Task 1 Overview (Board #1)

For this task you will be implementing a simulated file system with a simple directory structure and memory indexing policy.

The file system consists of 2^16 blocks of 256 bytes of memory. Within each of these blocks a single node can be stored.

This task defines the structure and node creation of this file system, manipulation of the file content will be covered in Task 2.

Before we begin we must declare the following:

```
#define BLOCK_SIZE 256
#define MAX_NAME_LENGTH 128
#define DATA_SIZE 254
#define INDEX_SIZE 127

typedef char data_t;
typedef unsigned short index_t;
```

Nodes

There exist four types of node:

1. Directory Information Nodes (DIR)
2. File Information Nodes (FILE)
3. Related Indices (INDEX)
4. Data (DATA)

These types are represented by the following enumerated type in your code:

```
typedef enum
{
   DIR,
   FILE,
   INDEX,
   DATA
} NODE_TYPE;
```

The FS_NODE contains descriptive information about either a directory or a file depending on its type:

```
typedef struct fs_node
{
   char name[MAX_NAME_LENGTH]; // name of the file or directory
   time_t creation; // creation time
   time_t last_access; // last access time
   time_t modified; // last modification time
   mode_t access_rights; // access rights for the file
   unsigned short owner; // owner ID
   unsigned short size; // Depends on type will be covered later.
   index_t block_ref; // Depends on type will be covered later.
} FS_NODE;
```

Nodes themselves are made up of their type and a union representing their content:

```
typedef struct node
{
   NODE_TYPE type;
   union
   {
      FS_NODE fd;
      data_t data[DATA_SIZE];
      index_t index[INDEX_SIZE];
   } content;
} NODE;
```

Memory is a large array of these nodes with a coorresponding bit vector keeping track of what space has been allocated:

```
NODE *memory; // allocate 2^16 blocks (in init)
char *bitvector; // allocate space for managing 2^16 blocks (in init)
```

I will now elaborate on the specifics of each type of node:

Board #2

Directory Information Nodes (DIR)

Directory information nodes are represented as the FS_NODEs described above They differ from file information nodes in the use of two of the variables contained within that struct:

- **size**: For directory information nodes the size refers to the number of files and directories that reside within this directory.
- **block_ref**: For directory information nodes the block reference always is the index of this directory's INDEX node (described in detail later). This INDEX node points to all of the FS_NODEs for any directories or files contained within this directory.

File Information Node (FILE)

File information nodes are also represented as FS*NODEs but differ from DIR nodes in how **size** and **block*ref** are used:

- **size**: refers to the number of bytes of content this file contains.
- **block_ref**:
    - **If the size of the file is less than 254**: This means all of the file's content can fit within a singular data node. In this instance the block reference holds the index of the data node containing all of this file's content.
    - **If the size of the file is greater than 254**: This means that the file's content needs to be distributed across several DATA nodes. The block reference therefore needs to be the index of an INDEX node containing the indices of the distributed DATA nodes. (See INDEX description).
    - **If the size of the file is equal to 254**: The program should crash. Just kidding. Do the above ^.

Related Indices (INDEX)

An INDEX node is represented as an array of 127 unsigned shorts, each representing the index of a block in memory related to the FS_NODE.

For a file that is larger than (or equal to) 254 the INDEX node holds the indices of the consecutive data blocks that make up the files content.

For a directory the INDEX node holds the indices of the FS_NODEs to all subdirectories or files within that

directory.

Data (DATA)

Just an array of 254 chars that make up part of (or all of) a file's content.

Superblock (Board #3.0 Split this board in half horizontally)

The superblock is just a special instance of a DIR node. It represents the root directory of the file system. It is set up at the file system initialization stage to have the following attributes:

- **name** = "/"
- **creation** = 0
- **access_rights** = Can't change name or delete
- **block_ref** is the location of an index block

The superblock always occupies the first block of memory so that it can be easily located. It's index block should probably be at the second block of memory.

Bit Vector (Board #3.5)

In addition to the nodes you will be using a bit vector for keeping track of which blocks of memory are taken and which are free. This will be represented as an array of chars. Each bit in the char represents one block of memory, meaning that each char represents 8 blocks of memory.

1 char == 1 block is incorrect.

You may want to look up bitwise operations for:

- Searching for an empty space
- Testing a bit
- Setting a bit
- Clearing a bit
- Displaying a bit vector (very helpful for debugging)

Functions (Board #4)

As stated in the overview, this task is only about file system and node creation and deletion so only the following functionality needs to be implemented using the following function names (parameters are not provided and need to be figured out on your own)

file_system_create

- Initializes memory
- Initializes bit vector
- Sets up the super block

file_create

- Sets up the FS_NODE for an empty file
- Sets a corresponding bit in the bit vector.

directory_create

- Sets up the FS_NODE for an empty directory.
- Sets up an empty INDEX node for that directory.
- Sets 2 corresponding bits in the bit vector.

file_delete

- Returns FS_NODE and all associated blocks to their original state (INDEX or DATA nodes included)
- Clear all these bits in the bit vector.

directory_delete

- Calls itself on all subdirectories and file_delete on all files within the directory.
- Then deletes the directory's index node and fs_node.
- Clears these bits in the bit vector.

Example (Board #5)

Here I show the filling of the content of memory and associated bit vector step by step for the following example file system:

- Root DIRECTORY
  - Short.txt (128 bytes)
  - Long.txt (300 bytes)
  - Logs DIRECTORY
    - SignIn.txt (200 bytes)
    - SignOut.txt (66 bytes)