Introduction to FUSE

FUSE is used for taking a user defined, virtual file system, such as you have (eventually) implemented at the end of Task 2, and making it recognized as well as interactive with the real operating system.

In the tutorial, there is a very simple file system containing one file.

The file's name and content is represented by the following code:

```
static const char *hello_str = "Hello World!\n";
static const char *hello_path = "/hello";
```

This represents that the file system contains one file, accessible through the path `/hello.txt`, containing the content:

`Hello World!`

FUSE allows the user to define a list of operations that they wish to have the OS use to interact with the virtual file system.

This list is defined in the tutorial as:

```
static struct fuse_operations hello_oper = {
 .getattr = hello_getattr,
 .readdir = hello_readdir,
 .open = hello_open,
 .read = hello_read,
};
```

This list contains several function pointers each representing our implementation of an operation allowing for interaction between the file system and the OS.

The tutorial provides much more detailed information than I will here.

You should read the tutorial in order to gain an understanding of what each function is for and how it is to be implemented.

One important thing to note is that each of these of these functions have a list of parameters as well as a return type determined by the FUSE library, this ensures that it takes in the information that will be provided by the operating system, and gives the operating system back what it expects.

You should therefore also try to understand what each parameter is for and what the return type represents.

Task 3:

You are not required to do any coding for Task 3. All of the code is provided to you at the end of the tutorial. You can either read the tutorial at this time or move onto the next task after compiling the code and running it.

You should provide a typescript showing you list the content of the "fake" directory when running the program using the `ls` command.

You should then output the content of the hello.txt file using the `cat` command.

This typescript is worth 100% of the points for this very easy task, please ensure it is readable or you will lose 2 points out of the available 10.

Task 4:

Starting from the code provided in the FUSE tutorial, extend the hello filesystem to contain the following 4 files:

- File 1. `Hello.txt` : Same as in the tutorial.
- File 2. `Goodbye.txt` : Just a second normal file.
- File 3. `Secret.txt` : "You can't read this".
    - Should not be able to be read.
    - Within the open() function, you have to check for proper permissions.
- File 4. `Random.txt` :
    - Every time you try and cat random.txt, it will generate a random length and message to replace the existing content.

The bash file provided in the lab description gives you a way to test the file system.

Task 5:

- Integrating your virtual file system operations with FUSE.
- Turn your file system outputs into FUSE outputs essentially.
- Can take in normal inputs or FUSE inputs.
    - It's better to translate the inputs and outputs using essentially a wrapper class.

- Common problems:
  - Access errors from mode_t
  - General confusion and panic

The single_efs.c example linked in the task description gives you a more complicated example to work with, but still for a single file.

Note: You don't need to implement all the operations contained in the single_efs.c example.