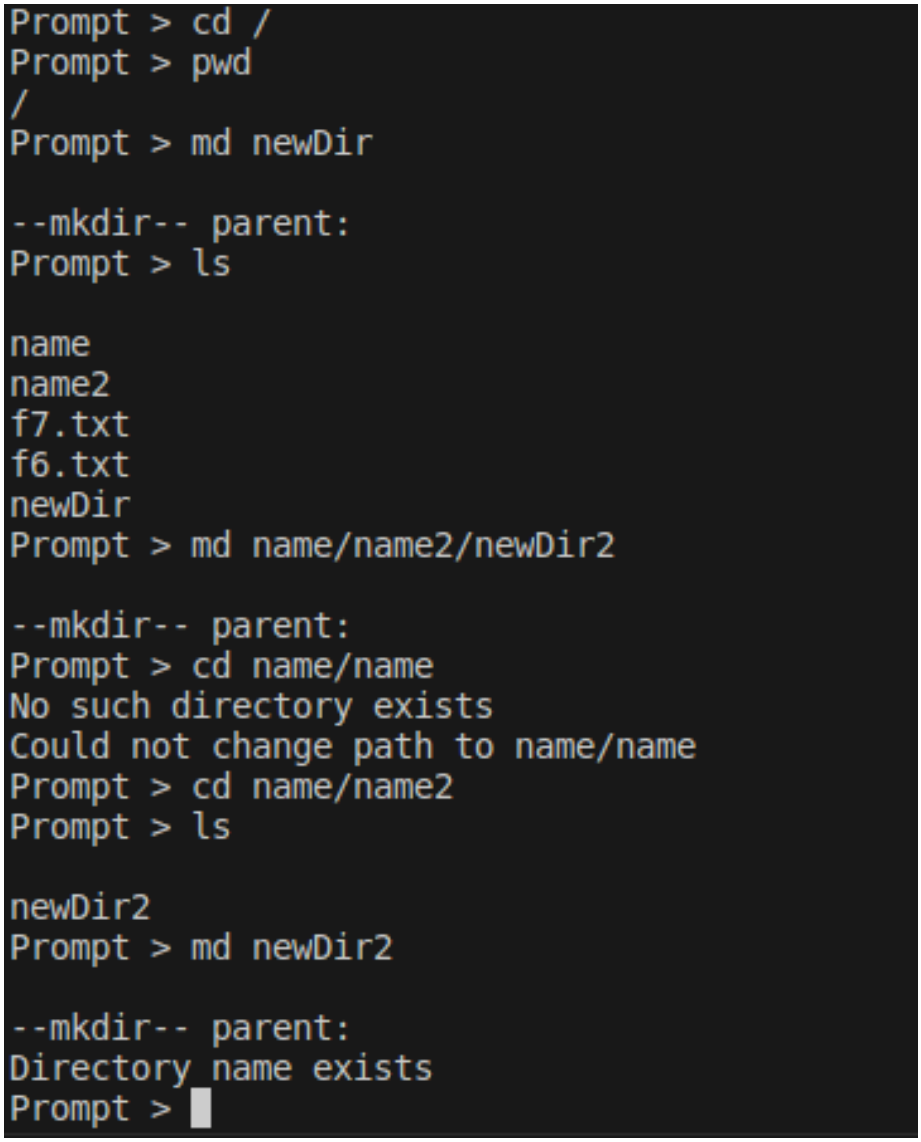
```
Prompt > cd ..
Prompt > cd ..
Prompt > ls

name
name2
f7.txt
f6.txt
Prompt > ls -l

D          3060   name
D          3060   name2
-           512   f7.txt
-             0   f6.txt
Prompt >
```

```
Prompt > ls -l

D          3060   name
D          3060   name2
-           512   f7.txt
-             0   f6.txt
Prompt > ls -a

name
name2
f7.txt
f6.txt
Prompt >
```

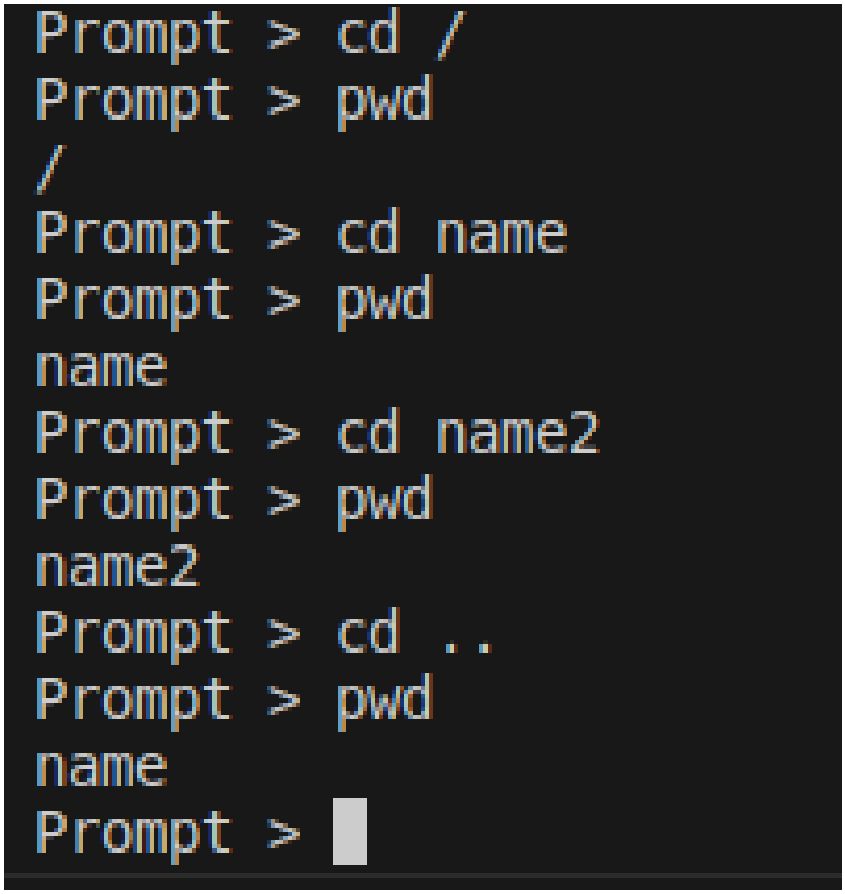
4) **“cd”:change directory command:**

```
Prompt > cd name
Prompt > cd ..
Prompt > pwd
/
Prompt > cd name
Prompt > pwd
name
Prompt > cd .
Prompt > pwd
name
Prompt > cd ..
Prompt > pwd
/
Prompt > cd name/name2
Prompt > pwd
name2
Prompt > cd name3
No such directory exists
Could not change path to name3
Prompt > cd ..
Prompt > ls

name2
f4.txt
f5.txt
f1.txt
Prompt > cd f4.txt
No such directory exists
Could not change path to f4.txt
Prompt >
```

5) “md”: make directory command:



7) “pwd”: present working directory



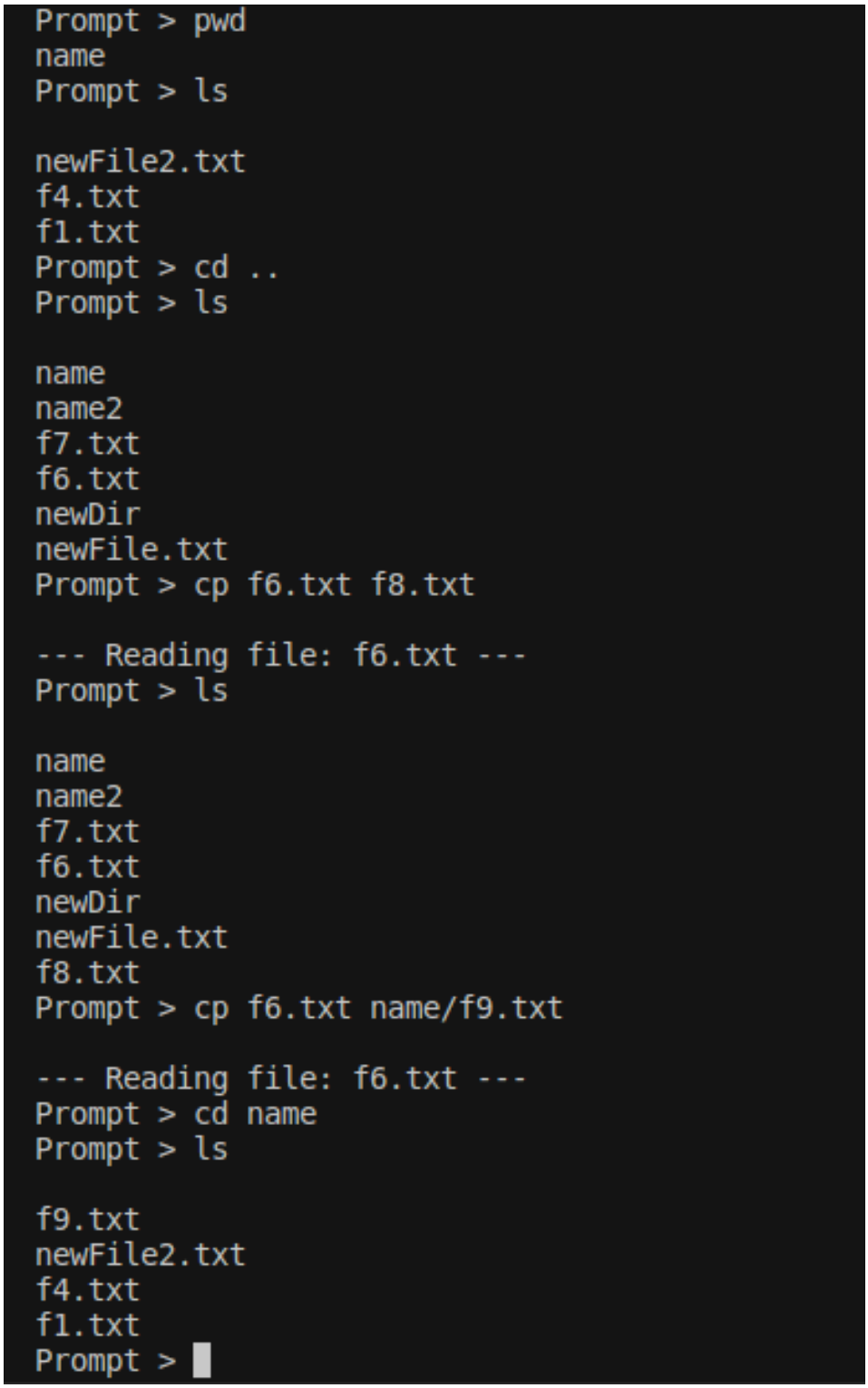
8) “touch” command:

```
Prompt > cd /
Prompt > ls

name
name2
f7.txt
f6.txt
newDir
Prompt > touch newFile.txt
Prompt > ls

name
name2
f7.txt
f6.txt
newDir
newFile.txt
Prompt > touch newFile.txt
Prompt > ls

name
name2
f7.txt
f6.txt
newDir
newFile.txt
Prompt > touch name/newFile2.txt
Prompt > ls

name
name2
f7.txt
f6.txt
newDir
newFile.txt
Prompt > cd name
Prompt > ls

name2
newFile2.txt
f4.txt
f5.txt
f1.txt
Prompt >
```

9) “rm” command:

```
Prompt > cd name
Prompt > ls

name2
newFile2.txt
f4.txt
f5.txt
f1.txt
Prompt > rm f5.txt
Prompt > ls

name2
newFile2.txt
f4.txt
f1.txt
Prompt > rm name2
--rmdir--
Directory is not emptyPrompt > cd name2
Prompt > ls

newDir2
Prompt > rm newDir2
--rmdir--
Prompt > ls

Prompt > cd ..
Prompt > ls

name2
newFile2.txt
f4.txt
f1.txt
Prompt > rm name2
--rmdir--
Prompt > ls

newFile2.txt
f4.txt
f1.txt
Prompt >
```

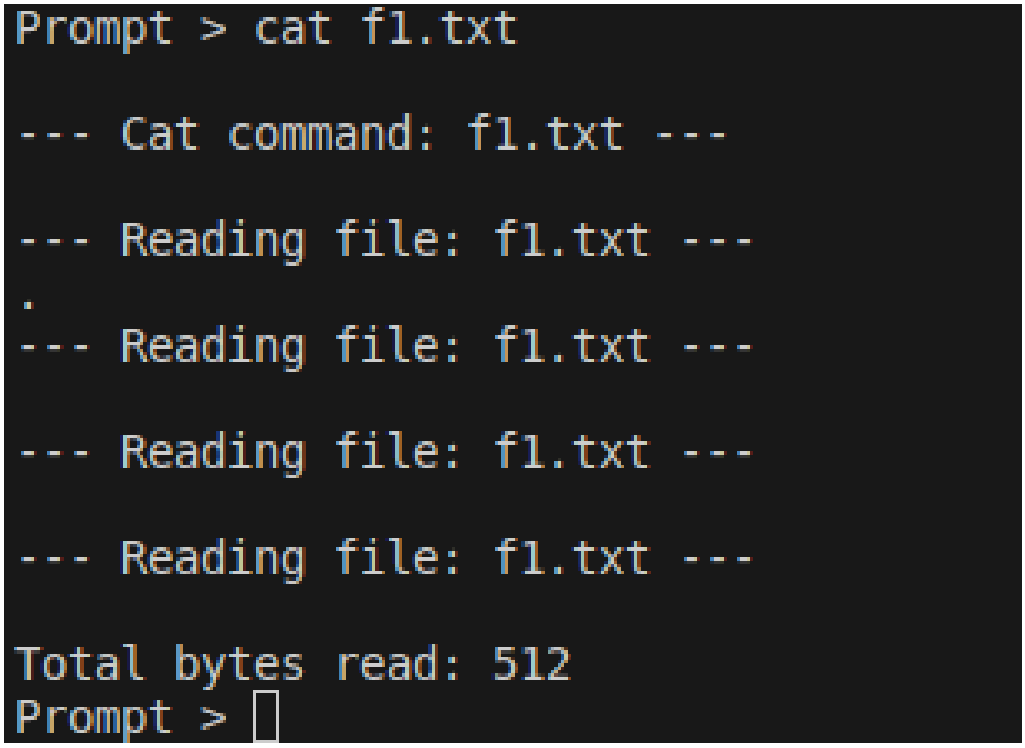
10) “cp” command:



11) “mv” move file command:

```
Prompt > cd name
Prompt > ls

f9.txt
newFile2.txt
f4.txt
f1.txt
Prompt > mv f9.txt ../f10.txt

--- Reading file: f9.txt ---

--- Reading file: f9.txt ---

--- Reading file: f9.txt ---
Prompt > ls

newFile2.txt
f4.txt
f1.txt
Prompt > cd ..
Prompt > ls

name
name2
f7.txt
f6.txt
newDir
newFile.txt
f8.txt
f10.txt
Prompt >
```

12) “cat” command:



Plans for each phase and changes made:

Phase 1: Formatting the volume

The initial phase focused on setting up the foundation for the file system. The plan was to:

1. Initialize the VCB in block 0, defining critical metadata such as total blocks, block size, and pointers to the free space bitmap and root directory.
2. Implement the free space management system using a bitmap stored in Block 1 where each bit represents the allocation status of a block
3. Create and initialize the root directory in Block 6 with self-referential entries (. and ..) to establish the starting point for the file system.

Changes Made:

During implementation we realized that additional verification steps were necessary to make sure the integrity of the VCB and bitmap after writing them to disk. We added validation logic to confirm successful writes and proper initialization of reserved blocks.

Phase 2: Implementing Directory Based Functions:

The second phase focused on directory management, including creation, deletion and traversal. The plan was to:

1. Implement basic directory operations fs_mkdir and fsrmdir to allow creation and removal of directories
2. Develop traversal functions fs_opendir, fs_readdir, and fs_closedir to enable listing and accessing directory entries
3. Build path-parsing logic to handle absolute and relative paths, ensuring directories could be accessed and manipulated correctly

Phase 3: Implementing File Operations

The final phase focused on file creation, reading, writing and deletion.

13. Conclusion

This project provided a practical and challenging opportunity to design and implement a custom file system from the ground up. By combining low-level block management with high-level file and directory operations, we gained a solid understanding of how file systems are structured and how they function. Implementing LBA for efficient data storage and retrieval, along with features like the VCB and bitmap based free space management, helped us appreciate the importance of well organized system metadata and efficient resource tracking.

As with any project of this scope, we encountered challenges along the way. Issues with segmentation faults, directory traversal errors, and bitmap allocation bugs pushed us to think critically and debug carefully.. These moments were not just about solving

problems but about understanding the underlying systems better. Collaborating as a team allowed us to break the work into manageable pieces and stay on schedule, while tools like Github kept the process smooth and well coordinated.

There’s plenty of potential for improvement. Optimizing block allocation to reduce fragmentation and introducing caching could make the system more efficient. Expanding metadata to include more details like file permissions and ownership would add more depth and flexibility. Additional features like support for symbolic links, journaling for crash recovery, and better handling of larger volumes would make the system more robust and practical for real world use.

While the interactive shell works well for basic commands, there’s room to enhance its functionality with additional commands or even a graphical user interface to make it more intuitive. Moving forward, thorough testing, including stress tests and validation of edge cases would help ensure the system is both reliable and resilient. Overall this project laid a strong foundation and it’s exciting to think about how it could grow and evolve with further development.