International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi : https://doi.org/10.32628/CSEIT2174128

# PINΔLΞ : FUSE based Filesystem and Folder Syncer

**Parth Gor[1], Rayson D'sa[1], Harit Acharya[1], Mrs. Geetha S.[2]**

[1]Diploma Student, Department of Computer Engineering, Shri Bhagubhai Mafatlal Polytechnic, Mumbai, Maharashtra, India

[2]Lecturer, Department of Computer Engineering, Shri Bhagubhai Mafatlal Polytechnic, Mumbai, Maharashtra, India

**Article Info**
Volume 7, Issue 4
Page Number: 350-358
**Publication Issue :**
July-August-2021
**Article History**
Accepted : 15 Aug 2021
Published : 20 Aug 2021

**ABSTRACT**

In this digital world the data is organized by an entity called a file system. In desktop operating systems the file manager manages the local data. But with cloud technologies, reliance on online storage has increased since the past decade, which demands a hybrid file managing solution. The proposed file system using FUSE and application give the functionalities such as support for multiple cloud storages on local file manager, automated syncing of folders on different devices.

**Keywords:** Filesystem, FUSE, Cloud, Pindle, Sync, Gdrivefs, Onedrivefs.

## I. INTRODUCTION

With global lockdown, business continuity challenges, and the trend to work from home, COVID19 is forcing organizations to migrate to the cloud environment to deal with the crisis and promote daily data access, sharing and work collaboration of spatially disseminated teams. The proliferation of transitions to the cloud has further promoted the embracement of cloud storage solutions by enterprises to achieve business continuity, operational workloads, and data warehousing. Additionally, due to scalability, flexibility, and ubiquitous availability benefits of the cloud, quite a large no. of companies are migrating their delegations to the cloud, leading to a demand for services such as consulting, training, support and maintenance, integration and implementation is also increasing.

Though there's more and more dependency on the cloud, consumers are lacking to make full utilization of its rich packed features. The features we use are its ability to store our system configuration and the use of respective cloud APIs. The basic idea of using cloud storage is just storing, editing, and accessing it through the WEB. We gave some thought to "how can we make this process of cloud system be used more conveniently", so we developed a file system to solve this problem. There are many GDriveFS and OnedriveFS available in the marketplace, but this one's unique in the regards of, it is configurable by GUI and provides ease of use and configuration to the layman.

In this hybrid work culture i.e., alternate days at office and work from home the other day has made life on an employee difficult as they have to continue the same work, but for that they have to carry the data by pen drive or mail it or upload in cloud storages, but no cloud storages does provide every software as SaaS. Therefore, synced folders at office

and home can give huge relief of carrying data every day with automated syncs.

## II. BACKGROUND

FUSE (Filesystem in User space) is an attachment to user space programs for using custom file systems with the Linux kernel. There are two major components of the FUSE first is the fuse kernel and second, the libfuse user space library. For interfacing the FUSE kernel module, libfuse caters us with a citated execution for the same. The FUSE file system linked with libfuse is usually standalone system. Libfuse library provides the capability to unmount and mount the filesystem, read kernel requisitions, and respond back. libfuse provides two APIs: a synchronous API which is "high level", and an asynchronous "low-level" API. [3] In both cases, incoming requests from the kernel are passed to the main program via call-backs. When using advanced APIs, the call-back can use the filename and path instead of the inode, and the request processing completes when the call-back function returns. When using low-level APIs, the callback must be used with the inode, and a separate set of API functions must be used to send the response explicitly.

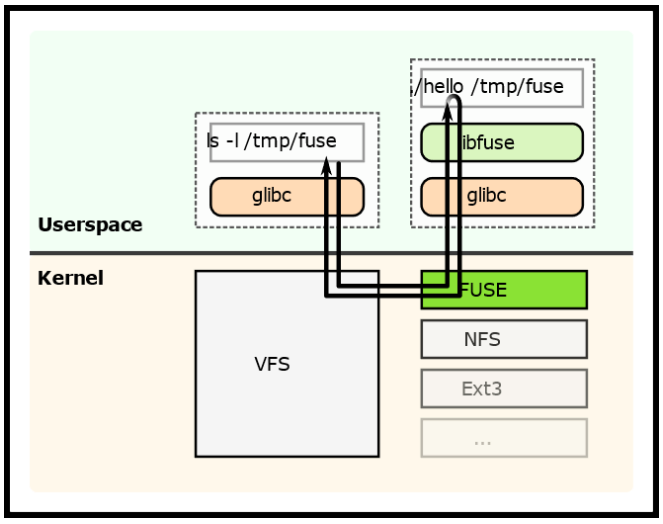


**Figure 1.** FUSE working diagram in Linux Kernel.

Figure 1. flowchart showing the working principle of FUSE [2]: The request list file (ls l / tmp / fuse) in the user space is redirected to FUSE by the kernel via VFS. FUSE then executes the registered handler program (./hello) and passes it the request (ls -l /tmp/fuse). The controller returns the response to FUSE, which then redirects it to the user space program that originally made the request.

## III. WORKING AND IMPLEMENTATION

### A. Mounter

Mounting a filesystem by making entries in fstab is not user-friendly, it may require some technical skills and if not configured properly it may prove hazardous. Complexity of mounting, the classical way, is quite cumbersome, so the mounter provides Graphical-User-Interface, hiding the under-lying procedure. Mounting filesystem by just providing the mail-id and the mount point. The first page displays the previously mounted records along with the accounts linked, if any. The Add button in the interface will prompt the new config window where the user will input the mail-id and mount point. Both are then verified; the format of the mail-id is verified, and the mount point is verified by checking whether the folder exists in the said path or doesn't plus it has to be empty. Clicking the OAuth setup will be done; accordingly, required permissions need to be granted by the user. After that a token or session file will be generated as a reference to sign-in the user later on whenever he/she desires a new mount point for the same account. The FUSE [FileSystem in User Space] kernel module mounts the G-Drive FS in the User Space and through the virtual FS, all the system calls are made without tampering with the kernel code. In the OneDrive and G-Drive the python file takes the mount point as an argument and passes it to FuseArgs. The OneDrive/G-Drive class inherits the Fuse class. Inside the OneDrive/GDrive class various functions are defined such as readdir, open, create, etc.

getattr() : It is an essential function for files and folders for assigning meta-data such as GID, UID, Size, last-modified, creation time, etc. Size of the file is fetched from the cloud and assigned here.

readdir() : Folders and files in the mounting directory say for example root will be read from cloud and as we browse through the mounted directory, all the files and folders will be shown using the FuseDir entry.

read() : the read function is used with a read proxy which is used to get a stream of bytes from the cloud using the HTTP GET method. When the end-user clicks on the file e.g. pdf, docx from explorer, FUSE calls this function , in that getattr is called which gets all the metadata related to the file. And one locally maintained data structure present in the .py file gives the file id and that gets passed to the read proxy which uses the base URL of API, Authorization URL and specifies the byte range to receive. After receiving a stream of bytes, the file is loaded with that data, and another GET request is sent to fetch the next chunk of bytes.

mkdir() : This function creates a new folder on the cloud after receiving the folder name and parent folder name.

rmdir(): After receiving the folder name rmdir removes the folder from the cloud. This removal is a forced removal, i.e., non-empty will also get deleted.

create() : In this function, the name of the file is received from FUSE, a temporary file is created and uploaded to cloud, and then renamed with the filename received.

write() :In the write function, file-ID is fetched from the data structure from the filename received from fuse, then that file is read from the cloud and stored into a temporary file. The modification done by the user is also overwritten in the temporary file and this file is uploaded to the file while the old file is deleted. Along with this write function, a truncate function is also called which updates the file size locally.

unlink() : It is used to delete the file. Based on the received file name, the file-ID is fetched, and the folder is deleted locally and from the cloud; also, the entry is automatically removed from the explorer.

All received mail-id and mounting points from different cloud platforms and multiple cloud accounts are stored in one JSON file which is used at boot time for mounting. Authenticated Token/Session file is also stored into the same folder and a log file for the user-account is generated in which all the operations performed while using the file system are logged. Mounting or un-mounting can be done from the Command line tool.

### B. Syncer

The next application is Syncer, as the name suggests what it does is syncs folders between two different PCs. Setting up it is easy; the user has to first download the package. The installation process is standard. Once having opened the application, the user needs to input his/her email u id through which the cloud account is adjoined, the application will validate the mail-id and will also detect the cloud provider, currently two leading cloud storage platforms are supported, G-Drive and OneDrive. Clicking next, the user will be redirected to the browser wherein the O-Auth and agreement to give permissions will occur. After that, both the systems among which the sync has to take place should be given unique usernames in the given field. Storage stats are given in the next page. The current synced folders are shown in the next page, using the edit button which will redirect to a new page where current directories are shown, and the user can select the folder for sync. The next phase in configuration is to map the folders to be synced. After pressing the apply button, the new configurations are uploaded to the cloud along with the content that needs to be synced. Once done, the folders which are to be synced are selected on both the systems, and the observer runs on both the devices from boot time(start-up) . Any event occurring under observed

folders will call respective functions. On-creating the event, the respective file will be uploaded to the cloud platform provided by the user and the event is cached in the configuration file and then uploaded. On-delete event, only the delete action is updated in the configuration file. Clicking 'Sync Now' displays, and thereby a confirmation that the changes are received. Henceforth both the folders are in sync. In the same window, the remove button is provided by which the user can discard the changes and the apply button will make changes to the destination. There is one connection-manager running behind which switches between online and offline service which continuously senses the network. Going from online to offline, stops the files from being uploaded, instead it is locally cached and whenever the connection resumes i.e. offline-to-online service it gets latest configuration from cloud and compares it with local configuration, if any deviation is found then locally applied changes are considered as the latest one and are applied again and it first clears the queue of pending requests.

## IV. RESULTS AND DISCUSSION

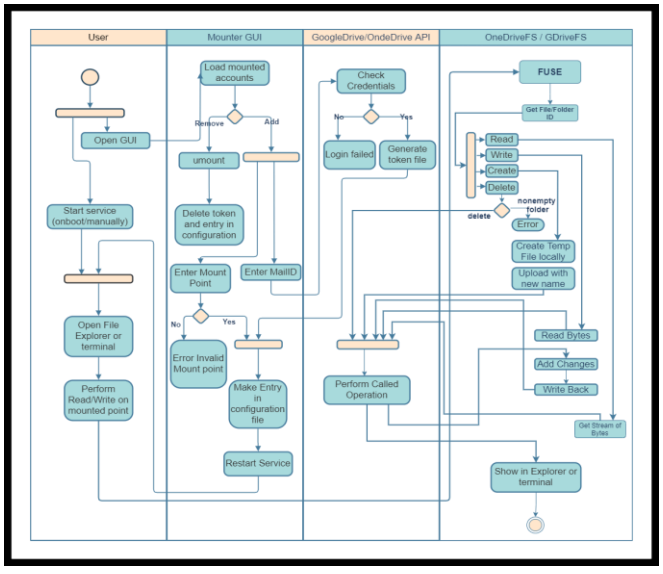


**Figure 2.** Activity diagram depicting the activities of cloud storage Mounter.

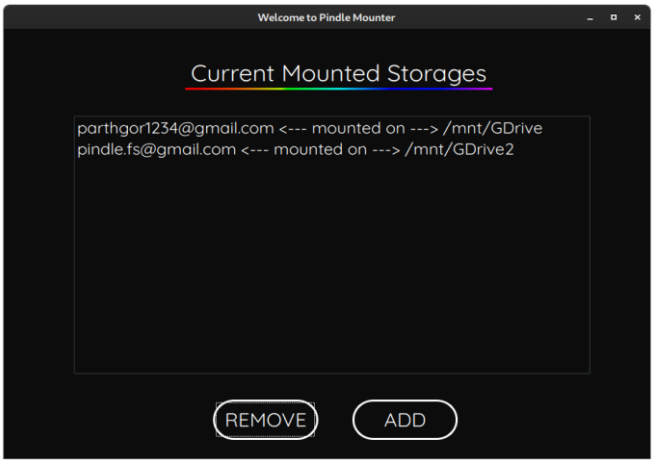


**Figure 3.** An edit window to add a new cloud storage account or remove an existing one.

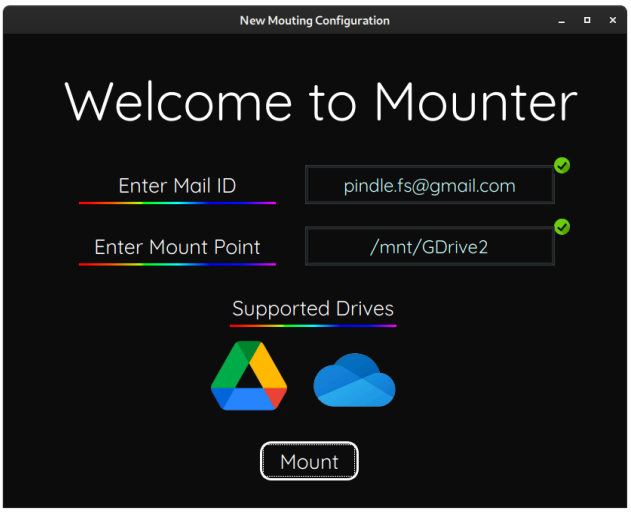


**Figure 4.** Window to add the cloud account to associate with the desired mount point in the system.

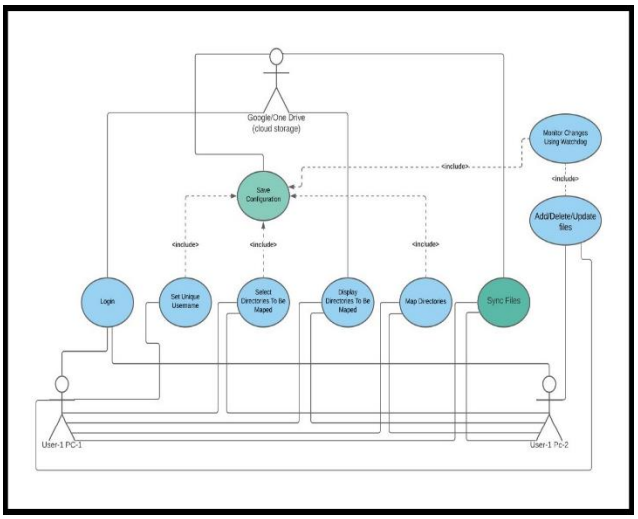


**Figure 5.** Use Case diagram showcasing the process between actors.

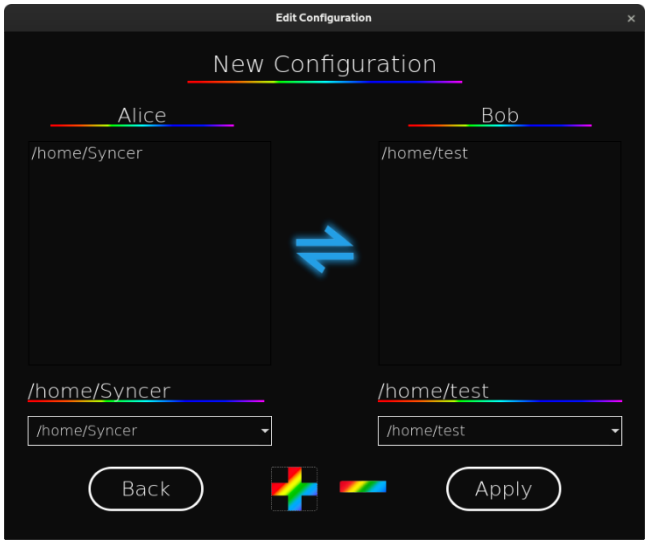


**Figure 6.** Selecting the folders that need to be synced.

## V. ACKNOWLEDGEMENT

Special vote of thanks to: Mike Kazantsev, Dustin Oprea, and Christian Reyes, who regardless of being unknown, gave guidance throughout and our dear friend Gaurav Chawda helped in, from designing the system all the way through fixing bugs.

## VI. CONCLUSION & FUTURE SCOPE

The Pindle application suite is a demanding solution for today's environment. The cloud mounter brings different cloud storages together with multiple accounts at one local place allowing users to save time and switch browser tabs. Python is not suitable for a faster filesystem as it is an interpreted language, but on the other hand it has rich API support and is easy to implement. The GUI is Intuitive and pretty much convenient as users don't have to use a terminal for mounting and unmounting. Local security could be worked out with the current authenticated tokens stored in a pickle file which can be easily penetrated. The current implementation of FUSE allows 4KiB of byte streams at once (some claims it to be 128KiB by adding extra parameter "big_writes" but it doesn't work for our implementation) which makes read write slower, by editing the FUSE kernel module and increasing allowed stream bytes we can improve read write speed significantly. Support for more cloud platforms such as : iCloud, Amazon S3, Mega etc.

In Syncer, Support for more than two-device could be added which will require a more robust concurrent algorithm. Multiple accounts at once are also helpful. The synced folder is not passed through security check-ups therefore the threat to another device also exists hence we require local file security. Also synced folders can be exploited, hindering privacy. Synchronization between folders could be improvised.

## VII. REFERENCES

[1]. https://googleapis.github.io/google-api-python-client/docs/epy/index.html

[2]. "Create GUI applications with Python & Qt5 - 4th edition April 2021" by Martin Fitzpatrick

[3]. Fuse-appendix. 2018. FUSE Library Options and APIs. http://www.fsl.cs.stonybrook.edu/docs/fuse/fuse-article-appendices.html

[4]. "Performance and Resource Utilization of FUSE User-Space File Systems" BHARATH KUMAR REDDY VANGOOR, Avere Systems, Inc. PRAFFUL AGARWAL, MANU MATHEW, ARUN RAMACHANDRAN, and SWAMINATHAN SIVARAMAN, Stony Brook University VASILY TARASOV, IBM Research - Almaden EREZ ZADOK, Stony Brook University ACM Trans. Storage, Vol. 15, No. 2, Article 15, Publication date: May 2019. https://dl.acm.org/doi/fullHtml/10.1145/3310148.

[5]. "Writing a FUSE Filesystem: a Tutorial" Joseph J. Pfeiffer, Jr., Ph.D. Emeritus Professor, Department of Computer Science New Mexico State University Version of 2018-02-04 https://www.cs.nmsu.edu/~pfeiffer/fuse-tutorial/

**Cite this article as :**

Parth Gor, Rayson D'sa, Harit Acharya, Mrs. Geetha S., "PINΔLΞ : FUSE based Filesystem and Folder Syncer", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 7 Issue 4, pp. 561-566, July-August 2021. Available at doi : https://doi.org/10.32628/CSEIT2174128
Journal URL : https://ijsrcseit.com/CSEIT2174128