CSC415 FILE SYSTEM ASSIGNMENT

TEAM	**0xAACD**
GITHUB	cherylfong
REPOSITORY URL	https://github.com/CSC415-2024-Fall/csc415-filesystem-cherylfong

MEMBER NAME	STUDENT ID	GITHUB USERNAME
Cheryl Fong	918157791	cherylfong
Arvin Ghanizadeh	922810925	arvinghanizadeh
Atharva Walawalkar	924254653	AtharvaWal2002
Danish Nguyen	923091933	dlikecoding

Table Of Contents

INTRODUCTION

Our team, 0xAACD, was tasked with this file system assignment to design and implement a functioning file system. The project is structured in three phases, where each progressively builds upon the previous to achieve a functional file system. As a team, we explored core concepts, including volume formatting, directory management, and file operations, while following the assignment's specifications and constraints.

The first phase focused on volume formatting, a necessary step to create the file system's structural framework. In this phase, we initialized the volume, defined a free space management system, and prepared for subsequent directory and file handling.

In the second phase, we delved into directory-based functions. We implemented methods to create, navigate, and manage directories. This included initializing a root directory and ensuring proper maintenance of directory metadata. Our directory handling allowed our system to mimic Linux-based functionalities.

The final phase involved file operations. We consider this the most intricate part of the assignment. We designed and implemented functions for creating, reading, writing, deleting, and managing files to ensure consistency and efficiency. Special attention was given to interfacing with low-level Logical Block Address (LBA) operations which were provided as part of the assignment in the fsLow.o library, i.e., using the LBAread and LBAwrite functions.

The LBA read/write operations enabled direct volume interaction. Additionally, utilities such as a hex dump tool allowed for analysis of the file system's internal state.

To enable basic functionality, we completed the shell driver (fsShell.c) to interact with the file system. This shell includes built-in commands for the user to manipulate directories and files.

We have met the objectives of this assignment to create a file system capable of managing free space, maintaining directories, and supporting file operations. Our implementation ensures functional interaction between the user and the file system. Furthermore, this assignment allowed us to systematically address challenges as a team to produce a functional file system.

Note that the term "disk" is used interchangeably to mean the virtual volume of our filesystem.

0xAACD’s File System

This section of the document describes the high-level overview of 0xAACD’s file system, i.e., in terms of functional components. This section concludes with an example usage of 0xAACD’s file system.

For a more thorough analysis of the individual components of our file system, please refer to the section titled “Project Progression.” Non-functional capabilities of the file system, i.e., characteristics of how our file system operates, are described in the “Design” section of this document.

Functional Components

The functional capabilities of our file system are comprised of volume and directory management, file operations, and the shell that interfaces between the user and our file system.

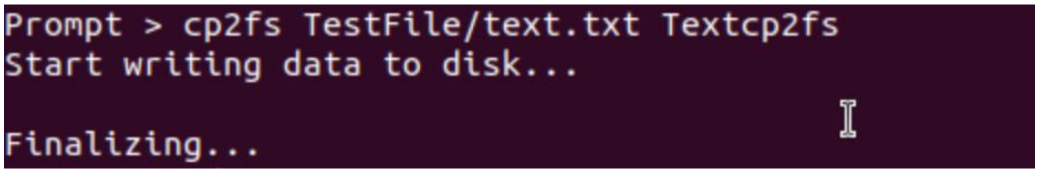
Volume management is the critical component that forms the basis of our entire file system. The Volume Control Block (VCB) is initialized on the virtual volume to track metadata that includes the amount of free space, the location of the root directory, and is responsible for volume configuration, such as the volume’s signature identification. The VCB communicates with the free space management component that is responsible for the allocation and release of blocks using extent-based mapping.

Directory management of our file system builds upon the functionality of our volume management design, supporting the transfer of data from virtual volume into memory. Each directory on our file system has a saved and tracked entry or record, colloquially known as a Directory Entry. This recording of directories enables operations related to directory structure and tracking of its metadata. Our file system permits operations like creation, deletion, and traversal of directories. Additionally, our file system consistently tracks the root and current working directories, allowing loading and unloading when needed.

With both working volume and directory management, this basis permits the utility of file operations, which includes the opening, reading, writing, seeking, and closing of files. On the block level, file manipulation and buffering use file descriptors and logical block reading and writing.

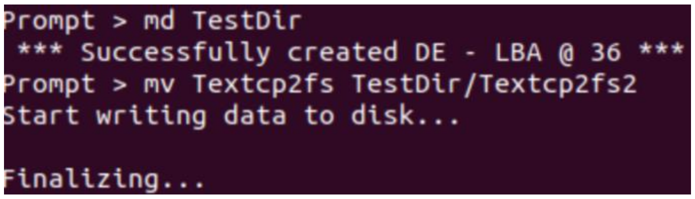
All previously mentioned components are abstracted from the user in the form of shell functions. The user is able to interact with our file system via simple shell commands provided by the shell driver. This simple shell allows the user to list the contents of directories and view the directory and file metadata, make new files and directories, and inspect the contents of a file. Commands are similar to Linux-based shell commands such as “ls” and “cat,” with some exceptions, such as “md” to make a new directory instead of “mkdir”.

Usage examples of 0xAACD's File System



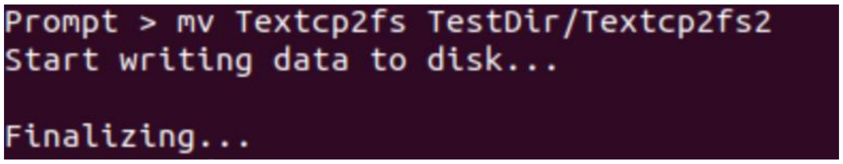
```
Prompt > cp2fs TestFile/text.txt Textcp2fs
Start writing data to disk...

Finalizing...
```

Copying file from the Ubuntu Virtual Machine to our filesystem.



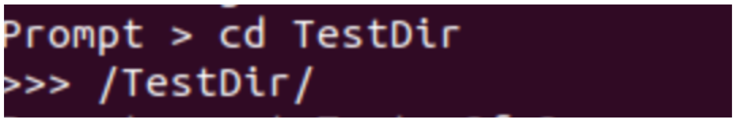
```
Prompt > md TestDir
 *** Successfully created DE - LBA @ 36 ***
Prompt > mv Textcp2fs TestDir/Textcp2fs2
Start writing data to disk...

Finalizing...
```

Creating a directory named TestDir in our filesystem.



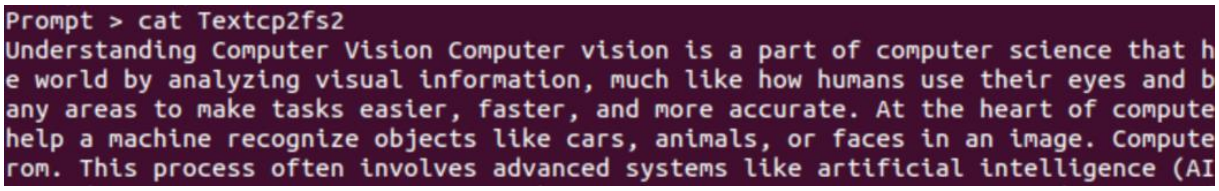
```
Prompt > mv Textcp2fs TestDir/Textcp2fs2
Start writing data to disk...

Finalizing...
```

Moving files to the relative destination path.



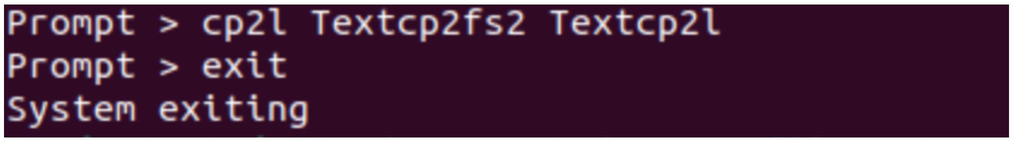
```
Prompt > cd TestDir
>>> /TestDir/
```

Changing the current working directory.



```
Prompt > cat Textcp2fs2
Understanding Computer Vision Computer vision is a part of computer science that h
e world by analyzing visual information, much like how humans use their eyes and b
any areas to make tasks easier, faster, and more accurate. At the heart of compute
help a machine recognize objects like cars, animals, or faces in an image. Compute
rom. This process often involves advanced systems like artificial intelligence (AI
```

Printing the contents of a file.



```
Prompt > cp2l Textcp2fs2 Textcp2l
Prompt > exit
System exiting
```

Files from our file system is able to be copied into the native file system of the Ubuntu Virtual Machine.

DESIGN

This section explains the characteristics of our file system and the design approach we have taken as a team.

The initialization of VCB sets up the file system by loading metadata or creating a new structure if none exists. It allocates memory for the VCB, reads it from the virtual volume, and checks for a valid signature to determine whether to load an existing file system or format a new one. The process initializes critical components like the free space map, root directory, and current working directory, ensuring they are ready for file operations.

Our file system generates a signature identification for the volume to indicate proper formatting to work with our implementation. If the file system signature matches the predefined value, the existing free space map and directory structure are loaded from the virtual volume.

There is also centralized initialization of our file system and cleanup routines for a consistent system state. If the signature doesn't match, the VCB is set up with default values, a new free space map is created, and the root directory is initialized. Diagnostic details are logged for verification. On shutdown, the VCB and free space map are written back to the virtual volume, and memory allocations are freed to ensure system integrity.

Our free space management design is scalable as it can dynamically handle primary, secondary, and tertiary extent tables. Furthermore, these extents are able to be merged for optimal free space usage. Together, these components allow free space maps and metadata to be saved to the virtual volume, which maintains the state between sessions – provided that the volume is not reformatted.

In greater detail, the free space management system is designed to maintain a structured and efficient free space map for a filesystem. It manages available volume blocks through the "freespace_st" structure, which contains metadata such as total block free to track the total number of free blocks on the volume, reserved block to record the number of blocks set aside for managing extents, and extentLength to indicate the length of the primary extent in the free space. etc.. The "freespace_st" structure is stored in the same LBA as the VCB for consistent access. Free Space system uses a hierarchy of extent tables (primary, secondary, and tertiary) to handle increasing storage demands while balancing memory usage and system scalability.

When the "initFreeSpace" function sets up the free space map during the filesystem's initial configuration. It calculates the number of reserved blocks based on the virtual volume size and fragmentation estimates, initializes the extent structures, and records their locations. Additionally, the function configures the primary extent table and marks the starting point for allocating free blocks.

"loadFreeSpaceMap" function handles the loading and maintenance of the free space map in memory. It avoids unnecessary disk reads by reusing a valid map already present in memory. When blocks are requested, the "allocateBlocks" function assigns them from the free space map while respecting constraints, such as the minimum number of contiguous blocks. The function modifies the map by updating or removing extents corresponding to the allocated blocks.

Block release is managed by the "releaseBlocks" function, which reclaims blocks by merging them with existing extents or creating new extents if required. It updates the free space map to reflect the returned blocks while addressing any overlaps to maintain consistency. Supporting functions, such as "addExtent" and "removeExtent", manage the addition and removal of extents within the map.

To handle larger datasets, the system dynamically integrates secondary and tertiary extent tables. These additional tables are created when the primary extent table reaches capacity. The tertiary table acts as a map store LBA of secondary tables. Some helper functions like "createSecondaryExtentTB" and "createTertiaryExtentTB" are used to create and link these tables, maintaining the structured approach to free space management.

The structure of our directory entry records enables the hierarchical organization of files and directories during traversal. Persistent maintenance of the current working directory and root directory metadata enables this traversal.

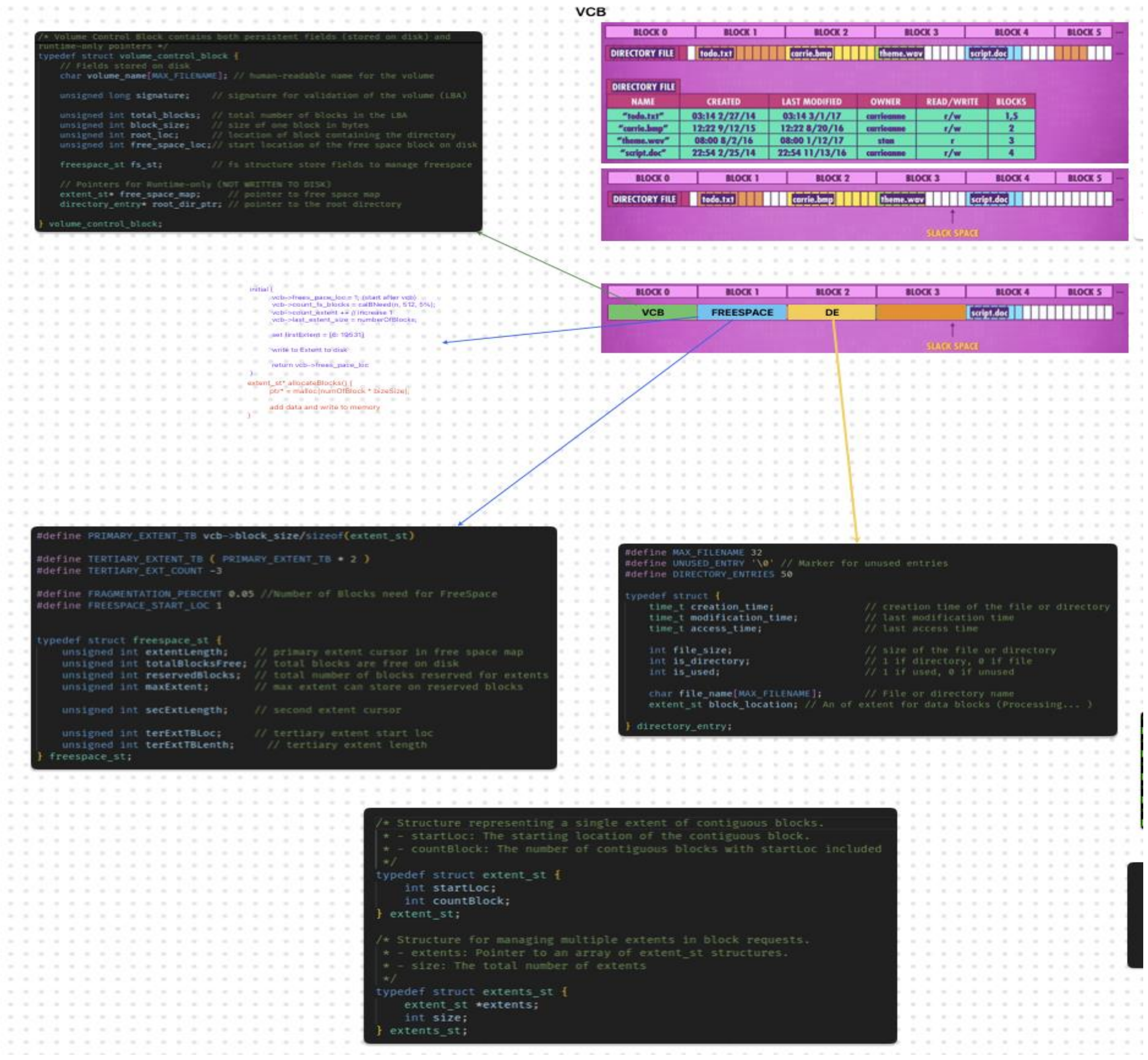
Our Design Approach

In brief, our code is modular to allow the members of our team to easily debug and develop our code base as we progress through the milestones of this project. By separating concerns across dedicated files such as fsInit.c, b_io.c, and FreeSpace.c, we created a system that facilitates debugging and integration of components through clearly defined function signatures.

We have used standardized file headers and well-documented code comments to enhance clarity. The intent is to provide a consistent structure that supports collaboration and ease of readability.

The robustness of our file system is achieved through basic error checking in critical operations like block allocation, file I/O, and directory management, along with mechanisms to protect against fragmentation by merging extents and efficiently reusing released blocks. Finally, the system is portable. It is designed to run on an Ubuntu Linux Operating system using C. Files from our virtual volume can be transferred into Ubuntu's native file system and vice versa while maintaining the file's integrity.

The screenshots below are used to illustrate our design process and considerations.



DIRECTORY FILE

NAME	CREATED	LAST MODIFIED	OWNER	READ/WRITE	BLOCKS
"todo.txt"	03:14 2/27/14	03:14 3/1/17	carrieanne	r/w	1,5
"carrie.bmp"	12:22 9/12/15	12:22 8/20/16	carrieanne	r/w	2
"theme.wav"	08:00 8/2/16	08:00 1/12/17	stan	r	3
"script.doc"	22:54 2/25/14	22:54 11/13/16	carrieanne	r/w	4

```c
/* Volume Control Block contains both persistent fields (stored on disk) and
runtime-only pointers */
typedef struct volume_control_block {
    // Fields stored on disk
    char volume_name[MAX_FILENAME]; // human-readable name for the volume

    unsigned long signature;    // signature for validation of the volume (LBA)

    unsigned int total_blocks;  // total number of blocks in the LBA
    unsigned int block_size;    // size of one block in bytes
    unsigned int root_loc;      // location of block containing the directory
    unsigned int free_space_loc;// start location of the free space block on disk

    freespace_st fs_st;         // fs structure store fields to manage freespace

    // Pointers for Runtime-only (NOT WRITTEN TO DISK)
    extent_st* free_space_map;     // pointer to free space map
    directory_entry* root_dir_ptr; // pointer to the root directory

} volume_control_block;
```

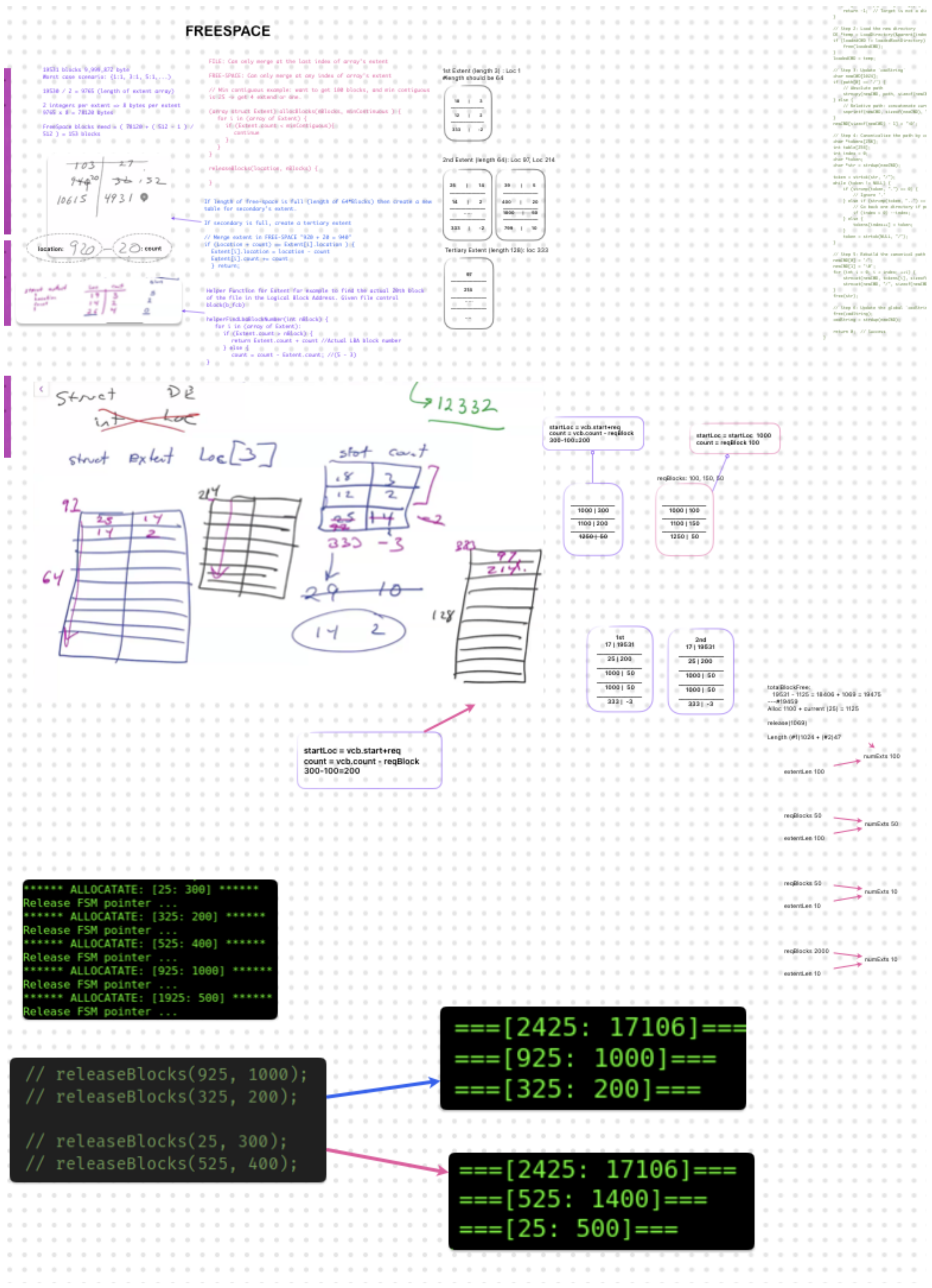
```
initial {
        vcb->frees_pace_loc = 1; (start after vcb)
        vcb->count_fs_blocks = calBNeed(n, 512, 5%);
        vcb->count_extent ++ // increase 1
        vcb->last_extent_size = numberOfBlocks;

        set firstExtent = [6: 19531]

        write to Extent to disk

        return vcb->frees_pace_loc
}
extent_st* allocateBlocks() {
        ptr* = malloc(numOfBlock * bizeSize);

        add data and write to memory
}
```

```c
#define PRIMARY_EXTENT_TB vcb->block_size/sizeof(extent_st)

#define TERTIARY_EXTENT_TB ( PRIMARY_EXTENT_TB * 2 )
#define TERTIARY_EXT_COUNT -3

#define FRAGMENTATION_PERCENT 0.05 //Number of Blocks need for FreeSpace
#define FREESPACE_START_LOC 1


typedef struct freespace_st {
    unsigned int extentLength;    // primary extent cursor in free space map
    unsigned int totalBlocksFree; // total blocks are free on disk
    unsigned int reservedBlocks;  // total number of blocks reserved for extents
    unsigned int maxExtent;       // max extent can store on reserved blocks

    unsigned int secExtLength;    // second extent cursor

    unsigned int terExtTBLoc;     // tertiary extent start loc
    unsigned int terExtTBLenth;     // tertiary extent length
} freespace_st;
```

```c
#define MAX_FILENAME 32
#define UNUSED_ENTRY '\0' // Marker for unused entries
#define DIRECTORY_ENTRIES 50

typedef struct {
    time_t creation_time;                 // creation time of the file or directory
    time_t modification_time;             // last modification time
    time_t access_time;                   // last access time

    int file_size;                        // size of the file or directory
    int is_directory;                     // 1 if directory, 0 if file
    int is_used;                          // 1 if used, 0 if unused

    char file_name[MAX_FILENAME];         // File or directory name
    extent_st block_location; // An of extent for data blocks (Processing... )

} directory_entry;
```

```c
/* Structure representing a single extent of contiguous blocks.
 * - startLoc: The starting location of the contiguous block.
 * - countBlock: The number of contiguous blocks with startLoc included
 */
typedef struct extent_st {
    int startLoc;
    int countBlock;
} extent_st;

/* Structure for managing multiple extents in block requests.
 * - extents: Pointer to an array of extent_st structures.
 * - size: The total number of extents
 */
typedef struct extents_st {
    extent_st *extents;
    int size;
} extents_st;
```

The above image shows Volume Control Block related notes.



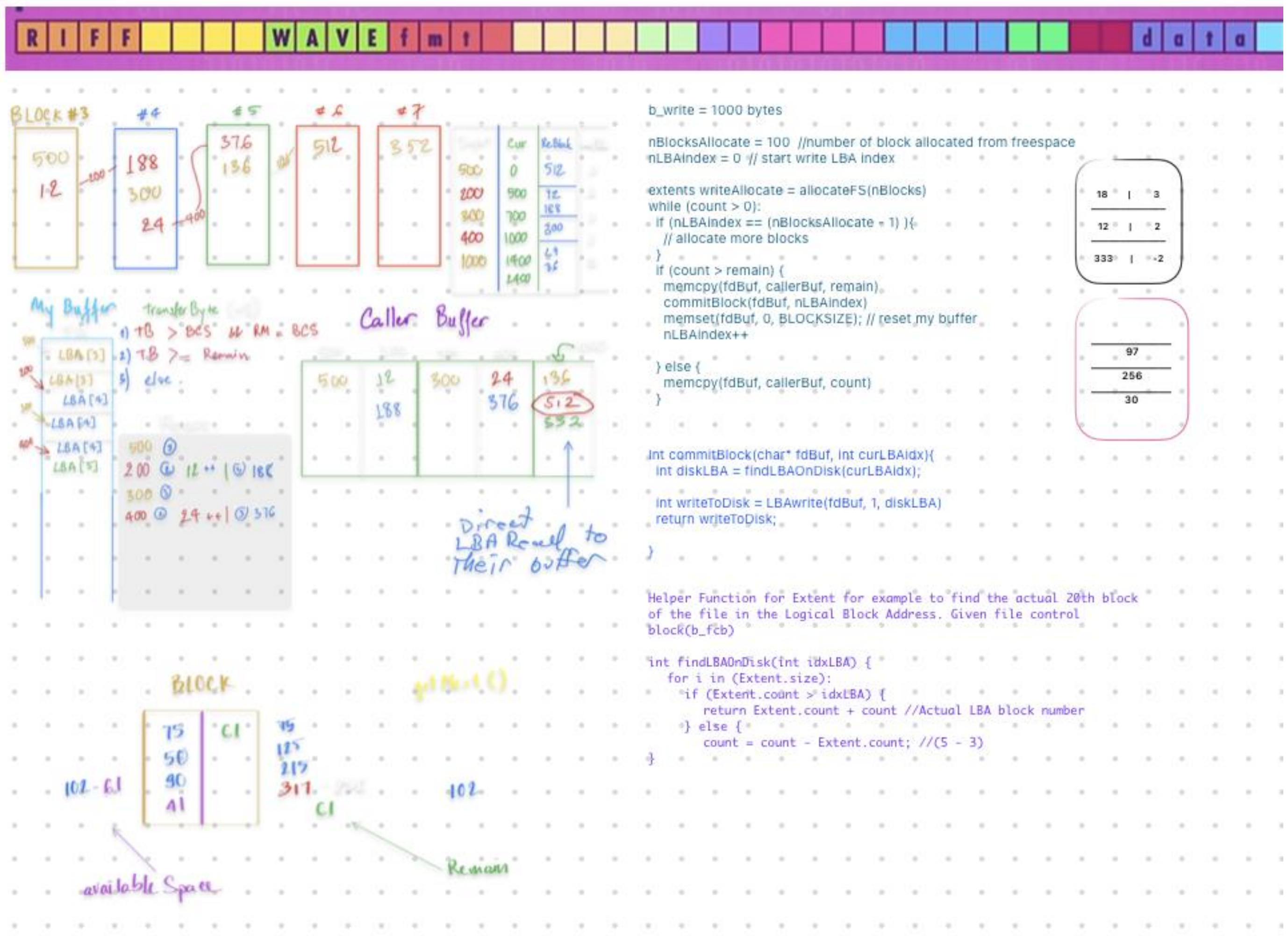
```
****** ALLOCATATE: [25: 300] ******
Release FSM pointer ...
****** ALLOCATATE: [325: 200] ******
Release FSM pointer ...
****** ALLOCATATE: [525: 400] ******
Release FSM pointer ...
****** ALLOCATATE: [925: 1000] ******
Release FSM pointer ...
****** ALLOCATATE: [1925: 500] ******
Release FSM pointer ...
```

```
// releaseBlocks(925, 1000);
// releaseBlocks(325, 200);

// releaseBlocks(25, 300);
// releaseBlocks(525, 400);
```

```
===[2425: 17106]===
===[925: 1000]===
===[325: 200]===
```

```
===[2425: 17106]===
===[525: 1400]===
===[25: 500]===
```

The above image shows a collection of notes related to Free Space Management.

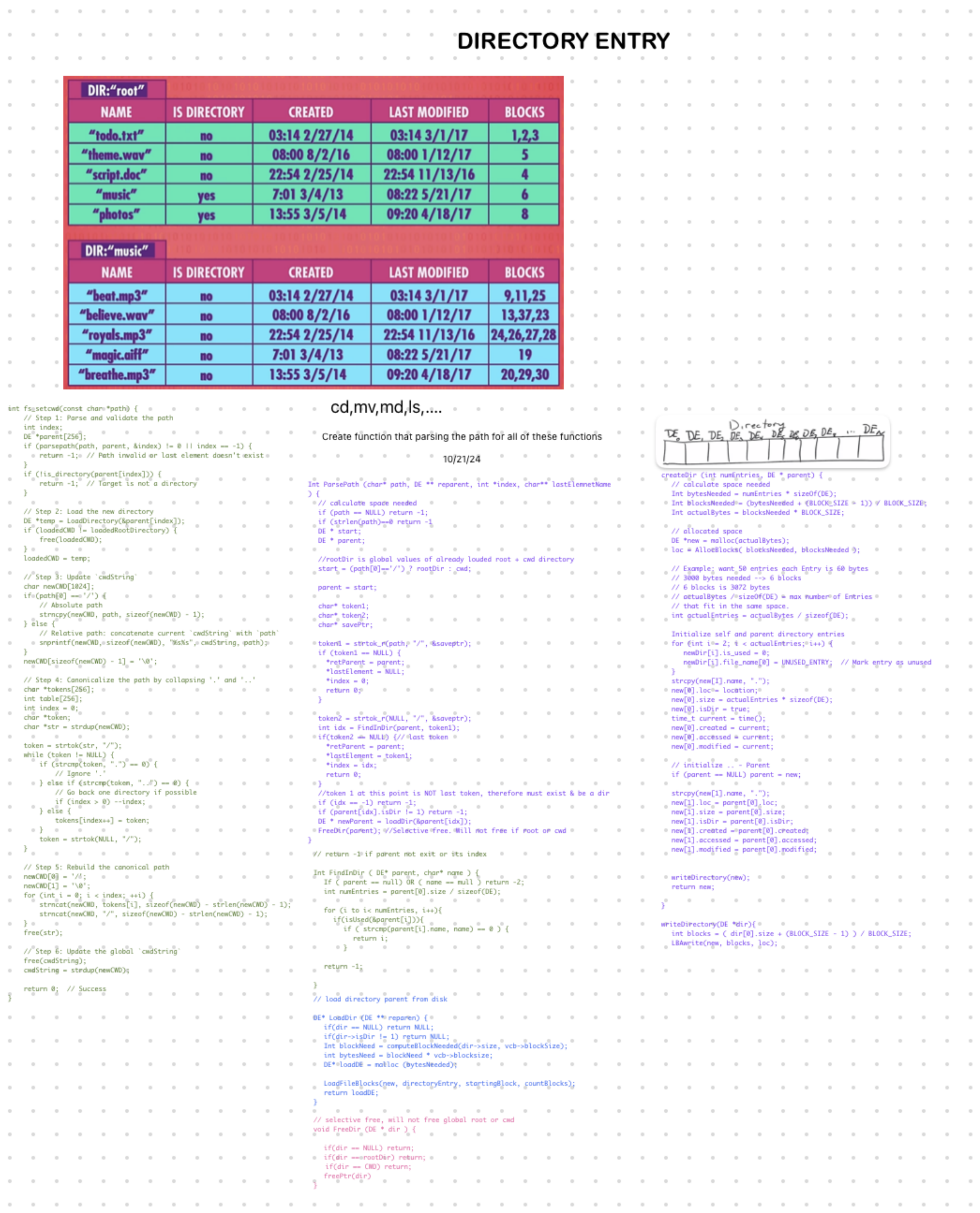


```
b_write = 1000 bytes

nBlocksAllocate = 100  //number of block allocated from freespace
nLBAIndex = 0 // start write LBA index

extents writeAllocate = allocateFS(nBlocks)
while (count > 0):
 if (nLBAIndex == (nBlocksAllocate - 1) ){
   // allocate more blocks
 }
 if (count > remain) {
   memcpy(fdBuf, callerBuf, remain)
   commitBlock(fdBuf, nLBAIndex)
   memset(fdBuf, 0, BLOCKSIZE); // reset my buffer
   nLBAIndex++

 } else {
   memcpy(fdBuf, callerBuf, count)
 }


int commitBlock(char* fdBuf, int curLBAIdx){
 int diskLBA = findLBAOnDisk(curLBAIdx);

 int writeToDisk = LBAwrite(fdBuf, 1, diskLBA)
 return writeToDisk;

}
```

Helper Function for Extent for example to find the actual 20th block of the file in the Logical Block Address. Given file control block(b_fcb)

```
int findLBAOnDisk(int idxLBA) {
   for i in (Extent.size):
      if (Extent.count > idxLBA) {
         return Extent.count + count //Actual LBA block number
      } else {
         count = count - Extent.count; //(5 - 3)
}
```

The image above are notes related to block-level operations.



DIRECTORY ENTRY

DIR:"root"

NAME	IS DIRECTORY	CREATED	LAST MODIFIED	BLOCKS
"todo.txt"	no	03:14 2/27/14	03:14 3/1/17	1,2,3
"theme.wav"	no	08:00 8/2/16	08:00 1/12/17	5
"script.doc"	no	22:54 2/25/14	22:54 11/13/16	4
"music"	yes	7:01 3/4/13	08:22 5/21/17	6
"photos"	yes	13:55 3/5/14	09:20 4/18/17	8

DIR:"music"

NAME	IS DIRECTORY	CREATED	LAST MODIFIED	BLOCKS
"beat.mp3"	no	03:14 2/27/14	03:14 3/1/17	9,11,25
"believe.wav"	no	08:00 8/2/16	08:00 1/12/17	13,37,23
"royals.mp3"	no	22:54 2/25/14	22:54 11/13/16	24,26,27,28
"magic.aiff"	no	7:01 3/4/13	08:22 5/21/17	19
"breathe.mp3"	no	13:55 3/5/14	09:20 4/18/17	20,29,30

```
int fs_setcwd(const char *path) {
    // Step 1: Parse and validate the path
    int index;
    DE *parent[256];
    if (parsepath(path, parent, &index) != 0 || index == -1) {
        return -1;  // Path invalid or last element doesn't exist
    }
    if (!is_directory(parent[index])) {
        return -1;  // Target is not a directory
    }

    // Step 2: Load the new directory
    DE *temp = LoadDirectory(&parent[index]);
    if (loadedCWD != loadedRootDirectory) {
        free(loadedCWD);
    }
    loadedCWD = temp;

    // Step 3: Update `cwdString`
    char newCWD[1024];
    if (path[0] == '/') {
        // Absolute path
        strncpy(newCWD, path, sizeof(newCWD) - 1);
    } else {
        // Relative path: concatenate current `cwdString` with `path`
        snprintf(newCWD, sizeof(newCWD), "%s%s", cwdString, path);
    }
    newCWD[sizeof(newCWD) - 1] = '\0';

    // Step 4: Canonicalize the path by collapsing '.' and '..'
    char *tokens[256];
    int table[256];
    int index = 0;
    char *token;
    char *str = strdup(newCWD);

    token = strtok(str, "/");
    while (token != NULL) {
        if (strcmp(token, ".") == 0) {
            // Ignore '.'
        } else if (strcmp(token, "..") == 0) {
            // Go back one directory if possible
            if (index > 0) --index;
        } else {
            tokens[index++] = token;
        }
        token = strtok(NULL, "/");
    }

    // Step 5: Rebuild the canonical path
    newCWD[0] = '/';
    newCWD[1] = '\0';
    for (int i = 0; i < index; ++i) {
        strncat(newCWD, tokens[i], sizeof(newCWD) - strlen(newCWD) - 1);
        strncat(newCWD, "/", sizeof(newCWD) - strlen(newCWD) - 1);
    }
    free(str);

    // Step 6: Update the global `cwdString`
    free(cwdString);
    cwdString = strdup(newCWD);

    return 0;  // Success
}
```

cd,mv,md,ls,....

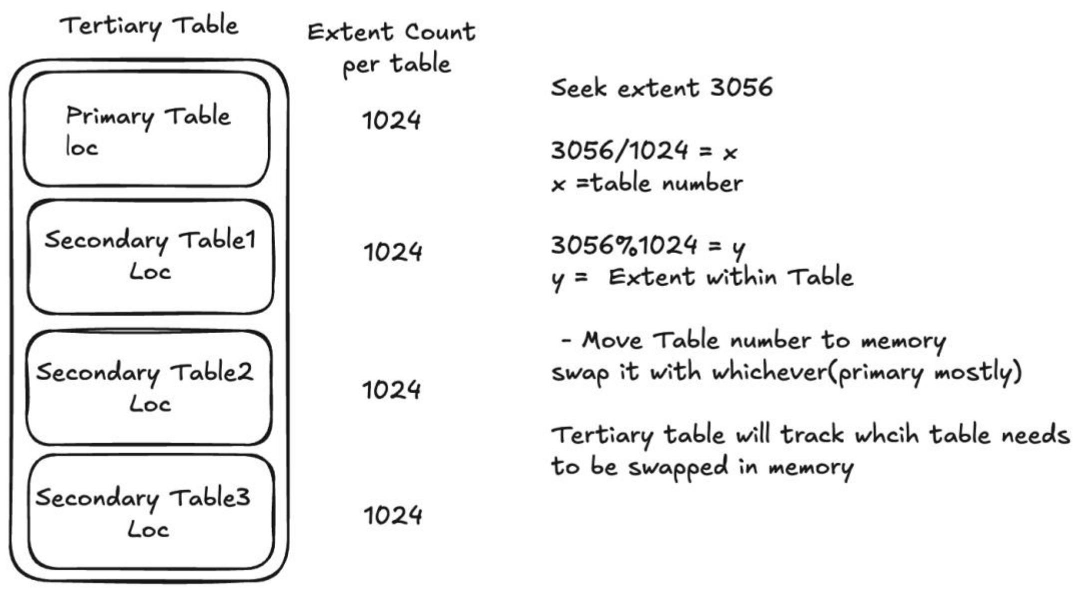
Create function that parsing the path for all of these functions

10/21/24

```
Int ParsePath (char* path, DE ** reparent, int *index, char** lastElemnetName
) {
  // calculate space needed
  if (path == NULL) return -1;
  if (strlen(path)==0 return -1
  DE * start;
  DE * parent;

  //rootDir is global values of already louded root + cwd directory
  start = (path[0]=='/') ? rootDir : cwd;

  parent = start;

  char* token1;
  char* token2;
  char* savePtr;

  token1 = strtok_r(path, "/", &saveptr);
  if (token1 == NULL) {
    *retParent = parent;
    *lastElement = NULL;
    *index = 0;
    return 0;
  }

  token2 = strtok_r(NULL, "/", &saveptr);
  int idx = FindInDir(parent, token1);
  if(token2 == NULL) {// last token
    *retParent = parent;
    *lastElement = token1;
    *index = idx;
    return 0;
  }
  //token 1 at this point is NOT last token, therefore must exist & be a dir
  if (idx == -1) return -1;
  if (parent[idx].isDir != 1) return -1;
  DE * newParent = loadDir(&parent[idx]);
  FreeDir(parent); //Selective free. Will not free if root or cwd
}

 // return -1 if parent not exit or its index

 Int FindInDir ( DE* parent, char* name ) {
   If ( parent == null) OR ( name == null ) return -2;
   int numEntries = parent[0].size / sizeof(DE);

   for (i to i< numEntries, i++){
     if(isUsed(&parent[i])){
       if ( strcmp(parent[i].name, name) == 0 ) {
         return i;
     }

   return -1;

 }
 // load directory parent from disk

 DE* LoadDir (DE ** reparen) {
   if(dir == NULL) return NULL;
   if(dir->isDir != 1) return NULL;
   Int blockNeed = computeBlockNeeded(dir->size, vcb->blockSize);
   int bytesNeed = blockNeed * vcb->blocksize;
   DE* loadDE = malloc (bytesNeeded);

   LoadFileBlocks(new, directoryEntry, startingBlock, countBlocks);
   return loadDE;
 }

 // selective free, will not free global root or cmd
 void FreeDir (DE * dir ) {

   if(dir == NULL) return;
   if(dir == rootDir) return;
   if(dir == CWD) return;
   freePtr(dir)
 }
```

```
createDir (int numEntries, DE * parent) {
  // calculate space needed
  Int bytesNeeded = numEntries * sizeOf(DE);
  Int blocksNeeded = (bytesNeeded + (BLOCK_SIZE - 1)) / BLOCK_SIZE;
  Int actualBytes = blocksNeeded * BLOCK_SIZE;

  // allocated space
  DE *new = malloc(actualBytes);
  loc = AllocBlocks( blocksNeeded, blocksNeeded );

  // Example: want 50 entries each Entry is 60 bytes
  // 3000 bytes needed --> 6 blocks
  // 6 blocks is 3072 bytes
  // actualBytes / sizeOf(DE) = max number of Entries
  // that fit in the same space.
  int actualEntries = actualBytes / sizeof(DE);

  Initialize self and parent directory entries
  for (int i = 2; i < actualEntries; i++) {
     newDir[i].is_used = 0;
     newDir[i].file_name[0] = UNUSED_ENTRY;  // Mark entry as unused
  }
  strcpy(new[I].name, ".");
  new[0].loc = location;
  new[0].size = actualEntries * sizeof(DE);
  new[0].isDir = true;
  time_t current = time();
  new[0].created = current;
  new[0].accessed = current;
  new[0].modified = current;

  // initialize .. - Parent
  if (parent == NULL) parent = new;

  strcpy(new[1].name, ".");
  new[1].loc = parent[0].loc;
  new[1].size = parent[0].size;
  new[1].isDir = parent[0].isDir;
  new[1].created = parent[0].created;
  new[1].accessed = parent[0].accessed;
  new[1].modified = parent[0].modified;


  writeDirectory(new);
  return new;

}

writeDirectory(DE *dir){
  int blocks = ( dir[0].size + (BLOCK_SIZE - 1) ) / BLOCK_SIZE;
  LBAwrite(new, blocks, loc);
```

The image above are notes related to shell commands and the Directory Entry structure.



This image shows a conceptual illustration of how the tertiary and secondary extent tables track free space.

IMPLEMENTATION

The basic outline of our file system's implementation follows the predefined milestone 1 to 3 requirements of the assignment. Our approach to implementation tackles as many of the requirements in each phase as possible.

In milestone 1, we had to ensure that the Volume Control Block (VCB) in block 0 was written to the virtual volume. Additionally, we implemented the initialization of the free space management system. This involved configuring the free space structure and reserving the necessary blocks, though the procedure for releasing allocated blocks back into the free space pool was not required at this stage. Finally, we wrote and initialized the root directory, ensuring it included at least the essential entries for "." (current directory) and ".." (parent directory).

To document this milestone, we utilized the HexDump utility to analyze the volume file, capturing and presenting the VCB, Free Space structure, and root directory contents. The HexDunp print can be found under this document's "Appendix" section. Furthermore, we provided detailed descriptions of the VCB, Free Space, and Directory system designs, which can be found under the "Milestone 1" subsection in "Project Progression."

For milestone 2, our focus shifted to implementing directory-based functionality and integrating these features into the interactive shell (fsshell.c). This milestone development emphasized allowing users to navigate, manage, and query the file system. Core functionalities such as making directories, listing directory contents, printing the working directory, and changing directories were implemented to provide basic file system navigation and organization capabilities. Components of fsshell.c that is prefixed with the function name "fs_" were completed in this milestone and are described in detail under the "Milestone 2" subsection in "Project Progression."

Objectives in milestone 3 focused on the implementation of file input/output operations on the block level that is stored and accessed from the virtual volume. Completed implementations in this phase are prefixed with the function name "b_". b_open initializes and manages file descriptors, validates file paths, and processes flags for creating or modifying files. The b_seek function provided precise control over file pointer positioning and supported modes for seeking from the beginning, current position, or end of a file while ensuring boundaries are respected. File data manipulation is facilitated through b_write and b_read, which buffered and handled direct operations with small and large data sizes. These functions dynamically managed storage allocation, handled fragmented files, and ensured data consistency during read and write operations. The b_close function cleaned up file usage by clearing buffers, updating metadata, and releasing resources to maintain the file system's integrity.

Note: a description of the challenges faced during implementation and the decisions made in our design is stated under “Project Progression” in the subsection “Implementation Challenges.”

Development Environment

Our team’s development environment consists of the following specifications:

Virtualization: VMware Workstation 17.5.2
Operating System: Ubuntu 22.04.3
Kernel: 6.8.0-40-generic
Architecture: x86_64 and arm66
IDE: Visual Studio Code 1.85.2
Version control: git and GitHub
External libraries: math.h, pthread.h, readline.h, history.h

PROJECT PROGRESSION

This section thoroughly describes the components of our filesystem organized from milestones 1 to 3.

MILESTONE 1

Key components implemented to reach the completion of milestone 1 are the volume control block, free space management, and recording keeping of the directory entries.

The Volume Control Block:

The design of our volume_control_block (VCB) structure is integral to the management and operation of the file system. It contains both persistent fields, which are stored on the virtual volume, and runtime-only pointers used during the file system's operation.

The volume_name field provides a human-readable identifier for the volume, enabling users to distinguish between different volumes. The signature field is used to validate whether the volume is formatted for our file system. If a volume does not exist or lacks the specified signature value, a new volume is created and initialized with a unique signature. This value is critical for ensuring compatibility with our file system.

The total_blocks field represents the total number of blocks in the volume, calculated by dividing the volume's size by the block size. For instance, in a volume with 10,000,000 bytes and a block size of 512 bytes, the total number of blocks would be 19,531. The block_size field specifies the size of each block in bytes, ensuring consistent allocation and management of storage.

The root_loc field stores the starting block location of the root directory. At runtime, a pointer to the root directory's directory_entry structure is maintained in root_dir_ptr, enabling efficient access and management of directory operations. Similarly, the free_space_loc field marks the starting location of the free space block on virtual volume, and the fs_st structure manages details about the volume’s free space, such as the number of available blocks and their distribution.

For runtime operations, the free_space_map pointer provides access to the dynamically managed free space map in memory, which is initialized during the file system's setup. Additionally, the cwdLoadDE pointer points to the directory_entry of the current working directory (CWD), while the cwdStrPath pointer stores the string representation of the CWD path, facilitating navigation and path resolution.

Together, these fields and pointers form a comprehensive structure that balances persistent storage requirements with efficient runtime operations, ensuring the integrity and functionality of the file system.

Our Free Space Management design is as follows:

The extent structure used for tracking free space in our file system is defined in Extent.h. This structure is fundamental for managing the allocation and deallocation of blocks within the volume. Each extent represents a contiguous sequence of free blocks, characterized by a starting block location and the number of blocks it encompasses.

The details about the free space management are encapsulated in the freespace_st structure defined in the FreeSpace.h source file. The free space block location is designated as block 1, represented by the constant FREESPACE_START_LOC. This location is crucial as it marks the starting point for the free space map stored on the virtual volume.

When the volume is formatted for the first time, the totalBlocksFree field holds the value 19,530. This is calculated from the total number of blocks (19,531), minus the blocks occupied by the Volume Control Block (VCB) and the free space structures. The first block is reserved for the VCB, so the available free blocks are adjusted accordingly.

The reservedBlocks field contains the value computed by the calBlocksNeededFS function. To determine the number of blocks needed to track free space, we estimated a fragmentation level of 5%. This estimation allows us to calculate the total space required for extents by multiplying the fragmentation percentage by the total number of blocks. The computeBlocksNeeded function then calculates how many blocks are necessary to store these extents, ensuring that at least one block is allocated. In our case, this results in 16 blocks reserved for free space management.

The extentLength field holds the position or index after the most recently added element in the free_space_map. It is incremented each time a new extent is added, effectively indicating the total number of extents in the map. The first element in free_space_map is at index 0, so extentLength corresponds to the count of extents present.

The curExtentLBA field, when initialized for the first time, holds the value from FREESPACE_START_LOC. This field tracks the current Logical Block Addressing (LBA) location for the extents, facilitating accurate allocation and deallocation operations.

The maxExtent field represents the maximum number of extent structures that can be stored. This is calculated by multiplying the number of reserved blocks (16) by the number of extents per block. Given that each block is 512 bytes and each extent structure is 8 bytes, there are 64 extents per block. Therefore, maxExtent holds a maximum value of 1,024 extent structures (16 blocks * 64 extents per block).

The terExtLength field starts at 0 when first initialized and increments whenever a tertiary extent is added. The terExtTBloc attribute holds the starting location of the tertiary extent table on the virtual volume. The terExtTBMap pointer references the tertiary extent table in memory, allowing for efficient access and management. This structure aids in locating the necessary primary or secondary extent tables, especially in scenarios where the free space map extends beyond the primary allocation.

Our file system effectively tracks free space by meticulously managing these structures and fields, ensuring optimal allocation and minimizing fragmentation.

The Directory Entry

The structure begins with fields that store timestamps using the time_t data type, which occupy the first 8 bytes of the entry. These timestamps track critical moments in the lifecycle of a directory or file, including creation, last modification, and last access times. These fields are initialized for both parent and child directory entries, ensuring that all hierarchical relationships within the file system retain accurate temporal metadata.

The size of a file or directory is represented by the file_size field, which records its size in bytes. To accommodate storage for directory entries, the system assumes a placeholder value of 50 entries per directory. This assumption enables the allocation of sufficient storage while maintaining flexibility for user-defined needs. To determine the space requirements, the total bytes for 50 entries are calculated as 50 * 80, equating to 4000 bytes. When converted into block allocation, this results in 8 blocks (4096 bytes) when considering a block size of 512 bytes. This calculated allocation ensures that sufficient space is reserved for each directory's entries.

A distinguishing characteristic of a directory or file is captured in the is_directory field. This flag identifies whether the entry represents a directory or a file. For instance, the root directory is initialized with is_directory set to 1, signifying its status as a directory. The is_used flag further specifies whether the entry is actively in use. For the root directory, this flag is set to 1 for both its parent and child relationships, reflecting its active status and role as a central point in the file system hierarchy.

The data_block field links the directory entry to its associated data blocks via the extents array. Memory for this array is dynamically allocated using the allocateBlocks function. For the root directory, the first element in the extents array is assigned to its data_block, ensuring proper tracking of its allocated space. This design provides a direct and efficient mechanism for accessing the blocks associated with a file or directory.

MILESTONE 2

File operations were the focus of this milestone. Below are the functions implemented during this phase of development.

fs_setcwd:

Changing the current working directory by validating the specified path and making sure it meets essential criteria: the path must exist, correspond to a directory, and have a valid index. The function employs a path parser (parsePath) to locate the target directory's metadata. If a valid directory is identified, its metadata is loaded into the VCB’s cwdLoadDE. To manage memory efficiently, the previously loaded current working directory (CWD) metadata is freed unless it points to the root directory, ensuring only the currently active directory's data remains in memory.

The function also updates the CWD's string representation by replacing the old path with a newly generated, clean version using cleanPath. This step ensures the path remains logically consistent and accurate.

fs_getcwd:

Retrieves the current working directory path and copies it into the provided buffer. It uses the VCB’s cwdStrPath attribute, which maintains the current directory's string representation.

fs_isFile:

This function checks if a given path leads to a file. It calls the fs_isDir function to see if the path is not a directory. If the path is not a directory, it returns 0 (meaning it’s a file). If it is a directory, it returns -1.

fs_isDir:

This function checks if a given path leads to a directory. It uses the parsePath function to verify if the path is valid and gather details about it. If the path is valid and points to a directory, it returns -1. If it is not valid or does not point to a directory, it returns 0.

fs_mkdir:

This function begins by validating the provided path using parsePath. If the path is valid, it creates a new directory using “createDirectory” and inserts it into the parent directory using “makeDirOrFile.” Memory is freed after the operation, and changes are written back to the volume to update the filesystem.

“makeDirOrFile” updates the parent directory by finding an unused entry, updating it with the new directory’s metadata (such as name, extents, and timestamps), and marking it as used. It differentiates between directories and files, setting the appropriate flags and data for each. If no free entry is found, it returns -1, indicating failure.

fs_opendir:

Opens a directory at pathname. Initializes tracked attributes of a directory and returns a pointer to a new record in preparation for the file or directory's data. Pointer to fdDir or NULL if the directory does not exist or an error occurs.

fs_readdir:

Retrieves the next directory entry in an "open" directory via dirp as input,and returns the directory or file details. Pointer to fs_diriteminfo or NULL when the end of the directory is reached, or an error occurs.

fs_closedir:

Closes an "open" directory and frees associated resources by freeing the pointer to the directory or file data record structure. If it is not valid or does not point to a directory, it returns 0.

fs_stat:

Retrieves and populates a file meta data for a given path. The path of the file or directory to be queried. Also, a pointer to a structure where the file metadata will be stored. If the path is valid and points to a directory, it returns -1. If it is not valid or does not point to a directory, it returns 0.

fs_rmdir/fs_delete:

The "fs_rmdir" and "fs_delete" functions are handled by the "deleteBlod" function. This function manages the deletion of both files and directories in an optimal way by parsing the provided "pathname" with the "parsePath" function to extract metadata. If the parsing fails or the target does not exist, the function returns an error, ensuring that the function only operates on valid paths. It then verifies that the target type matches the expected type ("isDir"): file ("isDir == 0") or directory ("isDir == 1"). Any mismatch results in an immediate error, preventing accidental deletions or invalid operations.

When handling directories, it verifies that the directory is empty before deleting by loading the directory entry into memory with "loadDir" and checking for emptiness using "isDirEmpty". If the directory cannot be loaded or is not empty, the function stops with an error message. Finally, the target is marked as unused in the parent directory's metadata using "removeDE", the allocated blocks are released to free space, and the changes are written back to the virtual volume using "writeDirHelper".

MILESTONE 3

This subsection is a detailed overview of block-level operations for our filesystem.

b_open:

b_open function initializes and sets up the file in the virtual file system, returning a file descriptor index for subsequent operations. It validates the file path using parsePath, rejecting invalid inputs like "." or "..". If the file doesn't exist and the O_CREAT flag is set, it creates a new file via makeDirOrFile. Errors in path validation or file creation result in an immediate failure.

For existing files, the function processes flags such as O_TRUNC, which clears associated data blocks and updates the modification time, or O_APPEND, which sets the file pointer to the end of the file. It records metadata such as the parent directory index and initializes FCB properties like flags, current block index, logical block address (LBA) position, and a memory buffer for file operations. The file’s access time is updated, and any memory allocation or setup error results in an error.

b_seek:

The b_seek function provides file pointer positioning within a file. It accepts three different positioning modes through the whence parameter: SEEK_SET sets the pointer to the start of the file and moves relative to that position using the given offset, SEEK_CUR sets it relative to its current position, and SEEK_END sets the pointer to the end of the file and adjusts it using the provided offset in case of SEEK_END the offset is usually negative. The function also ensures that the pointer follows the block boundaries and handles those cases, making sure the data is consistent. The function checks for file boundaries and makes sure it doesn't seek beyond the file size. It returns either a new position of the pointer or -1, depending on the case or error that occurred.

b_write:

This function writes data to a file in a virtual file system. It first checks if the file descriptor (fd) is valid and the file is opened in write-only mode. If the file has no extent (i.e., no blocks available) for writing, it allocates additional virtual volume blocks using allocateFSBlocks, passing in the default number of blocks (100 blocks). This process increases the space in chunks that double each time, merging new blocks with existing ones when possible to reduce fragmentation.

Writing is handled by writeBuffer, which uses both buffered and direct writing. Smaller pieces of data are stored in an internal buffer and written to the virtual volume when the buffer is full. Larger chunks of data are written directly into virtual volume without using the buffer. This approach provides fast and efficient writing for both small and large data sizes. The file's progress is tracked using an index that records how much data has been written.

The actual virtual volume writing is performed by commitBlocks, which converts the current block position into logical block addresses on the virtual volume. It writes as many contiguous blocks as possible and handles fragmented blocks when needed. If the file runs out of space during this step, more blocks are allocated dynamically to make sure the file can grow as needed while keeping the storage organized.

b_read:

This function reads data from a file in the virtual file system, supporting both sequential and random access patterns. It first validates the file descriptor, buffer, and read permissions, then calculates how many bytes can be read based on the current file position and size. The function optimizes performance by detecting block-aligned reads of 512 bytes or more and reading these directly from the virtual volume to the user's buffer, while smaller or unaligned reads use an intermediate buffer to handle partial blocks efficiently. For fragmented files, it uses findLBAOnDisk to locate the actual blocks on the virtual volume and manages reading across multiple extents while maintaining proper block boundaries and file position tracking. The function handles disk I/O through LBAread calls, using the file control block's buffer for caching data when needed. It updates the file's position and access time metadata, and returns the number of bytes actually read, 0 for EOF, or -1 on error. This comprehensive implementation ensures efficient handling of both large sequential reads and small random accesses while maintaining data consistency across fragmented storage.

b_close:

The b_close function is responsible for finalizing file operations by releasing the resources associated with an open file. It begins by validating the file descriptor to confirm it corresponds to an active file. If the file is opened in write mode, the function ensures any unsaved changes are committed to disk, updating file metadata such as size and modification time as needed. This includes trimming excess blocks to release unused space back to the free space map. Additionally, it manages the release of the file's buffer memory and resets the file's control block to indicate that the file is no longer active. The function guarantees that all associated resources are properly freed and the file is in a consistent state. The goal of this function is to ensure proper closure and the availability of resources for subsequent operations. Note that the O_WRONLY flag is used to determine whether the file was opened in write-only mode.

IMPLEMENTATION CHALLENGES

This section of the document provides an overview of decisions made when overcoming challenges faced during the design and implementation of our file system. Subsequent sections describe bugs in our code and ideas for future improvement.

At the start of this assignment, our team found learning the provided assignment code and connections between functions challenging to comprehend. Eventually, with enough exposure to reading and making changes to the code, connections with how the different components worked together became clearer. For some of us, beginning to use Visual Studio Code (our IDE) was challenging as well. However, with each other's help, we soon grew familiar with its features.

Implementation Considerations

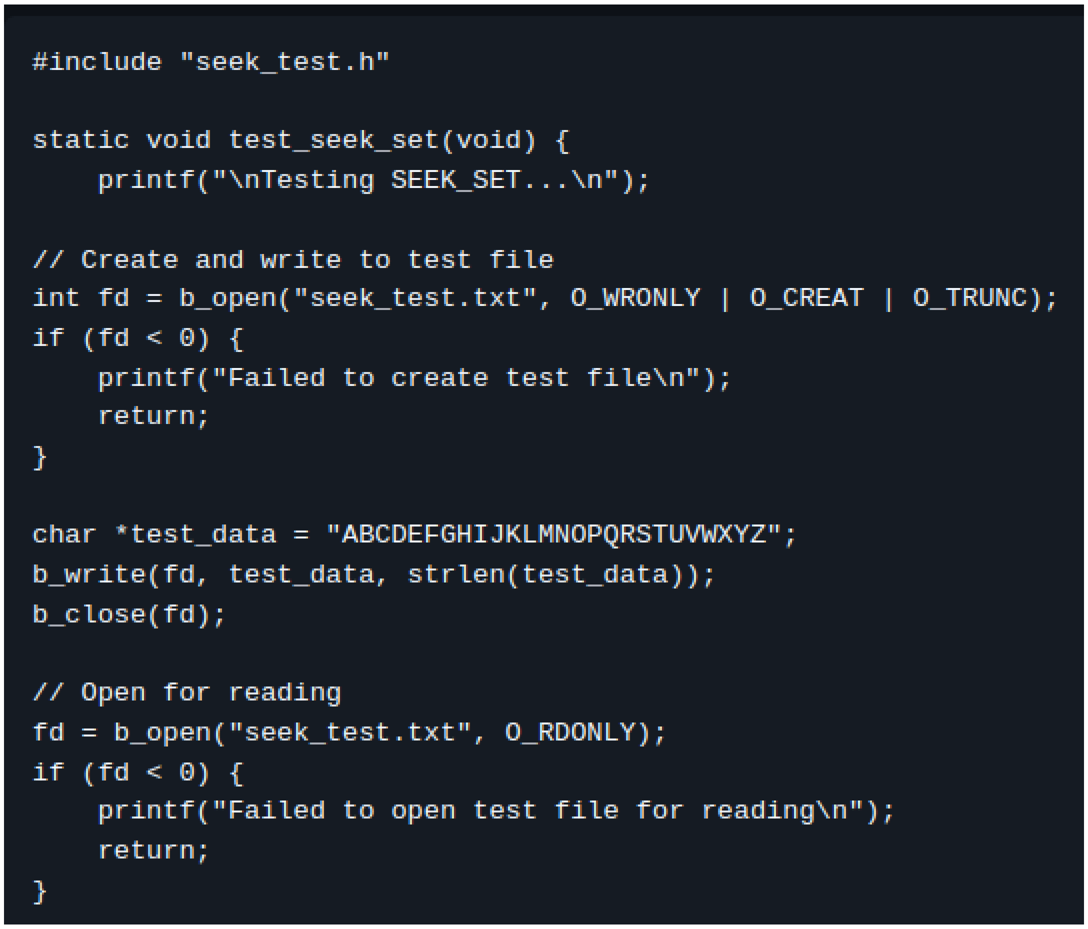
The implementation of the VCB and DE structures determines the type of data needed to be stored on the virtual volume and what should be loaded into memory during runtime. This consideration also affects choosing the optimal structure for managing free space. To achieve this, we decided to use an Extent-based structure for tracking groups of contiguous blocks. Extents allow efficient organization of free space by grouping blocks into larger, continuous segments and maintaining multiple levels of tables (primary, secondary, and tertiary) for scalability and flexibility.

Our decision to use extents over alternatives like bitmaps is predominately due to the extent's feature of significantly reducing the storage overhead required for metadata. Instead of representing each block individually, as in a bitmap, an extent represents a range of contiguous blocks with a single entry. For example, a single extent can describe 1,000 contiguous blocks with just two values: the starting block number and the length of the range. This approach reduces the number of digits and the total size of the metadata stored on disk, freeing up space for actual data.

In addition to reducing metadata, extents provide critical performance benefits. By minimizing the fragmentation of data and metadata, extents enable faster sequential access to large files, which is ideal for systems handling large datasets or files. Extents also simplify allocation and deallocation by handling groups of blocks at once rather than individual blocks. This approach improves efficiency for both storage management and runtime operations.

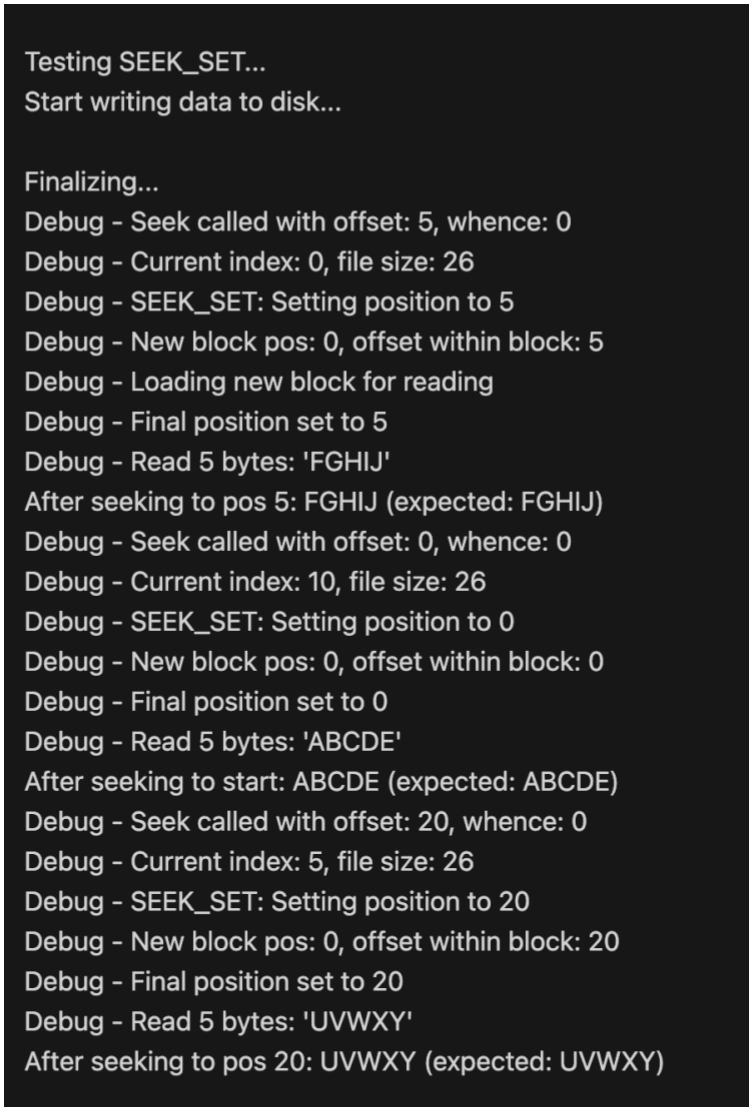
TESTING

To ensure that our filesystem met the objectives of each milestone, we thoroughly planned the design of the components before implementing them as functions. Unit tests were created to cover a range of cases. Below is an example of a unit test to ensure that the b_seek function sequentially seeks and reads the test_data five characters at a time.



```
#include "seek_test.h"

static void test_seek_set(void) {
    printf("\nTesting SEEK_SET...\n");

// Create and write to test file
int fd = b_open("seek_test.txt", O_WRONLY | O_CREAT | O_TRUNC);
if (fd < 0) {
    printf("Failed to create test file\n");
    return;
}

char *test_data = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
b_write(fd, test_data, strlen(test_data));
b_close(fd);

// Open for reading
fd = b_open("seek_test.txt", O_RDONLY);
if (fd < 0) {
    printf("Failed to open test file for reading\n");
    return;
}
```

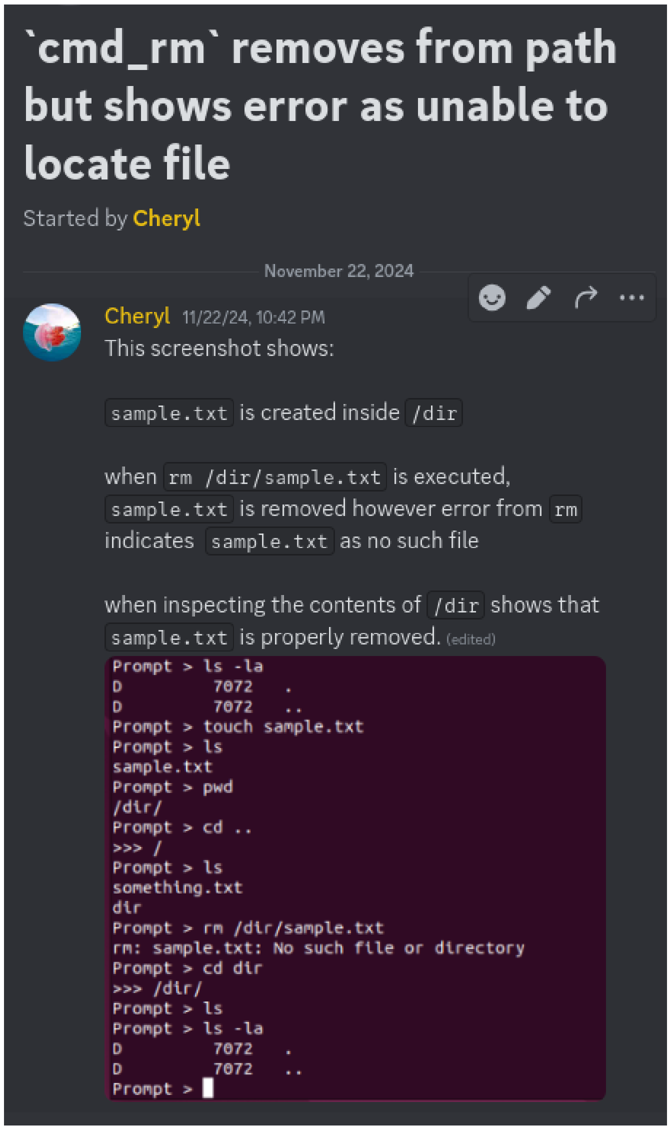
The output of the above unit test is shown below:



```
Testing SEEK_SET...
Start writing data to disk...

Finalizing...
Debug - Seek called with offset: 5, whence: 0
Debug - Current index: 0, file size: 26
Debug - SEEK_SET: Setting position to 5
Debug - New block pos: 0, offset within block: 5
Debug - Loading new block for reading
Debug - Final position set to 5
Debug - Read 5 bytes: 'FGHIJ'
After seeking to pos 5: FGHIJ (expected: FGHIJ)
Debug - Seek called with offset: 0, whence: 0
Debug - Current index: 10, file size: 26
Debug - SEEK_SET: Setting position to 0
Debug - New block pos: 0, offset within block: 0
Debug - Final position set to 0
Debug - Read 5 bytes: 'ABCDE'
After seeking to start: ABCDE (expected: ABCDE)
Debug - Seek called with offset: 20, whence: 0
Debug - Current index: 5, file size: 26
Debug - SEEK_SET: Setting position to 20
Debug - New block pos: 0, offset within block: 20
Debug - Final position set to 20
Debug - Read 5 bytes: 'UVWXY'
After seeking to pos 20: UVWXY (expected: UVWXY)
```

To test our implementation during each milestone phase, a team member would test the code of another member. Improvements and fixes were submitted directly to GitHub on the main branch after discussions on Discord. Using the thread feature on Discord, individual bug issues would have separate threads. This choice of bug triage is similar to opening an issue on GitHub; however, we opted to use the thread feature on Discord as we wanted a central point of reference.

The following screenshot is an example of a Discord thread containing a bug issue:



`cmd_rm` removes from path but shows error as unable to locate file

Started by Cheryl

November 22, 2024

Cheryl 11/22/24, 10:42 PM

This screenshot shows:

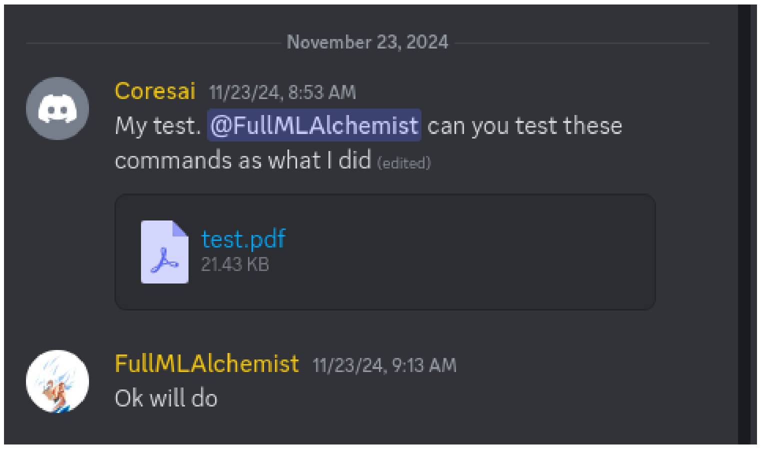
`sample.txt` is created inside `/dir`

when `rm /dir/sample.txt` is executed, `sample.txt` is removed however error from `rm` indicates `sample.txt` as no such file

when inspecting the contents of `/dir` shows that `sample.txt` is properly removed. (edited)

```
Prompt > ls -la
D         7072   .
D         7072   ..
Prompt > touch sample.txt
Prompt > ls
sample.txt
Prompt > pwd
/dir/
Prompt > cd ..
>>> /
Prompt > ls
something.txt
dir
Prompt > rm /dir/sample.txt
rm: sample.txt: No such file or directory
Prompt > cd dir
>>> /dir/
Prompt > ls
Prompt > ls -la
D         7072   .
D         7072   ..
Prompt > 
```

Creating a specific thread for a particular bug allows specific conversations and material to be linked directly to the issue. An example is shown in the screenshot below. A thread is marked closed when the bug is resolved.

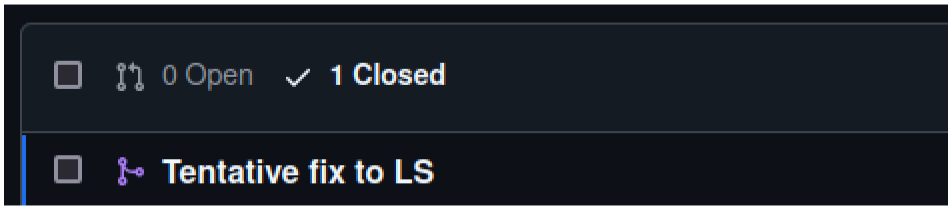

November 23, 2024

Coresai 11/23/24, 8:53 AM
My test. @FullMLAlchemist can you test these commands as what I did (edited)

test.pdf
21.43 KB

FullMLAlchemist 11/23/24, 9:13 AM
Ok will do

Additionally, fixes were merged with the main branch when a pull request was approved.


0 Open ✓ 1 Closed

Tentative fix to LS

The subsequent subsections illustrate known bugs and potential bugs that may occur in our file systems.

KNOWN AND POTENTIAL BUGS

This section describes known and potential bugs that may occur in our file systems.

Fixed Memory Size for File Meta Data

The buffer size to store a file's retrieved data from a disk is 4 kilobytes. In the unlikely event that a file's metadata exceeds 4 kilobytes, its data will be truncated when displayed to the user.

Fixed Volume Size for Directory Entries

Our filesystem is designed to use up to a maximum of 8 extents, where each extent is 8 bytes. A directory entry that exceeds 64 bytes will cause the file system to work incorrectly.

Altering the Macro BUFFERLEN from 200 to 2000

The macro BUFFERLEN was given as part of the assignment's base code. Since there was no restriction to change the shell driver, we have changed it to 2000. This corrected a bug that would not properly display an image (a non-text file) that was transferred from the Ubuntu VM to our file system and back to the VM.

POTENTIAL IMPROVEMENTS

A clear step forward, if we were to continue this file system project beyond milestone three, would be to create additional levels of extent for our Directory Entry. The implementation would be similar to the extent of handling in the free space management.

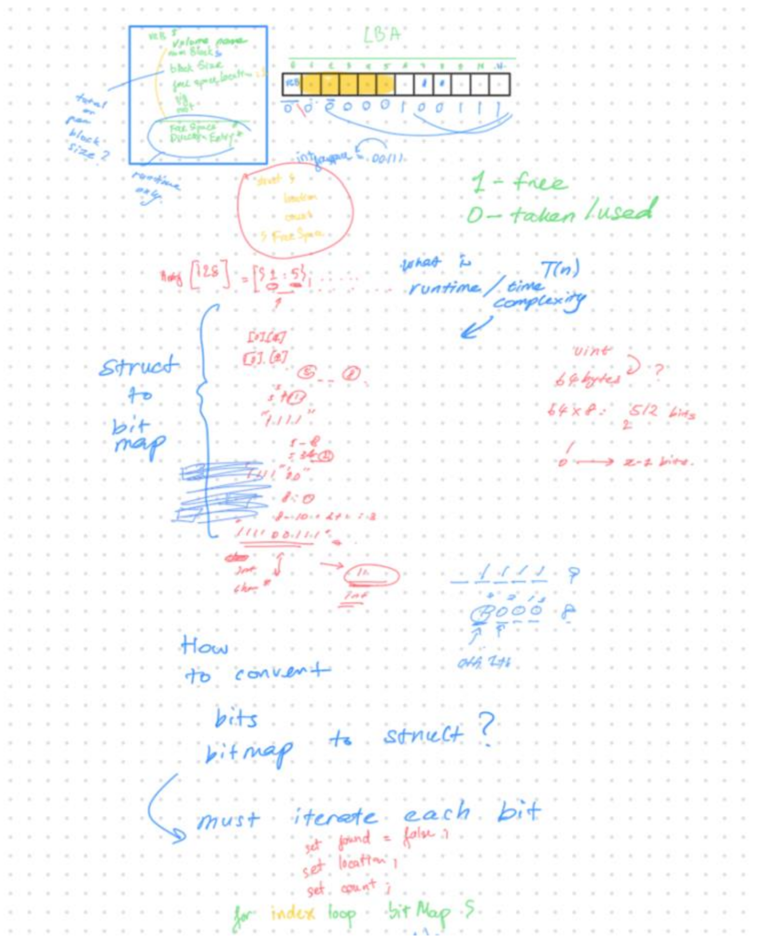
The current implementation of our file system does not utilize the b_seek function. Testing has been conducted on b_seek using unit tests. Options that could be taken as part of future improvements to our code base would be to call b_seek somewhere in our implementation, remove it entirely, or have b_read refactored to utilize b_seek.

TEAM AND COLLABORATION

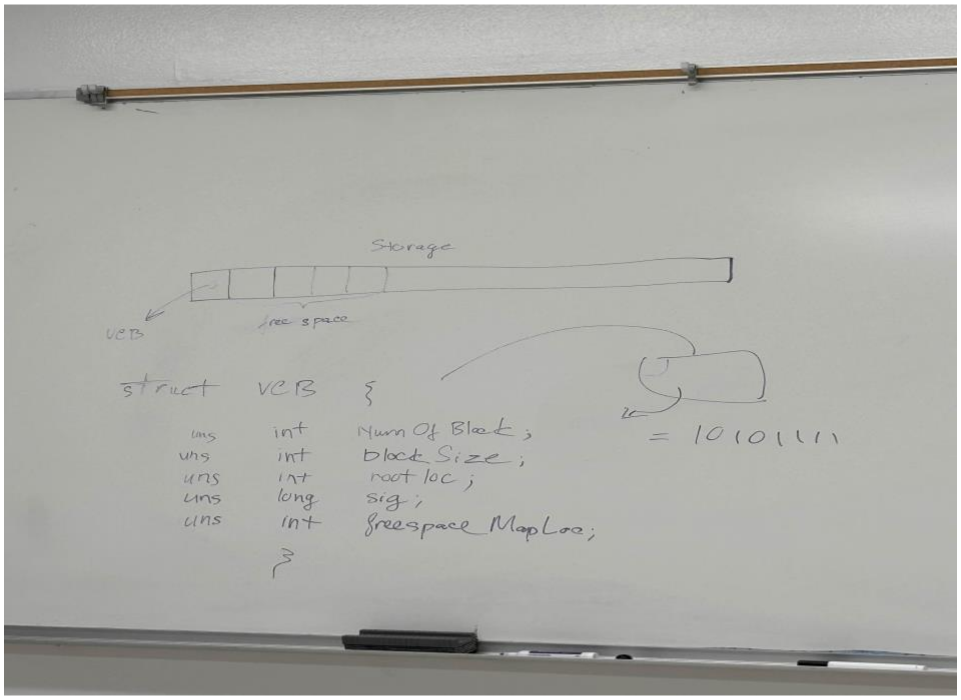
This section encompasses details on how our team functioned as a unit to complete this assignment and lessons and improvements on teamwork to consider in the future. It is organized by Collaboration Details and Roles and Responsibilities.

Collaboration Details

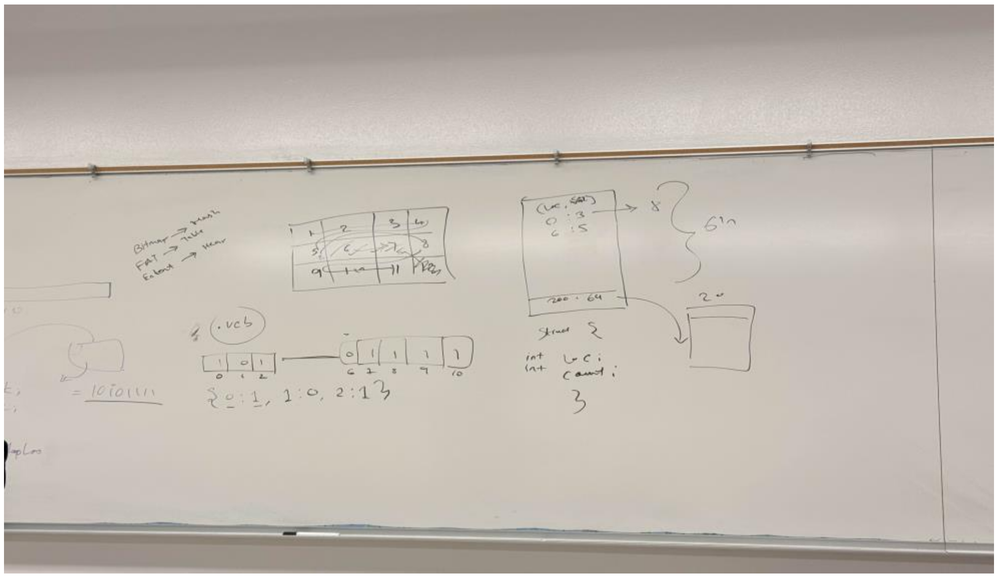
During our brainstorming sessions, we utilized drawing tools such as Apple's FreeForm app and OneNote to illustrate our ideas and to ensure understanding among the team. Saving our notes in this format allowed easy sharing of information.



We often used the whiteboard to emphasize and help each other understand core concepts that are necessary to progress and meet the project's goals. Routinely, we would agree to do preliminary research on topics and facts prior to our team discussions and sketch our logic before implementing any code.



Above is a whiteboard sketch of our initial VCB Structure.



This image of the whiteboard was taken when planning the design of the free space management (i.e., using a Bitmap versus a File Allocation Table versus using Extents)

Discord was our preferred mode of communication to maintain continuous development. This stream of communication allowed quick resolution of bugs in code, clarification of plans and decisions, the scheduling for online and in-person meetings, and served as a central point of reference for information we have gathered. We would also make use of its voice call and screen-sharing features to collaborate online.

Clear communication and mutual understanding were understood among our team as necessary for seamless collaboration. We strived to enforce this standard and helped each other as best as possible to complete the work and learn together.

Roles and Responsibilities

To complete this assignment, our team delegated and took responsibilities during each phase of development. Below in tabulated form shows how work was distributed to create our functional file system.

MILESTONE 1

Team Member	Milestone 1 - Roles and Responsibilities
Arvin Ghanizadeh	- Worked on the write-up with my teammates. - Discussed the usage of extents and delegation.
Atharva Walawalkar	- Helped plan and discuss the structure for freespace, VCB, and DE for the initial part of the project with the rest of the group members. - Helped Implement DE.c, which includes the directory entry structure and functions like readdir, writedir and loaddir; these are helper functions for DE.
Cheryl Fong	- Brainstormed with Dan on the decision to use bitmaps or extents. - Completed the writeup submission for milestone 1.
Danish Nguyen	- Organized all files and structured them into separate folders for easier navigation and management. - Volume Control Block (VCB) - Implemented the VCB, which involved handling file system initialization and formatting the disk. This process included creating a new VCB, setting up the free space map, and initializing directory entries. If a valid signature was detected on the disk, this loaded the root directory entry and free space map into memory. - Ensured that data was correctly written back to the disk and cleaned up memory related to the existing

	filesystem. - Free Space Management - Designed the free space structure and stored it in the same LBA as the VCB for the free space management system. - Starting with the implementation of a FreeSpace manager with a primary extent table, the file system was expanded to include secondary and tertiary extent tables into the free space map. Tests were created to verify the functionality and accuracy of these extent tables. - Directory Entry Management - Implemented the directory entry as outlined in the lecture guidelines - Created the createDirectory function, enabling the creation of new directories, including the “.” and “..” directory entries. - This implementation of Directory Entry successfully supports creating, reading, and writing directory entries from the disk using a single extent. - When a teammate attempted to improve the system to support multiple extents, issues arose with reading and writing directory entries on disk. Due to implementation challenges, his teammate was unable to resolve the issue. He had a meeting with his teammate and came up with a solution, but his teammate didn't have time to continue working on DE. As a result, he reverted to his earlier version, refining the directory_entry structure and the createDirectory function to support multiple extents. He developed new helper functions, writeDirHelper and readDirHelper, which allowed efficient reading and writing of directory entries to the disk. - Assisted a teammate with the write-up for Milestone 1, using hexdump to explain the structure of the VCB, Directory Entries, and Free Space. He explained what value is stored in those structures and how to convert hexadecimal values into the actual values.

MILESTONE 2

Team Member	Milestone 2 - Roles and Responsibilities
Arvin Ghanizadeh	- Worked on fs_stat.
Atharva Walawalkar	- Worked on the Parsepath. - Helped debug CLI functions cp and mv and their working. - Tested the commands where md failed to access the third level of a directory and debugged Parsepath.
Cheryl Fong	- Helped Arvin with fs_stat and help make the decision to use a buffer size to 4 kilobytes. - Worked on fs_read. - Worked on fs_open - Worked on fs_close - Completed the ls command implementation.
Danish Nguyen	- Collaborated with a teammate to implement ParsePath, assisting them with pushing the function to GitHub due to his teammate having issues with GitHub conflicts. - fs_getcwd and fs_setcwd (Created helper functions for fs_setcwd such as loadDir and cleanPath) - fs_mkdir (Implemented the makeDirOrFile helper function to handle directory and file creation) - fs_delete and fs_rmdir (Created the deleteBlob function for optimal deletions and implemented supporting functions: isDirEmpty and removeDE) - Provided debugging support for ParsePath, resolved parsePath throwing errors when creating new directories, and assisted another teammate in fixing an issue where fs_opendir, fs_readdir failed to open the current working directory.

MILESTONE 3

Team Member	Milestone 3 - Roles and Responsibilities
Arvin Ghanizadeh	- Worked on the b_close and helped with the write-up.
Atharva Walawalkar	- Worked on b_read. - Worked on b_seek.
Cheryl Fong	- Completed the write-up of the project. - Helped Arvin with the foundations of b_close.
Danish Nguyen	- b_open - b_write (Developed helper functions, including commitBlocks, writeBuffer, findLBAOnDisk, allocateFSBlocks, trimBlocks, and trimBlocksHelper, to optimize the file-writing process). - b_read: Created a fully functional version of the file-read operation, enabling comprehensive testing of file read-and-write functionality. During the process, he discovered that a teammate had implemented a similar b_read function with different variable names, demonstrating impressive parallel thinking. - b_close: He assisted his teammate with the implementation of *b_close*, explaining the essential checks for the write function when closing a file, memory cleanup requirements, releasing unused blocks to free space, and the necessary steps to write data back to the disk when closing a file.

APPENDIX

A section containing supplementary data of our file system. The following sections are contents of a sample volume when first initialized.

Contents of the SampleVolume, starting at block 1 for 4 blocks:

```
000200: 46 53 2D 50 72 6F 6A 65  63 74 00 00 00 00 00 00 | FS-Project......
000210: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000220: D6 43 A9 21 4F F8 1D 5B  4B 4C 00 00 00 02 00 00 | �C�!O�.[KL......
000230: 11 00 00 00 01 00 00 00  2C 4C 00 00 10 00 00 00 | ........,L......
000240: 01 00 00 00 01 00 00 00  00 04 00 00 00 00 00 00 | ................
000250: FF FF FF FF 00 00 00 00  00 00 00 00 00 00 00 00 | ����............
000260: 90 79 AD 27 13 BB 00 00  C0 99 AD 27 13 BB 00 00 | �y�'.�..���'.�..
000270: 00 00 00 00 00 00 00 00  70 79 AD 27 13 BB 00 00 | ........py�'.�..
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 | ................

000400: 1F 00 00 00 2C 4C 00 00  00 00 00 00 00 00 00 00 | ....,L..........
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 | ................
```

Contents of the SampleVolume, starting at block 2 for 4 blocks:

```
000400: 1F 00 00 00 2C 4C 00 00  00 00 00 00 00 00 00 00 | ....,L..........
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 | ................
```

Contents of the SampleVolume, starting at block 18 for 4 blocks:

```
002400: E5 A6 4E 67 00 00 00 00  E5 A6 4E 67 00 00 00 00 | �Ng....�Ng....
002410: E5 A6 4E 67 00 00 00 00  A0 1B 00 00 01 00 00 00 | �Ng....�.......
002420: 01 00 00 00 01 00 00 00  2E 00 00 00 00 00 00 00 | ................
002430: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002440: 00 00 00 00 00 00 00 00  11 00 00 00 0E 00 00 00 | ................
002450: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002460: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002470: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002480: 00 00 00 00 00 00 00 00  E5 A6 4E 67 00 00 00 00 | ........�Ng....
002490: E5 A6 4E 67 00 00 00 00  E5 A6 4E 67 00 00 00 00 | �Ng....�Ng....
0024A0: A0 1B 00 00 01 00 00 00  01 00 00 00 01 00 00 00 | �...............
0024B0: 2E 2E 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0024C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0024D0: 11 00 00 00 0E 00 00 00  00 00 00 00 00 00 00 00 | ................
0024E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0024F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

002500: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002510: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002520: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002530: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002540: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002550: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002560: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002570: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002580: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002590: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0025A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0025B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0025C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0025D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0025E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0025F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

002600: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002610: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002620: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002630: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002640: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002650: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002660: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002670: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002680: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

```
002690: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0026A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0026B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0026C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0026D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0026E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0026F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

002700: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002710: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002720: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002730: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002740: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002750: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002760: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002770: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002780: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002790: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0027A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0027B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0027C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0027D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0027E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0027F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

002800: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002810: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002820: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002830: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002840: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002850: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002860: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002870: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002880: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002890: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0028A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0028B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0028C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0028D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0028E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0028F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

002900: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002910: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002920: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002930: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002940: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```

```
002950: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002960: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002970: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002980: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002990: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0029A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0029B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0029C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0029D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0029E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0029F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

002A00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002A10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002A20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002A30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002A40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002A50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002A60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002A70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002A80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002A90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002AA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002AB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002AC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002AD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002AE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002AF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

002B00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002B10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002B20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002B30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002B40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002B50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002B60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002B70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002B80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002B90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002BA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002BB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002BC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002BD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002BE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
002BF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```