Assignment 2

Nicholas Alexeev, Edward Hazelton (contributed little)

July 2020

1 Introduction

2 Design

The class file system uses blocks as its basic unit of data store. Information about the blocks is stored in a file allocation table in the beginning of the file system, right after the magic number 0x00000005c1f16546. Files and directories larger then a block (4096 bytes) are stored as a singly linked list of blocks. A blocks used capacity and next block is stored in the file allocation table as the used_size and next_block fields.

2.0.1 File System layout

A directory listing is stored in the same manner as a regular file, as a linked list of blocks. A directory listing contains an array of myfs_dir_entry_struct_t structs. The myfs_dir_entry structs contains all of the metadata for the file or directory linked. The myfs_dir_entry_struct contains the index of the block that the file data resides in. The root directory is located in the 0th block.

2.0.2 Algorithm for loading files

In order to locate a file's data a path is first split on the character '/'. Next the root directory is loaded and searched for the first element in the split path string. The next folder is then loaded into memory and the process is repeated until the requested file's __myfs_dir_entry is found.

2.0.3 Algorithm for Allocating and Freeing Blocks

A free block is located by searching through the file allocation table in order from lowest index to highest index. For each element the is_used flag is checked. If it is zero then the block is marked as allocated and the is_used flag is set to 1. Pseudo code is shown below.

```
for fat in fat_table:
  if(fat.is_used==0):
    fat.is_used=1
    fat.used_size=0
    return
```

In order to free a block, the block's is_used flag is set to zero and all of its children's is_used flags are set through zero. Pseudo code is shown below.

```
while True:
  fat.is_used=0
  if(fat.next_block!=0):
    fat = fat.next_block=0
  else:
    return
```

3 Difficulties

I had a lot of difficulty debugging my file system and making sure that it does not corrupt itself. In particular I had trouble with read and write as both have to be implemented correctly in order for the correct results to be seen. If one of the functions is not correct then it is very difficult to tell which one worked. In order to implemented write and read I wrote write to be very inefficient but simple so that I knew that it worked. I then implemented read to be more efficient. Once I could see that files were not corrupt I then rewrote write to be more efficient.

4 Testing

I tested the code by attempting to use the file system in a normal way. When some strange behavior was found it was noted and it was duplicated with simpler actions in order to find out exactly what broke.