At first mountain, set everything to zero.

Store offsets, not pointers.

Make a header at the beginning of the space to use it as “global” variables store space for a key word initialization for first mountain, a “pointer” to a linked list of free memory spaces, a “pointer” to the root directory, and a size\_t with the size of the entire mountain space (including the global variable header).

typedef size\_t \_\_myfs\_off\_t;

typedef enum \_\_myfs\_inode\_type\_enum\_t {

directory,

reg\_file;

} \_\_myfs\_inode\_type\_t;

typedef struct \_\_myfs\_file\_block\_struct\_t {

size\_t size;

size\_t allocated;

\_\_myfs\_off\_t next;

\_\_myfs\_off\_t data;

} \_\_myfs\_file\_block\_t;

typedef enum \_\_myfs\_inode\_file\_struct\_t {

size\_t size;

\_\_myfs\_off\_t first\_block;

} \_\_myfs\_inode\_file\_t;

typedef enum \_\_myfs\_inode\_directory\_struct\_t {

size\_t number\_children;

\_\_myfs\_off\_t children;

} \_\_myfs\_inode\_directory\_t;

typedef struct \_\_myfs\_inode\_struct\_t {

char name[256];

struct timespec Times[2];

union { //Is either one or the other

\_\_myfs\_inode\_file\_t file;

\_\_myfs\_inode\_directory\_t directory;

} value;

} \_\_myfs\_inode\_t;

Helpers {

handler -> check if remountain or first mountain.

off\_to\_pointer -> Given the initial point address and an offset, add them together to make a pointer.

path\_resolver -> given a string, tokenize it and call another helper function to iterate the tokens and return the file/directory for the final result. If it fails, return NULL.

If path, handle, or the first character of the path is not slash we return NULL.

While resolvind the patch, make sure that we are in a directory (curr->type != DIRECTORY).

The . directory means myself, the .. directory means the previous directory.

}

Getattr: (get attributes of file/directory in path)

Path -> string separated with slashes (use it to find the identity to go to).

Fill in the arguments

Readdir: (MAKE THIS THE LAST ONE)

Check that we are on a directory.

Return the number of children that the directory have, (exclude . and ..).

Allocate a pointer to an array of pointers to characters.

Fills in all the names on the array.

(You can use strdup to duplicate the directory string names)

Mknod: (Make a file)

Cut the path into a parent in a base (build absolute path for the file)

If the parent is not a directory, we get an error because files cannot have files.

If the file already exist, we don’t create it and set an error.

If the name is longer than the allow (255), set errnoptr.

If the filename have an ‘/’, errnoptr = EPERM.

THEN, we create the file. We check that if the file is the first file, we create the child linked list to files.

Unlink: (destroys a files)

If we are trying to destroy a directory we set errnoptr = EISDIR.

Check that the filename given is less than 256.

Check that the filename do not have ‘/’.

Delete the files, update children number, time of last modification, etc.

Rmdir: (destroys a directory, IT MUST BE EMPTY!)

Check that they don’t want to delete ‘.’ or ‘..’

If the directory have childrens we set errnoptr ENOTEMPTY.

Check name size and if it contains ‘/’.

Mkdir: (make a directory inside another directory)

Rename: ()

If a directory is being rename, update all absolute path for files.

Truncate: (Truncates a file to a certain length)

If it is not a file, error.

Check that the space wanted is positive.

If the new size if the old size, do nothing but you MUST update time of last modification.

If less than needed, we lost the other information.

If more than what we have, we look for the extra space and set everything to 0’s.

(you can use memset to fill in the zeros).

Open: (Check if the thing is a file)

Read: (Read a certain amount of bytes in a file given an offset)

Check that it must be a file.

Check that the offset is positive and within the range of the file end point.

If the user is asking more than what you have you only return as much as you can if any.

(You can use memcpy)

Write: ()

Check that it must be a file.

You may need more space if it is appended.

If the write happens at the middle everything must be shifted.

(You can use your own truncate for the total space needed and then you start the shift or append).

Utimens: (set the modification times of directories/files)

Statfs: (This talks about the entire filesystem)

A block can be of any size, up to you.