CSE430: Operating Systems
Project 3 Solution
Vageesh Bhasin

1. General Approach:

As per the notes of Prof. Violet, the LibFS implements 2 types of nodes, *i_nodes* and *data_nodes.* The i_nodes store information related to the file/directory like size, type and pointers to data_nodes. The data nodes hold the data of the file/directory. In case of a directory, the data_nodes hold entries of child files/directories.
The bitmaps are created to store in exacly 8 bits per byte using the bitwise operators. Random bits have been corrupted to provide non-contiguous i_node and data_node allocation.
Once the APIs were written, the program was thoroughly tested against other test cases for correct behaviors and correct error codes. The program was also checked with *Valgrind* for memory leaks, and optimizations were done to avoid memory leaks.

2. Data Structures:

1. *Superblock:*
    - Stores a magic number which is used to verify if the FS is okay or corrupted.
2. *Bitmap:*
    - i_node bitmap [Type: char array of length 125] – Each byte of bitmap stores 8 bits, which are referenced to identify which i_nodes are empty or full.
    - data_node bitmap [Type: char array of length 125] – Each byte of bitmap stores 8 bits, which are referenced to identify which data_nodes are empty or full.
3. *i_node* [Total size: 128 bytes]
    - File_Type [Type: int] – Stores the type of file
    - File_Size [Type: int] – Stores the file size
    - Data_ptr [Type: int array of size 30] – Stores sector numbers of data nodes
4. *data_node* [Total size: 512 bytes]
    - Data – Stores data of the file/directory
5. *directory_node* [Total size: 20 bytes]
    - Name [Type: char array of size 16] – Stores the name of child file/directory
    - i_node_num [Type: int] – Stores the i_node number of the child file/directory
6. *Open_File_Table_Bitmap* [Total size: 32 bytes (256/8)]
    - Each byte of bitmap stores 8 bits, which are referenced to identify which file_table entries are empty or full.
7. *Open_File_Table:* [Total size: 256 * Sizeof(*File_Table_Entry*) bytes]
    - Stores *file_table_entry* data which is referenced to identify opened files
8. *File_Table_Entry:* [Total size: 4*(int) + 256 * (char) bytes]
    - i_node_num [Type: int] – Stores the i_node number of the opened file
    - read_file_ptr [Type: int] – Stores the current location of the file to read from (Used in File_Read())
    - write_file_ptr [Type: int] – Stores the current location on the file to write to (Used in File_Write())
    - path [Type: char array of size 256] – Stores the complete path of the opened file

3. Function Descriptions:

3.1 Generic File System API:

- **FS_Boot(char *path):**
  - Checks the path to see if the file can be booted up, returns error if file is corrupted (Superblock magic number not matching)
  - If the file is not created, sets up a new File System by performing the following task:
    - Writes magic number into superblock of disk at sector 0
    - Sets up i_node and data_node bitmaps at sector 1 and 2 respectively, and randomly corrupts a total of 125 bits
    - Sets up i_nodes in the sectors 5-255
    - Stores FS path name in global variable

- **FS_Sync():**
  - Saves the disk into the file path retrieved from the global path name variable

3.2 File Access API:

- **File_Create(char *file):**
  - Retrieves free i_node and data_node numbers from bitmaps
  - Creates a new i_node with file_type = 'file', file_size = 0 and data_ptrs pointing to 0 sectors, except the first one which is pointing to the data_node number fetched in the above step.
  - Overwrites the i_node at index 'free i_node num' with this i_node
  - Adds a directory entry of this i_node to the parent
    - If parent is root, then directly adds to sector 256
    - If parent is not root, then traverses the input 'file' and creates directories if not existing
  - Toggles the bitmaps to 'allocated'
  - Returns 0 on success or -1 on failure & sets error_code to 'E_CREATE'

- **File_Open(char *pathname):**
  - Traverses the nodes, starting from root, using the pathname passed in.
  - If file is not found, return -1 with error_code set to 'E_NO_SUCH_FILE'
  - If maximum allowed files are already open, return -1 with error_code set to 'E_TOO_MANY_OPEN_FILES'
  - If file is found,
    - Finds an open file_table index from the bitmap
    - Creates an *Open_Table_Entry* with path set to 'pathname', read and write pointers set to 0 and i_node_number set to the i_node of this file.
    - Returns 0

- **File_Write(int fd, void *buffer, int size):**
  - Takes the file descriptor 'fd' and fetches the i_node
    - If 'fd' is corrupt, aborts with RC = -1 & Error_code = E_BAD_FD
    - If file_size of aslready maximum, aborts with RC = -1 & Error_code = E_NO_SPACE
  - Goes to the data_node of the file
  - Moves the file pointer to the current 'write file pointer' retrieved from the Open_File_table

- Writes 'size' bytes from 'buffer' to the file, until
  - File size exceeds maximum file size, aborts with RC = -1 & Error_Code = E_FILE_TOO_BIG
  - All bytes of buffer have been written to the file, which might include assigning new data_nodes to this file and writing to them
- Persists the update with Disk_Write() and returns 'size'

- **File_Close(int fd):**
  - Takes the file_descriptor value and unsets the corresposing index bit in the File_Table_Bitmap and returns 0
  - If file_descriptor is corrupt, aborts with RC = -1 & Error_Code = E_BAD_FD

- **BONUS CREDIT - File_Read(int fd, void *buffer, size)**
  - Takes the file descriptor 'fd' and fetches the i_node
    - If 'fd' is corrupt, aborts with RC = -1 & Error_code = E_BAD_FD
    - If current read pointer is at EOF, aborts with RC = 0 and no error code.
  - Goes to the data_node of the file
  - Moves the file pointer to the current 'read file pointer' retrieved from the Open_File_table
  - Reads 'size' bytes from file to 'buffer', until
    - All bytes have been read, which might including traversing to other data_nodes pointed by this i_node
  - Returns the bytes that have been read, which might be equal to or less than 'size' bytes.

3.3 Directory API:

- **Dir_Create(char *path):**
  - Retrieves free i_node and data_node numbers from bitmaps
  - Creates a new i_node with file_type = 'directory', file_size = 0 and data_ptrs pointing to 0 sectors, except the first one which is pointing to the data_node number fetched in the above step.
  - Overwrites the i_node at index 'free i_node num' with this i_node
  - Adds a directory entry of this i_node to the parent
    - If parent is root, then directly adds to sector 256
    - If parent is not root, then traverses the path name and creates directories if not existing
  - Toggles the bitmaps to 'allocated'
  - Returns 0 on success or -1 on failure & sets error_code to 'E_CREATE'

- **BONUS CREDIT - Dir_Read(char *path, void *buffer, int size):**
  - Traverses the i_nodes and d_nodes to find the last directory in the pathname
    - Return -1 if directory does not exist
  - Fetches the i_node and all the data_ptrs of the data_nodes belonging to this directory
  - Traverses each data_node to check directory entry, and writes it to the buffer until
    - All entries have been covered & then returns to entries, or
    - Entries overflow the buffer, & then return RC = -1 with Error_Code = E_BUFFER_TOO_SMALL

4. Assumptions:

- File_Write() appends to the file, i.e., even if a file is closed and then opened again, the file pointer will point to the end of the file.
- File_Read() always reads from the beginning of the file if it is freshly opened, otherwise it continues from its last read pointers.
- FS_Sync() does not persist Open_File_Table and its entries to the disk.

5. Things not implemented:

- N/A – Everything written in the project description have been implemented, including bonus credits.