EEL 5737 Principles of Computer System Design

Homework #2 - Solutions

PART 2

Table of Contents

Table of Figures

1. Design for Hierarchical file system

1.1. Introduction

This design describes how the flat in-memory file system implemented in `memory.py` is extended to a hierarchical in-memory File system.

1.2. High Level Design

To understand how the memory.py is extended, let us first consider the original flat in-memory file system and see how contents are stored in a dictionary.

self.files[]

/ (root)	Metadata {st_mode, : : st_nlink}	Bindings for elements in Metadata
/dir1	Metadata {st_mode, : : st_nlink}	Bindings for elements in Metadata
/hello.txt	Metadata {st_mode, : : st_nlink}	Bindings for elements in Metadata
: :	: :	: :
/dirn	Metadata {st_mode, : : st_nlink}	Bindings for elements in Metadata

Figure 2 Flat in-memory FS Dictionary of metadata

self.data[]

/hello.txt	data
/python.py	data
/exampels.txt	data
: :	: :
/filelist.txt	data

Figure 1 Flat in-memory FS - Dictionary for file data

Now, let us have a look at the designed hierarchical in-memory file system as shown in Figure 3

Every directory has a dictionary associated with it and it contains an element ‘Sub’ in addition to metadata. The key ‘Sub’ is bound to another dictionary which holds dictionaries of children directories.

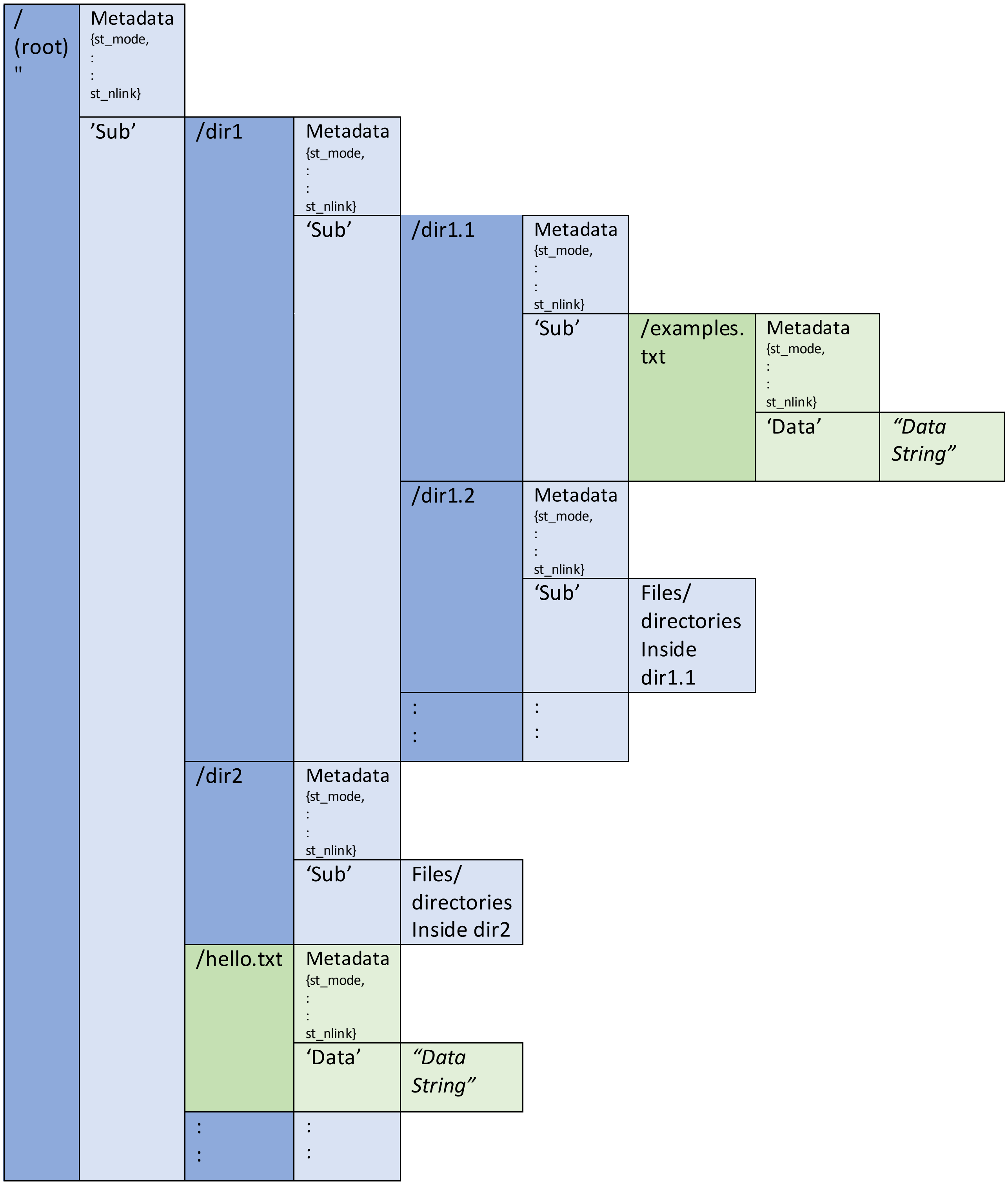
/ (root) ''
Metadata {st_mode, : : st_nlink}
'Sub'
/dir1
Metadata {st_mode, : : st_nlink}
'Sub'
/dir1.1
Metadata {st_mode, : : st_nlink}
'Sub'
/examples.txt
Metadata {st_mode, : : st_nlink}
'Data'
"Data String"
/dir1.2
Metadata {st_mode, : : st_nlink}
'Sub'
Files/ directories Inside dir1.1
/dir2
Metadata {st_mode, : : st_nlink}
'Sub'
Files/ directories Inside dir2
/hello.txt
Metadata {st_mode, : : st_nlink}
'Data'
"Data String"

Figure 3 Hierarchical in-memory FS - Dictionary for metadata and file data storage

1.3. Low level design

In order to implement a hierarchical file system as shown above, two new functions are added in memory.py of which,

- write_metdat() is used to modify existing metadata or create a dictionary for a new directory/file and write metadata to it.
- read_metdat() is used to read metadata or get dictionary associated with a directory or file.

For both these functions, traversing to the final location indicated by the path from root directory is necessary to perform any operation. Traversing for the path is done as follows:

- Root dictionary, Path (and metadata for write) are passed as arguments to these functions.
- The path is split and elements of path are extracted into a list.
- Using the elements of the list one at a time, the next dictionary in the path is extracted, starting form root dictionary.
- Once, the final dictionary (the target) is reached, we perform the requested operation there (read/write).

Each instance of write to self.files[path] in memory.py is replaced by a function call to write_metdat() with root dictionary, path and Metadata to be written as arguments.

Each instance of read from self.files[path] in memory.py is replaced by a function call to read_metdat() with root dictionary and path as arguments.

2. Design for Client/Server implementation

Yet to be done.

3. Test Specification and Report

3.1. Testing hierarchical file system

For testing hierarchical file system, the modified memory.py file i.e., hierarchicalFS.py is mounted onto fusemount on linux machine.
Then a series of commands are given and results are compared against expected results.

```
jai@ubuntu:~/fusepy/fusemount$ ls
jai@ubuntu:~/fusepy/fusemount$ mkdir dir1
jai@ubuntu:~/fusepy/fusemount$ ls
dir1

jai@ubuntu:~/fusepy/fusemount$ mkdir dir1/dir1.1
jai@ubuntu:~/fusepy/fusemount$ ls
dir1
jai@ubuntu:~/fusepy/fusemount$ mkdir dir2
```

```
jai@ubuntu:~/fusepy/fusemount$ mkdir dir3
jai@ubuntu:~/fusepy/fusemount$ ls
dir1  dir2  dir3

jai@ubuntu:~/fusepy/fusemount$ cd dir1
jai@ubuntu:~/fusepy/fusemount/dir1$ mkdir dir1.2
jai@ubuntu:~/fusepy/fusemount/dir1$ mkdir dir1.3
jai@ubuntu:~/fusepy/fusemount/dir1$ ls
dir1.1  dir1.2  dir1.3

jai@ubuntu:~/fusepy/fusemount/dir1$ cd ..
jai@ubuntu:~/fusepy/fusemount$ echo "hello">hello.txt
jai@ubuntu:~/fusepy/fusemount$ ls
dir1  dir2  dir3  hello.txt

jai@ubuntu:~/fusepy/fusemount$ cat hello.txt
hello

jai@ubuntu:~/fusepy/fusemount$ echo "examples of memory">dir1/dir1.1/examples.txt
jai@ubuntu:~/fusepy/fusemount$ cd dir1/dir1.1
jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ ls
examples.txt

jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ mkdir dir1.1.1
jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ ls
dir1.1.1  examples.txt

jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ cat examples.txt
examples of memory

jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ mv examples.txt memorytypes.txt
jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ ls
dir1.1.1  memorytypes.txt

jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ cat memorytypes.txt
examples of memory

jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ ls
dir1.1.1  memorytypes.txt

jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ rm memorytypes.txt
jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ ls
dir1.1.1

jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ cd ..
jai@ubuntu:~/fusepy/fusemount/dir1$ cd ..
jai@ubuntu:~/fusepy/fusemount$ rmdir dir1/dir1.1/dir1.1.1
jai@ubuntu:~/fusepy/fusemount$ cd dir1/dir1.1
jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ ls
jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$
```