

# ENGR-E-516-Engineering-Cloud-Computing

## Assignment 3 – File Systems Using Cloud Storage

Param Rajeev Patil ([papatil@iu.edu](mailto:papatil@iu.edu))

### Overview

In this assignment, I have created a file system that utilizes cloud object storage through the FUSE (File systems in User Space) interface. The assignment encompasses the implementation of basic file operations and directory operations, focusing on correctness and offering the possibility to evaluate performance by comparing operation latency to native file systems. Basic functionalities like mount, create, open, read, write, close are implemented.

### Design Details

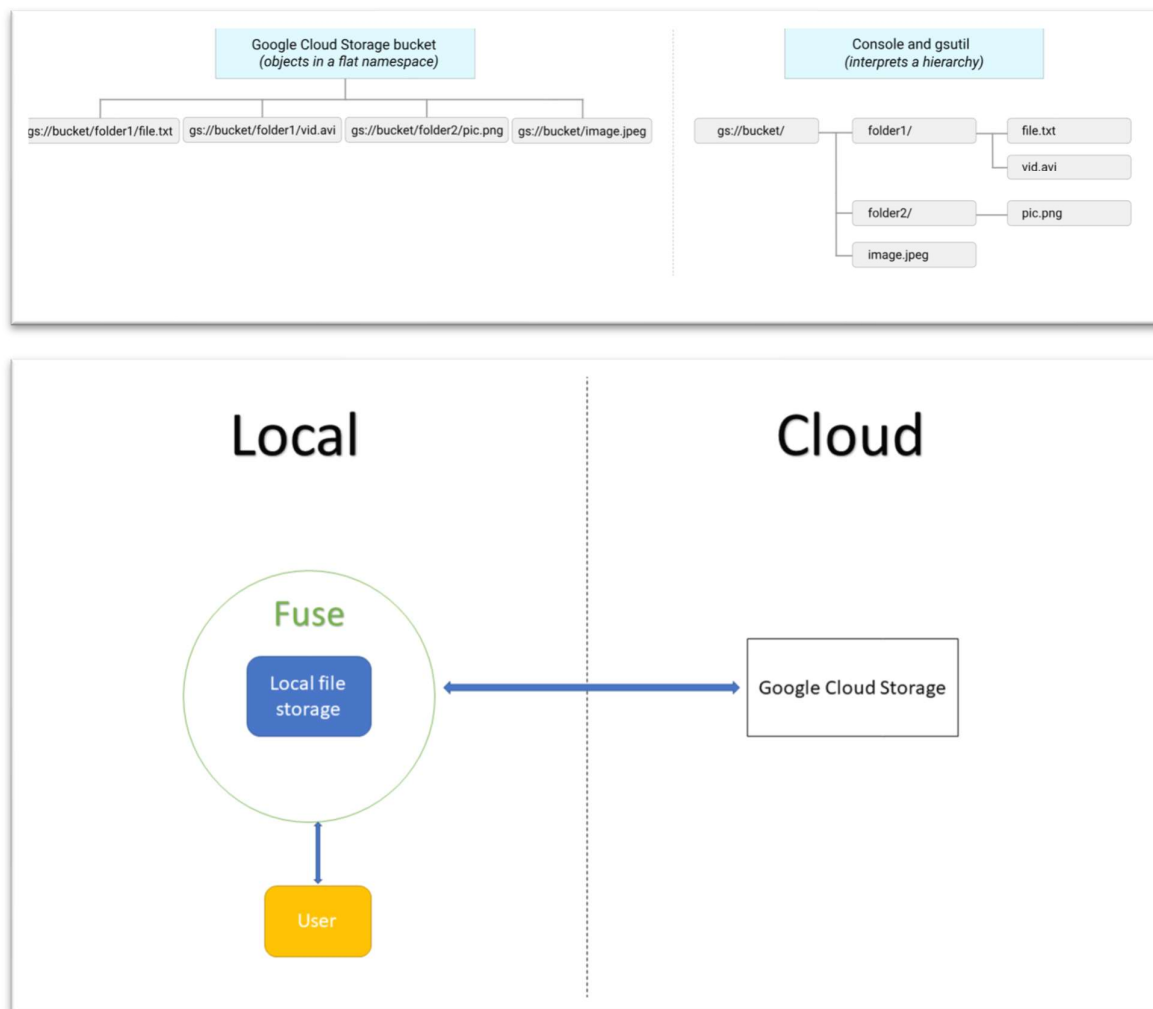


Figure 0-1 System Design

### FUSE Implementation:

I have used fusepy library for python to implement FUSE file system for Google Cloud storage. I have used google cloud python client in my program to connect to cloud storage. We had

to implement mount create, open, read, write, close, mkdir, opendir, readdir, rmdir functions using FUSE.

1. **Mount:**

- **Linux Command:** **mount**
- **FUSE Function:** The **mount** operation is not explicitly implemented in FUSE; it is a standard Linux command used to mount file systems, including FUSE-based file systems, onto a mount point directory. It establishes a connection between the FUSE file system and the specified mount point.

2. **Create:**

- **Linux Command:** **touch** (for creating an empty file)
- **FUSE Function:** The **create** operation is used to create a new file within the FUSE file system. This operation typically maps to the **open** system call with the **O\_CREAT** flag.

3. **Open:**

- **Linux Command:** **open**
- **FUSE Function:** The **open** operation is triggered when a file is opened for reading or writing. It usually involves checking permissions, locking, and preparing the file for subsequent read or write operations.

4. **Read:**

- **Linux Command:** **cat**, **less**, or any command that reads file content
- **FUSE Function:** The **read** operation is called when data from a file is requested. It reads the specified portion of the file and returns it to the user.

5. **Write:**

- **Linux Command:** **echo**, **dd**, or any command that writes data to a file
- **FUSE Function:** The **write** operation is invoked when data is written to a file. It takes the data from the user and writes it to the file.

6. **Close:**

- **Linux Command:** **close**
- **FUSE Function:** The **close** operation is called when a file is closed after reading or writing. It typically involves finalizing any pending write operations and releasing resources.

7. **Mkdir:**

- **Linux Command:** **mkdir**
- **FUSE Function:** The **mkdir** operation is used to create a new directory within the FUSE file system.

8. **Opendir:**

- **Linux Command:** **opendir** is not a direct Linux command but rather a function used within programs and applications to open a directory.
- **FUSE Function:** The **opendir** operation is not a direct Linux command but is an operation called when a program or application attempts to open a directory within the FUSE file system. It typically prepares the directory for listing its contents.

#### 9. **Readdir:**

- **Linux Command:** **ls** or any command that lists directory contents
- **FUSE Function:** The **readdir** operation is triggered when a directory's contents are requested. It returns a list of files and subdirectories within the specified directory.

#### 10. **Rmdir:**

- **Linux Command:** **rmdir**
- **FUSE Function:** The **rmdir** operation is used to remove (delete) a directory from the FUSE file system. It typically involves checking for directory emptiness and permissions before deletion.

#### 11. **Flush:**

- **FUSE Function:** The **flush** operation is called when a file is flushed to ensure any buffered data is written to the underlying storage.
- **Example:** Flushing data to a file using the **sync** command to ensure data is written to the storage device.

#### 12. **Fsync:**

- **FUSE Function:** The **fsync** operation is used to synchronize a file's state, ensuring any changes are persisted to storage.
- **Example:** Syncing a file to disk using the **sync** command to make sure data is written to the storage device.

#### 13. **Unlink (Delete):**

- **Linux Command:** **rm** or **unlink**
- **FUSE Function:** The **unlink** operation is invoked when a file is deleted. It removes the specified file from the file system.
- **Example:** Deleting a file using the **rm** or **unlink** command.

#### 14. **Rename:**

- **Linux Command:** **mv**
- **FUSE Function:** The **rename** operation is used to rename files or move them between directories within the FUSE file system.
- **Example:** Renaming a file using the **mv** command.

#### 15. **Link:**

- **Linux Command: In**
- **FUSE Function:** The **link** operation allows creating hard links to files within the FUSE file system. Hard links are multiple directory entries pointing to the same data.
- **Example:** Creating a hard link to a file using the **ln** command.

I have created a mount directory on my local system. The fuse is mounted on the same directory, whenever any of the above commands are run the FUSE handles the call instead of kernel and passes it to my program. So the call is then redirected inside my custom class CloudStorageFUSE.

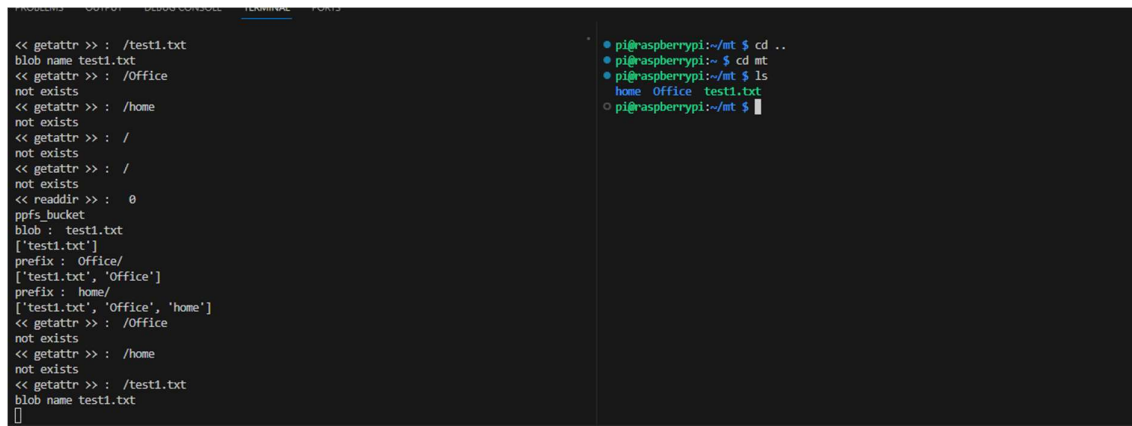
In the init function I have initialised the google cloud storage and fetched the bucket. There are few helpers function like full path and get object which gives me the blob from bucket. I have also set the environment variable for Google cloud storage so that anyone running this code access my storage.

## Testing

### Test Cases:

Check if LS command works:

The ls command works perfectly, along with cd.



```

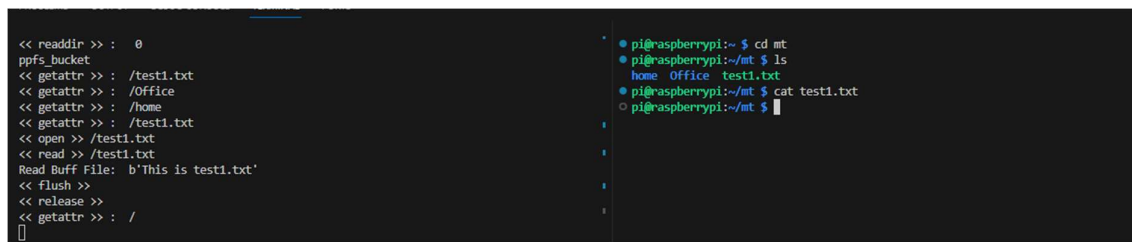
<< getattr >> : /test1.txt
blob name test1.txt
<< getattr >> : /Office
not exists
<< getattr >> : /home
not exists
<< getattr >> : /
not exists
<< getattr >> : /
not exists
<< readdir >> : 0
ppfs_bucket
blob : test1.txt
['test1.txt']
prefix : Office/
['test1.txt', 'Office']
prefix : home/
['test1.txt', 'Office', 'home']
<< getattr >> : /Office
not exists
<< getattr >> : /home
not exists
<< getattr >> : /test1.txt
blob name test1.txt
[]

pi@raspberrypi:~/mt $ cd ..
pi@raspberrypi:~/ $ cd mt
pi@raspberrypi:~/mt $ ls
home Office test1.txt
pi@raspberrypi:~/mt $

```

Check if Read works:

I am receiving the data from google cloud storage, but it is not getting printed out for user.



```

<< readdir >> : 0
ppfs_bucket
<< getattr >> : /test1.txt
<< getattr >> : /Office
<< getattr >> : /home
<< getattr >> : /test1.txt
<< open >> /test1.txt
<< read >> /test1.txt
Read Buff File: b'This is test1.txt'
<< flush >>
<< release >>
<< getattr >> : /
[]

pi@raspberrypi:~/ $ cd mt
pi@raspberrypi:~/mt $ ls
home Office test1.txt
pi@raspberrypi:~/mt $ cat test1.txt
pi@raspberrypi:~/mt $

```

Check if Mkdir works:

Similar to above function, the file is created on cloud but unable to reflect on local mount point.

## Performance Evaluation

Compared to local storage passthrough example, cloud FUSE is a bit slower as it takes time so send a HTTP request to get the data from buckets.

## Limitations and Future Improvements

The current implementation has some limitations:

- The FUSEPY library cannot handle simultaneous calls.
- Currently the system is accepting only text files.
- Does not support all function calls.

Future improvements and enhancements could include:

- This implementation can be improved with bugs fixing.
- The efficiency could be improved with better logic

## Conclusion

The FUSE implementation allows file systems to be built in user space, making it a flexible and powerful tool for creating custom file systems that interact with cloud storage. Key aspects of this include correctly handling various file system operations and ensuring that the semantics of each operation are accurately implemented, including error handling and arguments.

The FUSE agent runs on the local machine, where FS clients or programs also operate. The agent intercepts file system calls and invokes the corresponding cloud storage API operations, managing interactions with cloud storage efficiently and consistently.

## References

<https://github.com/memcached/memcached/wiki/Commands#standard-protocol>

<https://github.com/memcached/memcached/blob/master/doc/protocol.txt>

[\(317\) Memcache Fundamentals in Python | Python PyMemcache Tutorial - YouTube](#)

[Python client library | Google Cloud](#)

[About Cloud Storage buckets | Google Cloud](#)

## Process to run script:

- 1) Use an Ubuntu system
- 2) Run “pip install fusepy”
- 3) Run “pip3 install --upgrade google-cloud-storage”
- 4) The mt directory is mounted by default and you can perform operations in that directory.