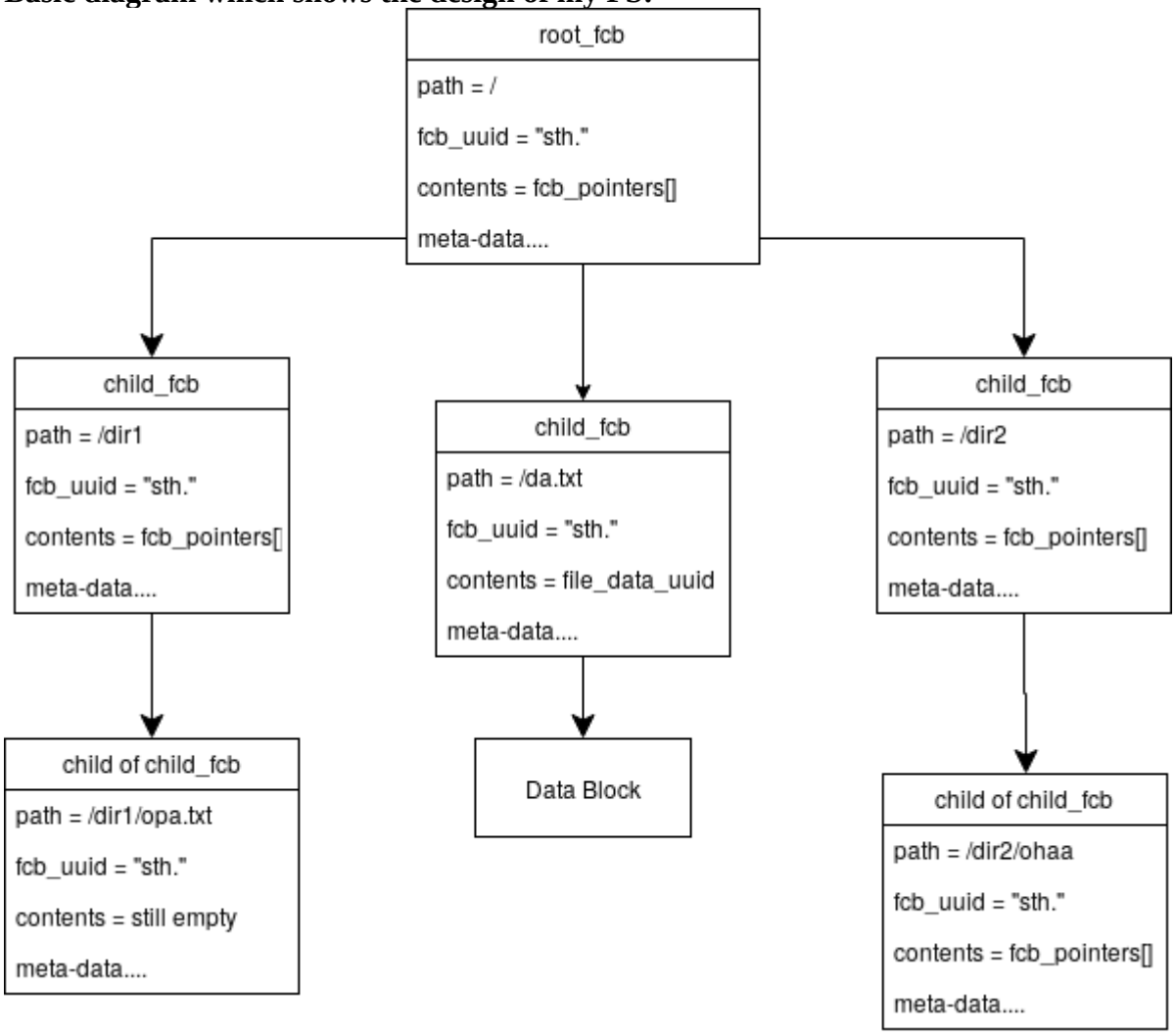
DESIGN & IMPLEMENTATION

FS Structure & important decisions

First I will describe the overall structure of my FS design.
I chose to support variable sized files and accept up to 255 bytes file path size as this is common amongst most popular file systems. For the purpose of satisfying the basic specification I impose a simplification limit of 200 files per folder, which can be extended to variable sized folders. I think that 200 files can satisfy basic needs for the practical. In my fcb, apart from the mode metadata, path and uuid of the fcb, I store a union which is either a single uuid_t, which points to the data of a regular file or an array of uuids which point to file control blocks in the database for nested children of a directory. I have added access time variable to the fcb to enhance the usability my file system. When displaying file size I have decided to display the number of bytes in the data block of that file, while when displaying directory size I show the accumulated sum of all file control blocks + sizes of file data blocks inside in order to show the user the full size. I have also decided to update access times, not only modification and change times of dirs/files. Modification times for directories are only updated for immediate parent, not nesting further back.
Basic diagram which shows the design of my FS:



Functionality

I have thoroughly logged my implementation with helpful information, so I have commented less to not overwhelm the reader. This extended logging helped me debug my implementation and at the

same time is useful for a reader to understand what exactly is happening during the execution of the fuse file system.

Helper functions

myfcb fetch_uuid_data(uuid_t data_id) – given a uuid fetches its data from the database.

myfcb find_fcb_from_path(path) – given a path tries to find the uuid with that path in the database. This is done by tokenizing the path, starting from the root and searching each directory in the hierarchy for the target. Here, as always if my target result is the root I pass the cached root fcb to keep track of root contents. If target is not found its parent is returned.

myfcb find_parent_from_child_path(path) – fetches parent fcb given one of its children's path. Basically, it strips the last level (part of path) to achieve that. This is used to fetch parent when having to apply update insertion/deletion of child or data block if file.

void print_uuid(uuid_t uuid) – simply unparses the uuid and puts it into a string buffer and prints it. Used for logging.

void update_root_in_db & update_parent_in_db (myfcb target_fcb, int written, bool isDeletion) - These two functions are used to apply updates to parents coming from their children. update_parent_in_db has one more parameter (its fcb), because root is cached and we don't need to pass its fcb. Target_fcb is the child that the update is coming from, written is num of bytes written or 0 if making new file/dir. A boolean variable is used to differentiate if parent size is to be increased or decreased. When update is not in the root folder we loop all the way down to root to update the size of every parent.

Fuse functions

myfs_getattr – simply searches for the given path in the file system, if it finds it, it fills the stat buffer with appropriate meta data.

myfs_readdir – reads the contents of a given directory path. It first finds the directory and then goes through its contents, while flushing them with filler. Access time is updated appropriately. I have two options listing root or listing any other dir.

myfs_read – Find fcb of file to read, try to fetch its data block from db, if successful adjust number bytes to read and flush them in the given buffer. Modify access time of file.

myfs_create – first check if file path is not too long. Then assign meta data and generate uuid for store. Finally, store child in db and update parents.

myfs_utime – Simply fetch mod time and act time from ubuf and store file fcb.

myfs_write – First fetch fcb of file from db. Then calculate the new data block size and fetch the data block into a buffer or generate a new uuid for the data block if it is empty. After that I write the the new data into the data_block buffer and store afterwards. Then I simply update file fcb and parents size. Given more time I would have shrinked the end of this function (the update of parents) and probably added it as a condition to the aforementioned update methods.

myfs_truncate – First fetch fcb of file from db, calculate current data size, try fetch uuid of data block from DB. Then either get everything from store and store extended block back to db or fetch

shrink size and store. After that we again gave these long updates, which could be made more modular. And update mtime at the end.

myfs_chmod & myfs_chown – Simply find fcb and update metadata from parameters, also update ctime at the end.

myfs_mkdir – First check if path is not longer than maximum path. Set contents of fcb appropriately and store the new directory fcb, and finally update parents.

myfs_unlink – Fetches fcb from database, calculates data size and check if file exist. Then if there is data block delete it first, then delete fcb of file and update parents.

myfs_rmdir – Simply fetches dir to delete from db, checks if it is empty, deletes the fcb and updates parents.

myfs_rename (extension) – First checks if new path is less than maximum path. Then finds the fcb of file/dir to move and calculates its size. In file block pointers I always store sizeof(myfcb) + data size so I need to substract myfcb size from total size to get the size of the data. Afterwards if it is a regular file I save its data block in a buffer, because I would need to copy it after that. Next, I delete dir or file in current place and move it to the new one by invoking my already implemented fuse functions.

TESTING

For the purposes of testing I have used my extended log file and using the file explorer and the command line to use my fuse functions. I have put my file system under various scenarios to test its functionality by mixing multiple calls of functions. Bellow are some of the screen shots that I captured to depict the process. Note that some of the tests are with different maximum number of files sizes, so sizes of directories between some tests differ.

Nesting multiple directories

```
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ ./myfs /cs/scratch/nd33/mnt
init fs
init store: root object was not found
init fs: writing root fcb
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ mkdir /cs/scratch/nd33/mnt/op
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ mkdir /cs/scratch/nd33/mnt/op/2
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ mkdir /cs/scratch/nd33/mnt/op/2
/3
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ ls -all /cs/scratch/nd33/mnt
total 5
drwxrwxr-- 2 nd33 nd33 1600 Nov 29 12:58 .
drwx------ 3 nd33 nd33 4096 Nov 29 12:44 ..
drwxrwxr-- 1 nd33 nd33 1200 Nov 29 12:59 op
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ mkdir /cs/scratch/nd33/mnt/oppa
a
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ mkdir /cs/scratch/nd33/mnt/oppa
ads
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ ls -all /cs/scratch/nd33/mnt
total 6
drwxrwxr-- 2 nd33 nd33 2400 Nov 29 12:59 .
drwx------ 3 nd33 nd33 4096 Nov 29 12:44 ..
drwxrwxr-- 1 nd33 nd33 1200 Nov 29 12:59 op
drwxrwxr-- 1 nd33 nd33  400 Nov 29 12:59 oppaa
drwxrwxr-- 1 nd33 nd33  400 Nov 29 12:59 oppaads
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ rmdir /cs/scratch/nd33/mnt/oppa
a
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ ls -all /cs/scratch/nd33/mnt
total 6
drwxrwxr-- 2 nd33 nd33 2000 Nov 29 12:59 .
drwx------ 3 nd33 nd33 4096 Nov 29 12:44 ..
drwxrwxr-- 1 nd33 nd33 1200 Nov 29 12:59 op
drwxrwxr-- 1 nd33 nd33  400 Nov 29 12:59 oppaads
```

Truncating files

```
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ ./myfs /cs/scratch/nd33/mnt
init fs
init store: root object was not found
init fs: writing root fcb
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ echo "odataaaaaaaa" > /cs/scrat
ch/nd33/mnt/da.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ ls -all /cs/scratch/nd33/mnt
total 11
drwxrwxr-- 2 nd33 nd33 7053 Nov 29 19:13 .
drwx------ 3 nd33 nd33 4096 Nov 29 16:23 ..
-rw-r--r-- 1 nd33 nd33   13 Nov 29 19:13 da.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ truncate -s 2 /cs/scratch/nd33/
mnt/da.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ ls -all /cs/scratch/nd33/mnt
total 11
drwxrwxr-- 2 nd33 nd33 7042 Nov 29 19:13 .
drwx------ 3 nd33 nd33 4096 Nov 29 16:23 ..
-rw-r--r-- 1 nd33 nd33    2 Nov 29 19:14 da.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ cat  /cs/scratch/nd33/mnt/da.tx
t
odpc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$
```

Writing & Reading Files

```
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ ./myfs /cs/scratch/nd33/mnt
init fs
init store: root object was not found
init fs: writing root fcb
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ echo "grmigjroawigjeoginjw\goir
wangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg" > /cs/scratch/nd33/mnt/ohaa/2.txt
bash: /cs/scratch/nd33/mnt/ohaa/2.txt: No such file or directory
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ echo "grmigjroawigjeoginjw\goir
wangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg" > /cs/scratch/nd33/mnt/2.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ echo "grmigjroawigjeoginjw\goir
wangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg" >> /cs/scratch/nd33/mnt/2.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ echo "grmigjroawigjeoginjw\goir
wangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg" >> /cs/scratch/nd33/mnt/2.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ echo "grmigjroawigjeoginjw\goir
wangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg" >> /cs/scratch/nd33/mnt/2.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ echo "grmigjroawigjeoginjw\goir
wangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg" >> /cs/scratch/nd33/mnt/2.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ echo "grmigjroawigjeoginjw\goir
wangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg" >> /cs/scratch/nd33/mnt/2.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ echo "grmigjroawigjeoginjw\goir
wangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg" >> /cs/scratch/nd33/mnt/2.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ echo "grmigjroawigjeoginjw\goir
wangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg" >> /cs/scratch/nd33/mnt/2.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ echo "grmigjroawigjeoginjw\goir
wangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg" >> /cs/scratch/nd33/mnt/2.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ echo "grmigjroawigjeoginjw\goir
wangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg" >> /cs/scratch/nd33/mnt/2.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ echo "grmigjroawigjeoginjw\goir
wangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg" >> /cs/scratch/nd33/mnt/2.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ echo "grmigjroawigjeoginjw\goir
wangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg" >> /cs/scratch/nd33/mnt/2.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ cat /cs/scratch/nd33/mnt/2.txt
grmigjroawigjeoginjw\goirwangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg
grmigjroawigjeoginjw\goirwangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg
grmigjroawigjeoginjw\goirwangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg
grmigjroawigjeoginjw\goirwangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg
grmigjroawigjeoginjw\goirwangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg
grmigjroawigjeoginjw\goirwangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg
grmigjroawigjeoginjw\goirwangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg
grmigjroawigjeoginjw\goirwangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg
grmigjroawigjeoginjw\goirwangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg
grmigjroawigjeoginjw\goirwangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg
grmigjroawigjeoginjw\goirwangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg
grmigjroawigjeoginjw\goirwangoiwrgjriwgjreogijwrpgegdpirnbhprmgprwogfmkweofmjawzpogfpireg
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$
```

Rmdir & Unlink & ReadDir

```
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ ./myfs /cs/scratch/nd33/mnt
init fs
init store: root object was not found
init fs: writing root fcb
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ touch /cs/scratch/nd33/mnt/2.tx
t
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ echo "da" >  /cs/scratch/nd33/m
nt/23.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ mkdir /cs/scratch/nd33/mnt/2
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ mkdir /cs/scratch/nd33/mnt/2/23
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ touch /cs/scratch/nd33/mnt/2/te
xt.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ ls -all /cs/scratch/nd33/mnt/
total 18
drwxrwxr-- 2 nd33 nd33 21123 Nov 29 20:07 .
drwx------ 3 nd33 nd33  4096 Nov 29 16:23 ..
drwxr-xr-x 1 nd33 nd33 10560 Nov 29 20:07 2
-rw-r--r-- 1 nd33 nd33     3 Nov 29 20:06 23.txt
-rw-r--r-- 1 nd33 nd33     0 Nov 29 20:06 2.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ ls -all /cs/scratch/nd33/mnt/2
total 14
drwxr-xr-x 1 nd33 nd33 10560 Nov 29 20:07 .
drwxrwxr-- 2 nd33 nd33 21123 Nov 29 20:07 ..
drwxr-xr-x 1 nd33 nd33  3520 Nov 29 20:07 23
-rw-r--r-- 1 nd33 nd33     0 Nov 29 20:07 text.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ rmdir /cs/scratch/nd33/mnt/2
rmdir: failed to remove '/cs/scratch/nd33/mnt/2': Directory not empty
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ rmdir /cs/scratch/nd33/mnt/2/23
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ rm /cs/scratch/nd33/mnt/2/text.
txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ ls -all /cs/scratch/nd33/mnt/2
total 7
drwxr-xr-x 1 nd33 nd33  3520 Nov 29 20:09 .
drwxrwxr-- 2 nd33 nd33 14083 Nov 29 20:07 ..
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ rmdir /cs/scratch/nd33/mnt/2
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$ ls -all /cs/scratch/nd33/mnt
total 15
drwxrwxr-- 2 nd33 nd33 10563 Nov 29 20:07 .
drwx------ 3 nd33 nd33  4096 Nov 29 16:23 ..
-rw-r--r-- 1 nd33 nd33     3 Nov 29 20:06 23.txt
-rw-r--r-- 1 nd33 nd33     0 Nov 29 20:06 2.txt
pc3-067-l:~/Documents/CS3104/FilesystemPractical/code_no_malloc nd33$
```

EVALUATION & CONCLUSION

I think that I have made a reasonable implementation of the basic requirements of this practical, as well as having implemented rename and keeping track of change time and access time. However, I could have made the code more modular and reduced the redundancy, because I have a couple of long pieces of code, which are quite similar, but did not have the time to shrink them. Moreover, if I had more time I would have implemented variable sized directories by having a Linked List or some sort of Array List in my contents union, instead of just using an array of fixed size. Also the find_fcb_from_path() function could have been simplified by using recursion instead of nesting a while loop inside a for.

References

https://unqlite.org/api_intro.html

https://www.cs.hmc.edu/~geoff/classes/hmc.cs135.201001/homework/fuse/fuse_doc.html#readdir-details