FILE SYSTEM IMPLEMENTATION USING C

Group 2
Operating Systems

AN INTRODUCTION TO FILE SYSTEMS

A file system is a process that manages how and where data on a storage disk is stored, accessed and managed. It is a logical disk component that manages a disk's internal operations.

Regardless of type and usage, a disk contains a file system and information about where disk data is stored and how it may be accessed by a user or application. A file system typically manages operations, such as storage management, file naming, directories/folders, metadata, access rules and privileges.

Commonly used file systems include File Allocation Table 32 (FAT 32), New Technology File System (NTFS) and Hierarchical File System (HFS).

ADVANTAGES OF FAT32

Overcomes the Volume Limitation of FAT16	The purpose of FAT32 was to overcome the limitations of FAT16 and add support for larger media. The major enhancements introduced by FAT32 included support for much larger volumes, better performance and more flexibility and robustness.
Wide Operating System Compatibility	Another advantage of FAT32 is its compatibility across popular operating systems to include Windows, macOS, and Linux distributions such as Debian and Ubuntu. It is also compatible with Android, video game consoles, and media players. A storage hardware formatted based on this file system can be read and written by these operating systems.

FILE ALLOCATION TABLE 32 (FAT32)

FAT32 is a version of the File Allocation Table (FAT) file system that was introduced by Microsoft in 1996 with its Windows 95 OEM Service Releases 2 (OSR2) operating system. It is an extension of Microsoft's FAT16 file system.

A FAT file system contains four basic regions: reserved, FAT, root directory and file and data directory region.

Directories cannot have more than 65,535 files and other directories.

FAT directories are not sorted or indexed. This decreases efficiency for many operations like creation of new files, when the directory size becomes large.

OUR FILE SYSTEM IMPLEMENTATION

Our file system is based on the FAT32. It has the following functions which the user can access (after running the FileSystem.c using "`gcc -o fs FileSystem.c && ./fs`" on the terminal).

- exit
- ls
- mkdir
- close
- open
- cd
- create
- rm
- rmdir
- read
- write
- sf
- format

GROUP MEMBERS AND WORKFLOW

Names		Functions
Ishan	Mehak	`Utils`
Dushant	Mohit	`Basic`
Madhav	Dheeraj	`Read`
Harsh	Jatin	`Write`
Ishika	Diksha	`Delete`
Nikunj	Harshvir	`Create`
Jayshree	Naman	`Others`

UTILS

```
// Initialize FCB according to the given parameters
void fcb_init(fcb *new_fcb, const char* filename, unsigned short first,
unsigned char attribute);

// Initializes an file entry with the given parameters
void useropen_init(useropen *openfile, int dirno, int diroff, const char*
dir);

// Realeaste the used FAT Blocks
void fatFree(int id);

// Get a Free FAT Block
int getFreeFatid();
```

UTILS

```
// Get a Free File Table Entry
int getFreeOpenlist();

// Get the next FAT Table and if not, then create it
int getNextFat(int id);

// Check if is its legal to open the table below File Table
int check_fd(int fd);

// Split a path using '/'
int spiltDir(char dirs[DIRLEN][DIRLEN], char *filename);
```

UTILS

```
// Delete the last directory from the string
void popLastDir(char *dir);

// Splits the last directory from the string
void splitLastDir(char *dir, char new_dir[2][DIRLEN]);

// Get the corresponding Disk Block Number and Offset of a certain length in
a certain FAT, which is used to record the position of a File item
corresponding to the FAT in its Parent Directory
void getPos(int *id, int *offset, unsigned short first, int length);

// Normalize and check the path
int rewrite_dir(char *dir);
```

BASIC OPERATIONS

```
// According to the Disk Block Number and Offset, directly read the information
of the specified length from the FAT
int fat_read(unsigned short id, unsigned char *text, int offset, int len);

// Read the specified length information of a file
int do_read(int fd, unsigned char *text, int len);

// Write the specified length information directly from the FAT according to the
Disk Block Number and Offset
int fat_write(unsigned short id, unsigned char *text, int blockoffset, int len);
```

BASIC OPERATIONS

```
// Write the specified length to a file
int do_write(int fd, unsigned char *text, int len);

// Find the FCB Folder with the corresponding name from the Directory
int getFcb(fcb* fcbp, int *dirno, int *diroff, int fd, const char *dir);

// Open a File under a Directory
int getOpenlist(int fd, const char *org_dir);

// Open the File
int my_open(char *filename);
```

READ OPERATIONS

```
// Read the FCB Information under a Folder
int read_ls(int fd, unsigned char *text, int len);

// Print out the FCB Information under a Folder
void my_ls();

// Print out the contents of a File according to the File Pointer
int my_read(int fd);

// Re-read the FCB Content of a File from Disk
void my_reload(int fd);
```

WRITE OPERATION

```
// Writes information entered by the user after opening the File
int my_write(int fd);
```

DELETE OPERATION

```
// Set the Free of FCB of a specified Directory to 1
void my_rmdir(char *dirname);

// Set the Free of FCB of a specified File to 1
void my_rm(char *filename);
```

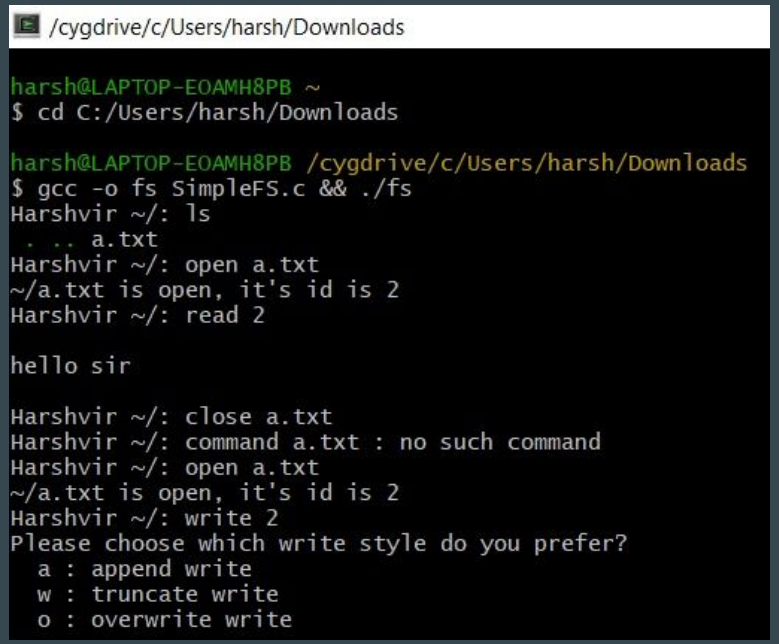
CREATE OPERATIONS

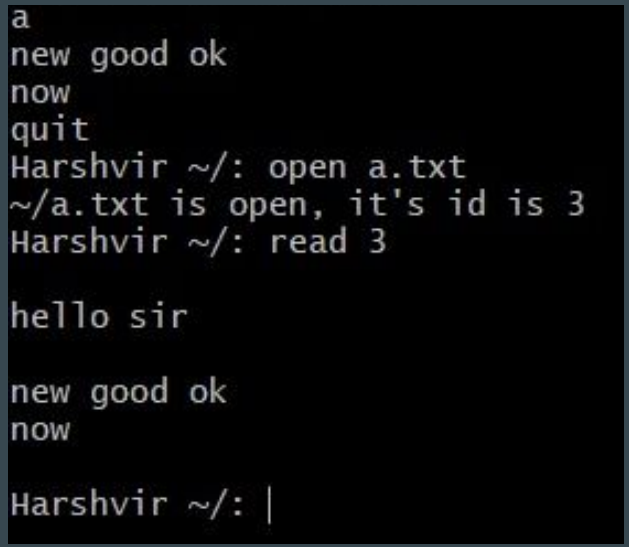
```
// Format
void my_format();

// Create a File or Folder in the specified Directory
int my_touch(char *filename, int attribute, int *rpafd);

// Call 'touch' to create a File
int my_create(char *filename);

// Call 'touch' to create a Folder
void my_mkdir(char *dirname);
```

OTHER OPERATIONS

```
// Exit the System
void my_exitsys();

// Store the FAT Information of a File
void my_save(int fd);

// Close a File
void my_close(int fd);

// Use 'my_open' to switch the current Directory to the specified Directory
void my_cd(char *dirname);
```

IMPLEMENTATION SCREENSHOTS





IMPLEMENTATION SCREENSHOTS

```
Ishan ~/: mkdir test
Ishan ~/: ls
 . .. test
Ishan ~/: rmdir test
Ishan ~/: ls
 . ..
Ishan ~/:
```

```
Ishan ~/: mkdir test
Ishan ~/: cd test
Ishan ~/test/: create test.txt
~/test/test.txt is created, it's id is 0
Ishan ~/test/: ls
 . .. test.txt
Ishan ~/test/: rm test.txt
Ishan ~/test/: ls
 . ..
Ishan ~/test/:
```