At the 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 keyword initialization for the 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 re-mountain 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 result. If it fails, return NULL.

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

While resolving the patch, ensure we are in a directory (curr->type != DIRECTORY).

The “.” directory means me, and 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 in a directory.

Return the number of children the directory has (exclude . and ..).

Allocate a pointer to an array of pointers to characters.

Fill 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 an absolute path for the file)

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

We don’t create the file and set an error if it already exists.

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

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

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

Unlink: (destroys a file)

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 does not have ‘/’.

Delete the files, update the children's 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 has children, we set error ENOTEMPTY.

Check the name size and if it contains ‘/.’

Mkdir: (make a directory inside another directory)

Rename: ()

If a directory is being renamed, update all absolute paths 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 is the old size, do nothing, but you MUST update the time of the 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 number 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 endpoint.

If the user asks for more than 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 writing happens in the middle, everything must be shifted.

(You can use your own truncate for the total space needed, and then 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.

Questions:

What is the root directory name?

“/”

Does the path name always start at the root directory?

Yes

How do we get the current time to set the files/directory?

Call char &ctime(const time\_t \*timer)

static void \_\_set\_current\_time(\_\_inode\_t \*node, int set\_mod) {

struct timespec ts;

If (node == NULL) return;

If (clock\_gettime(CLOCK\_REALTIME, &ts) == 0) {

node->times[0] = ts

If (set\_mod) {

node->times[1] = ts;

}

}