BABEȘ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE

DIPLOMA THESIS

GCSF – A VIRTUAL FILE SYSTEM BASED ON GOOGLE DRIVE

Supervisor
Lect. dr. Mircea Ioan-Gabriel

Author
Pușcaș Sergiu Dan

2018

UNIVERSITATEA BABEȘ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ ENGLEZĂ

LUCRARE DE LICENȚĂ

GCSF – UN SISTEM DE FIȘIERE VIRTUAL BAZAT PE GOOGLE DRIVE

Conducător științific
Lect. dr. Mircea Ioan-Gabriel

Absolvent
Pușcaș Sergiu Dan

2018

Contents

Chapter 1

Introduction

The past decade has brought the rise of cloud storage services as an extension of traditional disk-based storage systems. There are many options to choose from, and multiple ways of interacting with them. Chapter 2 analyses some of the most popular choices. This thesis will focus on a particular service – Google Drive.

Unfortunately, business requirements encourage companies to target the largest demographic possible, in order to provide the best experience for the average user. This puts advanced users at a disadvantage. Chapter 3 aims to provide a solution tailored specifically for them. This solution comes in the form of a new desktop application for Unix-like systems, named GCSF. GCSF is a virtual file system built on top of Google Drive, allowing users to mount their account locally and interact with it as they would with a traditional disk-based storage system.

Chapter 4 explores existing alternatives and compares them against GCSF, focusing on two aspects – performance and user experience.

Finally, Chapter 5 provides some directions for further development and improvement, and reflects on the progress made since the beginning of this project.

Chapter 2

State of the art

2.1 Cloud storage services

Cloud storage services have been steadily increasing in popularity and reached mainstream status in the past decade. In essence, these services allow users to upload and store files on the internet without the need to own the physical storage medium. Files can be accessed from almost any "smart" consumer device with an internet connection, provided that the user follows the required authentication process.

Most cloud storage services provide free plans, which usually limit the total storage capacity. Their premium plans either follow the subscription business model or even go as far as to require a one-time purchase in exchange for a lifetime subscription [13].

As of July 2018, some of the most popular names ring a bell even to non-technical users:

- Microsoft OneDrive – 115 million customers between 2007 and 2017 [49]
- Google Drive – 800 million active users, March 2017 [44, 58]
- Apple iCloud Drive – a subsystem of Apple iCloud, which reached 782 million users in February 2016 [1]
- Box Drive – 47 thousand paying customers as of August 2015 [3]
- Dropbox – 500 million registered users, 10 million paying users as of December 2017 [51]

This is no coincidence. It is a common practice among tech companies to attempt to draw users into their own ecosystem, especially when it is a vast one. Consumers that already use several products owned by the same company tend to favor that company's other products over their direct competitors'. There are multiple advantages in doing so. For one, familiarity is a key factor. A competitor's service may be more difficult to use by a consumer who is unfamiliar with it. In addition, a single ecosystem provides greater cohesion between its different elements. This translates into better integration between services, which improves the overall user experience.

Two of the largest technology companies – Apple Inc. and Alphabet Inc. – are commonplace examples of this strategy. Users who sign up for a free Gmail account automatically receive an associated 15 GB storage plan on Google Drive. Moreover, this storage is shared among three different services: Google Drive, Gmail and Google Photos. Similarly, Apple offers 5 GB of free storage on iCloud:

> *iCloud is built into every Apple device. That means all your stuff — photos, files, notes, and more — is safe, up to date, and available wherever you are. And it works automatically, so all you have to do is keep doing what you love. Everyone gets 5 GB of free iCloud storage to start, and it's easy to add more at any time.* [26]

It seems that offering free storage is similar in concept to planting a seed that will reap a greater harvest. Indeed, making a user familiar with your service will make him more likely choose your premium plan when deciding to upgrade, instead of your competitor's.

2.2 User interface

All services mentioned in section 2.1 support user interaction through web platforms which allow users to manage their files and data from a web browser. Web browsers represent an ubiquitous category of software programs, so a large userbase can enjoy cloud storage services without the overhead of installing additional software on their machines.

These interfaces are updated regularly with new features and improved design. In May 2018, Google rolled out a new update for Drive which is more in tune with its latest material design principles [23, 24]. Some of the changes can be observed in figures 2.1 and 2.2.

A secondary medium is represented by mobile applications. As of July 2018, Google Drive for Android has more than one billion installs [15]. Dropbox and OneDrive are close behind, with more than 500 million and 100 million installs, respectively [10, 31].

Figure 2.1: Google Team Drive before redesign. Source: [23]

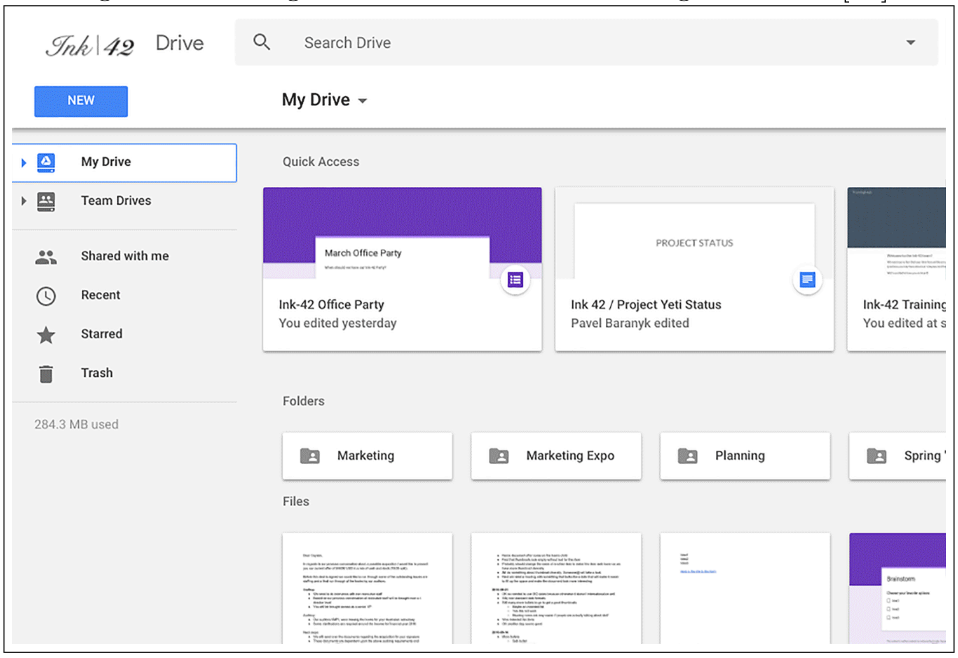


Figure 2.2: Google Team Drive after redesign. Source: [23]

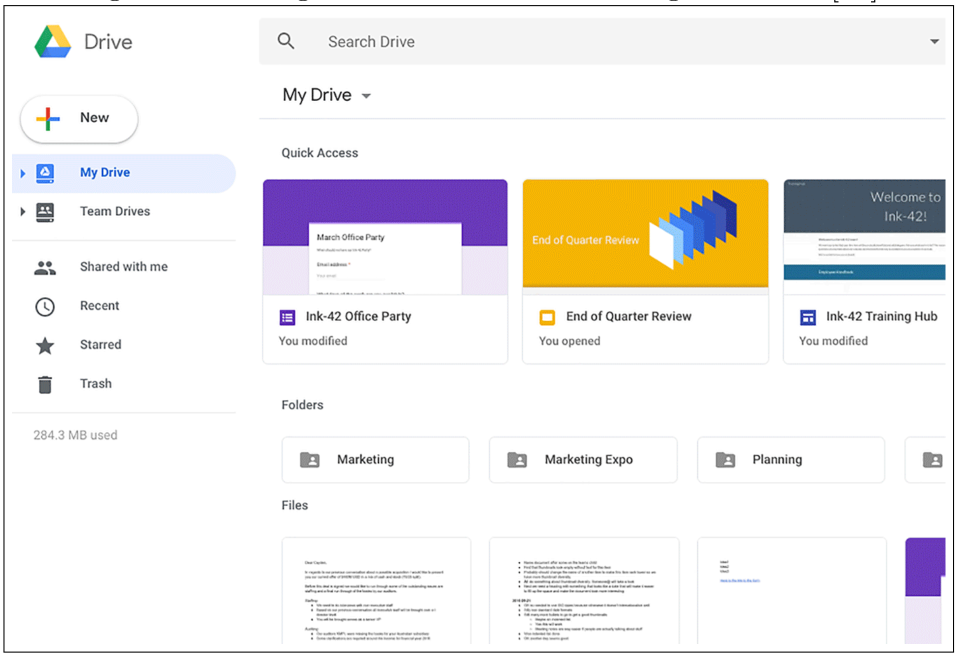
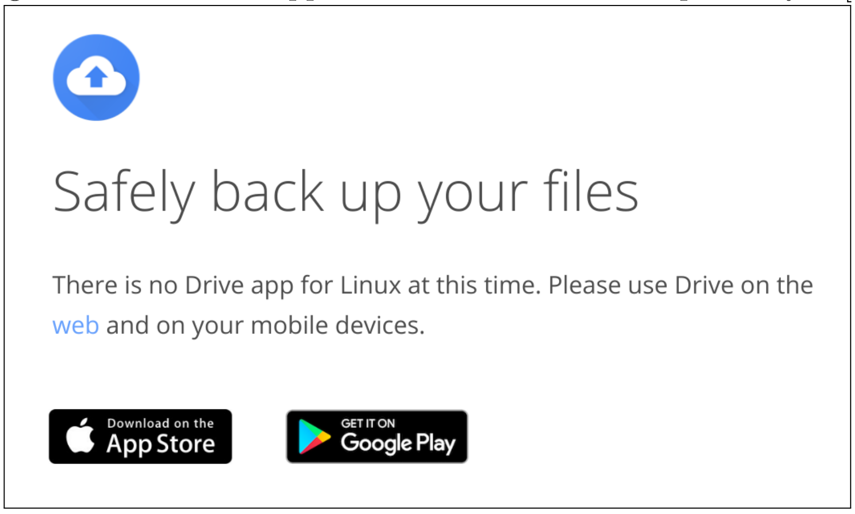


Figure 2.3: No Drive app for Linux. Source: Backup and Sync [2]



Safely back up your files

There is no Drive app for Linux at this time. Please use Drive on the web and on your mobile devices.

2.3 File sync

Originally, users did not interact with cloud storage services as they do today. When Dropbox was initially founded in 2007, it did not rely on a web interface for its service. It did not even own the *dropbox.com* domain [52]. Users had to download a desktop application from *getdropbox.com* and run it locally, thus interacting with the service.

The application followed an approach that was mainly designed by Andrew Houston, the company's CEO. The key element was a special folder added to the user's file system. This folder was then synced to all devices that were linked to the same Dropbox account. Any file placed in this folder would get sent to the cloud and become available to other devices.

The success of Dropbox convinced other companies to follow suit. Google Drive was introduced in April 2012 for Windows, macOS and Android [67]. A few months later, an iOS application was launched as well [45].

In the same year, Google famously told users: *"We're working on Linux support – hang tight!"* [63]. The announcement went under the radar and the project was eventually canceled. This gave birth to an infamous joke website which counts how much time users have been waiting for the Linux client [78].

In September 2017, Google made another announcement regarding the desktop application. The official Drive app would be discontinued in March 2018, its place being taken by the new *Backup and Sync* app [72, 50]. Unfortunately, it doesn't support Linux either, as seen in fig. 2.3.

It is easy to notice then that Linux users have been at a disadvantage ever since Drive was first launched. They essentially had three options: switch to a different storage service or operating system, use the web platform, or use unofficial third-party clients, which were usually bugged and lacked complete functionality.

2.4 Main advantages

2.4.1 Cloud storage

Many technology consumers and enthusiasts use cloud storage services on a daily basis. Compared to traditional disk-based storage systems, cloud-based services provide several advantages. The most palpable one is data redundancy and backup. Users no longer risk the chance of losing their data due to a lost or stolen device or because of a hardware failure. Files that are stored in the cloud provide an extra layer of safety. On top of this, file sharing becomes almost trivial. What used to require manual transfers between different storage mediums (e.g. floppy disks, CDs, flash drives or external hard drives) is now done almost entirely automatically. It is enough to place the desired files in the special folder on a computer and they will appear on all other linked devices which have an internet connection.

Indeed, this is the inconvenience that gave birth to Dropbox. Andrew Houston founded the company after repeatedly forgetting his USB flash drive while being a student at MIT [25].

Another advantage is cross-platform integration. Over a third of the world's population owns a smartphone, and this number is expected to grow in the coming years [56]. Therefore, cloud storage services narrow the gap between traditional desktop computers and mobile phones.

Finally, the freemium model adopted by many of the services mentioned in section 2.1 translates into free extra storage for everyday users. Even the paid plans are a strong competitor for traditional storage systems. In 2018, the cost per gigabyte of storage commonly reaches values as low as \$0.05/GB [55]. With Google Drive's most popular paid plan, this value is approximately 4 times lower: \$0.012/GB/month [19]. Although this is a recurrent cost, it often makes sense financially to opt for cloud storage, especially with all the other provided benefits.

2.4.2 File sync

File syncing is in itself an enticing feature. Having a special folder on an existing file system comes with no additional learning curve. Any computerliterate user can instantly make sense of such a service and use it without prior preparation. Having the data physically present on the local storage system makes it readily available and still accessible even in the absence of an internet connection.

2.4.3 Service integration

As discussed in section 2.1, cloud storage services are seldom provided by themselves. The best example of this is Google, which aims to merge most of its services into a cohesive and consistent user experience. Special files created in Google Docs, Forms, Sheets or Slides use Drive as the storage medium and integrate with it seamlessly. Files can easily be shared with Gmail contacts. Mail attachments can be saved to Drive with a single click. This improves workflow of many consumers globally. Teams no longer have to pass text documents back and forth; all members can edit the same document at the same time.

2.5 Main disadvantages

2.5.1 File sync

Although generally regarded as a good idea, the file sync concept has at least one disadvantage. It requires data to be physically stored on the user's machine. This can be a limiting factor if the storage capacity of the machine is on the lower end of the spectrum. In this case, storing the files only in the cloud and having access to them locally as needed is a better alternative.

2.5.2 Security

Although useful in many aspects, the rise of cloud storage services is a cause of worry for many users. One of the main concerns comes from the risk associated with storing sensitive information on someone else's servers, making it vulnerable to data breaches and leaks.

Dropbox is just one case of controversy. In June 2011 an authentication problem allowed the access of multiple accounts for several hours *without* passwords [53]. In August 2016, 68 million accounts have been compromised by hackers who stole email addresses and hashed passwords [61].

2.5.3 Availability

Throughout the years, web services in general have proved that 100% reliability is unachievable in practice. Google offers a 99.9% uptime guarantee for G Suite customers of Google Drive [20]. However, there have been multiple outages since the service's initial launch in 2012:

- March 2013 [66]
- October 2014 [80]
- January 2016 [48]. Google later apologized:

  > *At Google we recognize that failures are statistically inevitable, and we strive to insulate our users from the effects of failures. As that did not happen in this instance, we apologize to everyone who was inconvenienced by this event. Our engineers are conducting a post-mortem investigation to determine how to make our services more resilient to unplanned network failures, and we will do our utmost to continue to make Google service outages notable for their rarity.* [81]

- September 2017 [62]

These outages carry an immense impact on users. When all Google services went out for 5 minutes in August 2013, the total internet traffic dropped by 40% [74]. Even so, Google Drive's reliability still beats that of user-owned hardware. Consumers are more likely to lose data because of their own hard drives failing than Google's. Still, the perceived effect is the opposite. When you lose your data, you might be tempted to pass it off as a fact of life. When Google loses your data, there is outrage, and rightly so.

2.5.4 Power user experience

Although cloud hosting services usually provide a first-rate user experience, they do not cover every possible use case. This is especially true when discussing about power users. For the purpose of this argument, I will use the following definition of the term:

(noun) **power user**: a computer user who uses advanced features of computer hardware, operating systems, programs, or web sites which are not used by the average user.

This includes students, professional programmers, hobbyists and all other computer users with a large knowledge base who like to tinker with computers in interesting and unexpected ways.

This description matches the traditional stereotype of a computer hacker. The term is known in popular press as someone who breaks into computers. However, it has a different meaning among programmers. In this context, a hacker is usually someone who strikes to achieve mastery by their own merit, being driven by curiosity more often than not. It is not necessarily a computer term: a famous anecdote regarding Nobel laureate Richard Feynman describes his pastime of breaking into safes containing secret documents just for the fun of it [59].

Hackers often create wealth by writing open source software and publishing it for the entire world to use and modify freely [47]. Plenty of them come into contact with programming at an early age and follow formal education only as a secondary way of learning. Hackers are seldom happy with their systems and their knowledge. They always seek new and better ways of making computers do what they want. Living in the terminal usually becomes the preferred way of getting things done.

Although Linux has a modest market share of only 2.16% for desktop and laptop usage, it is far more popular with developers and power users, 48.3% of them having used a variation of it for development work in 2018 [29, 6].

It is therefore unfortunate that the very people who create software are the ones who benefit the least from it, at least in the case of cloud hosting services. Forcing a power user to use a browser in order to share files can easily be an impediment to their productivity. The command line simply does not pair well with browser-based web services. Imagine wanting to write a script that uploads some backup file to such a service and having to implement it in a way that integrates it into the browser. This sort of automation task is not unheard of among power users, yet the lack of native Linux support makes it way harder than it has to be.

Chapter 3 provides a solution to this problem.

Chapter 3

Proposed approach

3.1 Aim

This project aims to improve the experience of using Google Drive specifically for power users. Regular users can benefit from it as well – provided they follow some prerequisite steps in order to use the application.

If successful, it will drastically diminish the some of the issues described in section 2.5.

3.2 Summary

In essence, Google Drive is nothing more than a remote storage system. All the operations that a user might want to execute on a local storage system (e.g. copying files, creating and organising directories, reading and writing data) have an equivalent operation on Google Drive.

Disk-based storage devices are organized by the operating system using a file system in order to keep track of all the data they contain. This happens behind the scenes. Users can interact with the storage device using system calls. Some of the more popular ones are `read`, `write`, `close`, `wait`, `exec`, `fork`, `exit`, `kill`. Note that not all of these deal with file storage. Some of them are a proxy which expose different functionalities of the operating system. In addition, users seldom execute system calls manually. It is the task of higher level applications to do this instead.

An interesting concept comes to mind: why not model a Google Drive account in such a way that it behaves identically to a traditional file system? The only difference would be that instead of storing and reading data from a local disk, it would interact with Google's servers.

GCSF does exactly that. It is a virtual file system on top of Google Drive.

```
$ gcsf
GCSF 0.1.3
Sergiu Puscas <srg.pscs@gmail.com>
Filesystem based on Google Drive

USAGE:
    gcsf <SUBCOMMAND>

FLAGS:
    -h, --help       Prints help information
    -V, --version    Prints version information

SUBCOMMANDS:
    help      Prints this message or the help of the given
    subcommand(s)
    logout    Delete credentials file
    mount     Mount the file system
```

Listing 3.1: GCSF help menu

```
$ gcsf mount /mnt/gcsf
Please direct your browser to https://accounts.google.com/o/
   oauth2/[...] and follow the instructions displayed there.
```

Listing 3.2: GCSF mount

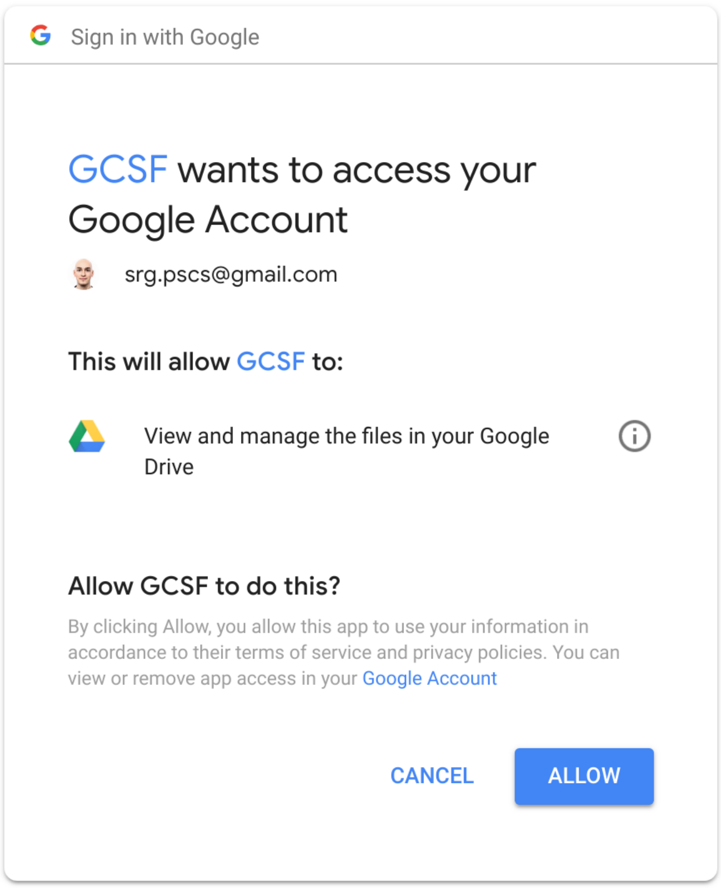
It allows users to mount their Drive account locally and interact with it as they would with a regular disk partition. This is achieved using the FUSE (Filesystem in Userspace) interface [28], as described in section 3.4.

3.3 Usage

Before delving into implementation details, I present a brief overview of how GCSF works from the user perspective.

GCSF consists of a single binary. When executed with no arguments, it prints a help menu as seen in listing 3.1. Users can choose to mount the file system to a local directory. GCSF points to an authentication URL (listing 3.2) that must be accessed in order to authorize access to Google Drive (fig. 3.1). Upon completing the authentication process, GCSF mounts the file system and populates it with all files and directories contained in the *My Drive* directory on Google Drive. This can be observed using tools such as `df` (listing 3.3) and `mount` (listing 3.4).

Now the mount directory can be accessed using a file explorer such as



Sign in with Google

GCSF wants to access your Google Account

srg.pscs@gmail.com

This will allow GCSF to:

View and manage the files in your Google Drive

Allow GCSF to do this?

By clicking Allow, you allow this app to use your information in accordance to their terms of service and privacy policies. You can view or remove app access in your Google Account

Figure 3.1: Google Drive authorization

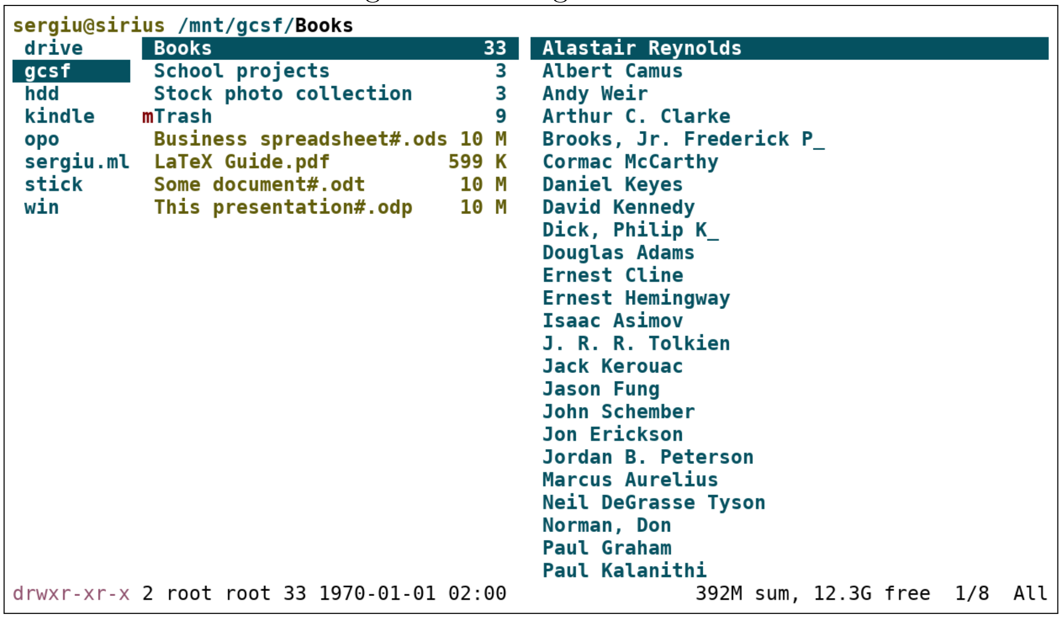
```
$ df -h
Filesystem        Size  Used  Avail  Use%  Mounted on
/dev/nvme0n1p5     64G   45G    17G   74%  /
/dev/nvme0n1p6    644G  559G    53G   92%  /home
/dev/nvme0n1p1    256M   73M   184M   29%  /boot
GCSF               15G   11G   4.6G   70%  /mnt/gcsf
```

Listing 3.3: Size and capacity of mounted file systems

```
$ mount | grep GCSF
GCSF on /mnt/gcsf type fuse (rw,nosuid,nodev,relatime,user_id
    =1000,group_id=100,allow_other)
```

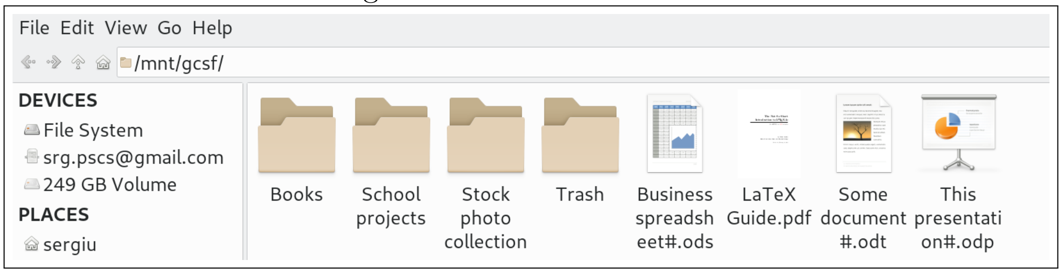
Listing 3.4: Output of mount

Figure 3.2: Ranger window



```
sergiu@sirius /mnt/gcsf/Books
 drive      Books                          33  Alastair Reynolds
 gcsf       School projects                 3  Albert Camus
 hdd        Stock photo collection          3  Andy Weir
 kindle    mTrash                           9  Arthur C. Clarke
 opo        Business spreadsheet#.ods 10 M  Brooks, Jr. Frederick P_
 sergiu.ml  LaTeX Guide.pdf           599 K  Cormac McCarthy
 stick      Some document#.odt         10 M  Daniel Keyes
 win        This presentation#.odp     10 M  David Kennedy
                                               Dick, Philip K_
                                               Douglas Adams
                                               Ernest Cline
                                               Ernest Hemingway
                                               Isaac Asimov
                                               J. R. R. Tolkien
                                               Jack Kerouac
                                               Jason Fung
                                               John Schember
                                               Jon Erickson
                                               Jordan B. Peterson
                                               Marcus Aurelius
                                               Neil DeGrasse Tyson
                                               Norman, Don
                                               Paul Graham
                                               Paul Kalanithi
drwxr-xr-x 2 root root 33 1970-01-01 02:00                392M sum, 12.3G free  1/8  All
```

Figure 3.3: Thunar window

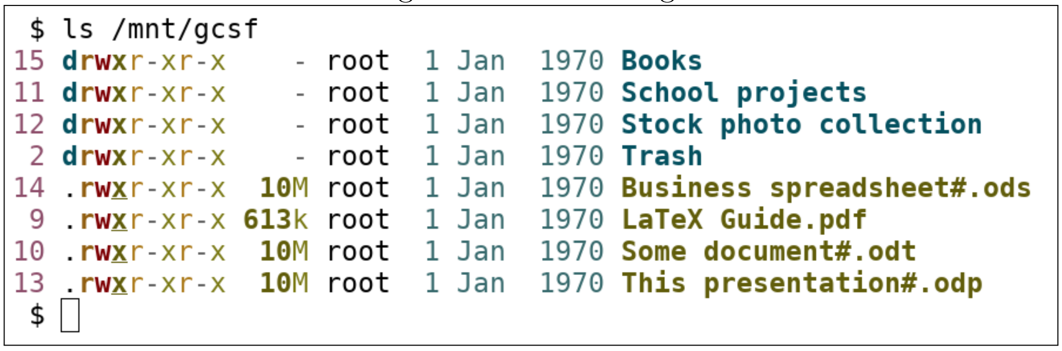


Ranger (fig. 3.2) or *Thunar* (fig. 3.3). Command line programs such as `ls` or `exa` work as well (fig. 3.4). From this point onward, the mount directory can be treated as a regular local directory apart from a few exceptions.

3.3.1 Exceptions

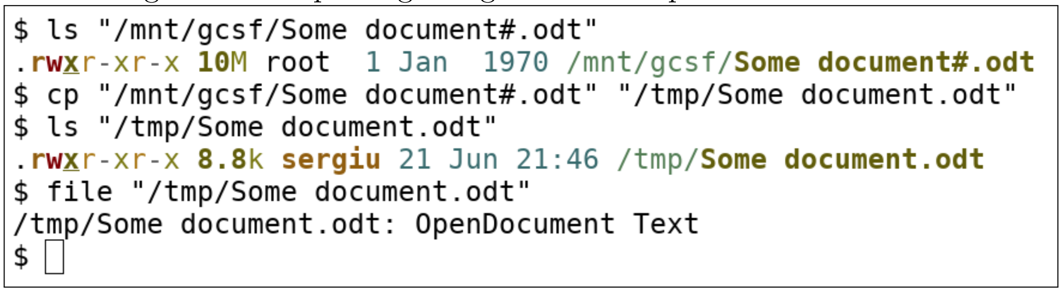
As can be seen in fig. 3.4, some files have a `#` symbol attached at the end of their name. This is the case for special Google Drive files, including spreadsheets, docs and slides. Since their file format and size is undefined at this point, GCSF reports the maximum file size supported for these types of files. When such a file is accessed by a user, GCSF chooses the most appropriate filetype and exports the file from Drive. It also adds the appropriate extension to the file name and updates its real size (fig. 3.5). Due to the

Figure 3.4: File listing



```
 $ ls /mnt/gcsf
15 drwxr-xr-x    - root  1 Jan  1970 Books
11 drwxr-xr-x    - root  1 Jan  1970 School projects
12 drwxr-xr-x    - root  1 Jan  1970 Stock photo collection
 2 drwxr-xr-x    - root  1 Jan  1970 Trash
14 .rwxr-xr-x  10M root  1 Jan  1970 Business spreadsheet#.ods
 9 .rwxr-xr-x 613k root  1 Jan  1970 LaTeX Guide.pdf
10 .rwxr-xr-x  10M root  1 Jan  1970 Some document#.odt
13 .rwxr-xr-x  10M root  1 Jan  1970 This presentation#.odp
 $ 
```

Figure 3.5: Exporting Google Docs as OpenDocument files.



```
$ ls "/mnt/gcsf/Some document#.odt"
.rwxr-xr-x 10M root  1 Jan  1970 /mnt/gcsf/Some document#.odt
$ cp "/mnt/gcsf/Some document#.odt" "/tmp/Some document.odt"
$ ls "/tmp/Some document.odt"
.rwxr-xr-x 8.8k sergiu 21 Jun 21:46 /tmp/Some document.odt
$ file "/tmp/Some document.odt"
/tmp/Some document.odt: OpenDocument Text
$ 
```

nature of such files, they cannot be edited locally and are essentially limited to read-only access. However, they can be exported in a known file format, edited locally and then stored separately using GCSF.

3.4 Implementation

3.4.1 Rust

Rust is a relatively new systems programming language sponsored by the Mozilla Foundation [34]. According to *The Rust Programming Language* book [54],

> Rust is a programming language that helps you write faster, more reliable software. High-level ergonomics and low-level control are often at odds with each other in programming language design; Rust stands to challenge that. Through balancing powerful technical capacity and a great developer experience, Rust gives you the option to control low-level details (such as memory usage) without all the hassle traditionally associated with such control.

From this description, Rust seems like a good fit for this type of project. However, there are a few specific reasons why I chose it instead of a different language:

1. Performance. Rust is in many cases on par with C/C++ (or even faster!) in terms of performance [4]. Compared to interpreted languages like Python or Ruby, this is a clear advantage.
2. Type safety. Any code that may lead to undefined behavior in Rust must be explicitly wrapped within an `unsafe { }` block, making it the programmer's responsibility to make sure that the code is correct. GCSF contains a single unsafe block, required for mounting the file system [11].
3. Memory safety.
    - *No null pointer dereferences.* Rust does not have the concept of a `NULL` pointer. Instead, it uses a construct borrowed from the functional world (`Option<T>`) which encourages the programmer to always check the existence of a packed value.
    - *No dangling pointers.* Instead of using a garbage collector, Rust has a set of rules that define when and how allocated memory is freed. All of these rules are enforced at compile time, thus eliminating an entire category of runtime errors. The rules are:
        (a) Any value has a single owner at any given time.
        (b) References can not outlive the objects they point to.
        (c) At any point, there can be at most one mutable reference *or* any number of immutable (read-only) references to a value.
    - *No buffer overruns.*
4. Easy integration of third-party libraries using the `cargo` package manager and the official package registry [38].
5. Support for the functional paradigm. Rust uses many functional concepts, *closures* and *iterators* being two notable examples.
6. Standardized ecosystem and builtin tools. Rust packages are commonly published on crates.io [38]. Documentation is generated by `rustdoc` and published automatically to docs.rs [8]. The official style guide can be enforced using `rustfmt`.
7. Community and growth. According to the *stackoverflow developer survey* [7], developers have chosen Rust as the most loved programming language for three years in a row.

Some of the points stated above are discussed in more depth in Jim Blandy's book *Why Rust?* [42].

3.4.2 FUSE

FUSE (**F**ilesystem in **Use**rspace) is a project that allows users to create virtual file systems in the user level. Internally, it delegates tasks to a kernel module. As a result, users do not have to interact with the kernel directly. FUSE is a popular choice for esoteric file systems which do not store data themselves. A few notable examples are *sshfs* [36], which mounts a remote file system using SFTP [35], and *WikipediaFS* [40] which allows users to view and edit articles locally.

FUSE is made up of two components:

- the *fuse* kernel module
- the *libfuse* userspace library

FUSE file systems are usually implemented as regular applications that interact with the *libfuse* library. This library provides two interfaces that are useful for defining the behavior of the file system. In both cases, the file system receives incoming requests from the kernel, which are provided as calls to methods defined in the interface.

First, there is the high-level interface. It uses concepts such as file names and paths in most of its methods. Second, there is the low-level interface, which uses inodes in order to identify files [75]. GCSF uses the latter because of the available language library for Rust, provided through the `fuse` crate [33].

3.4.3 Drive API

Google provides a REST API for interacting with Drive [21]. It also provides official client libraries for multiple programming languages: Java, JavaScript, .NET, Objective-C, PHP and Python. There are also early-stage libraries for Dart, Go, Node.js and Ruby.

Unfortunately, Rust is not officially supported. There is however a set of unofficial libraries, programmatically generated by Sebastian Thiel [76, 77]. Although imperfect, they are good enough for the scope of this project. Some limitations are described in section 3.5.

3.4.4 Architecture

The heart of the application is the `GCSF` struct. It implements the `Filesystem` trait from the `fuse` module, essentially making it a mountable file system. Internally, a `FileManager` is used for bookkeeping. The `FileManager` provides all the required functionality for dealing with local files and directories. It keeps track of the file hierarchy, inodes, metadata and performs regular syncs. In order to communicate with Drive, it uses a `DriveFacade` which facilitates remote operations. The `DriveFacade` is linked to the user's account.

These abstractions create a distinct separation of responsibilities. For instance, anything that requires network communication is performed by the `DriveFacade`. Information about any file can be obtained by simply querying the `FileManager`. (Un)mounting the file system and responding to system calls is done by `GCSF`.

Figure 3.6 outlines how these components interact with each other.

3.4.5 Configuration

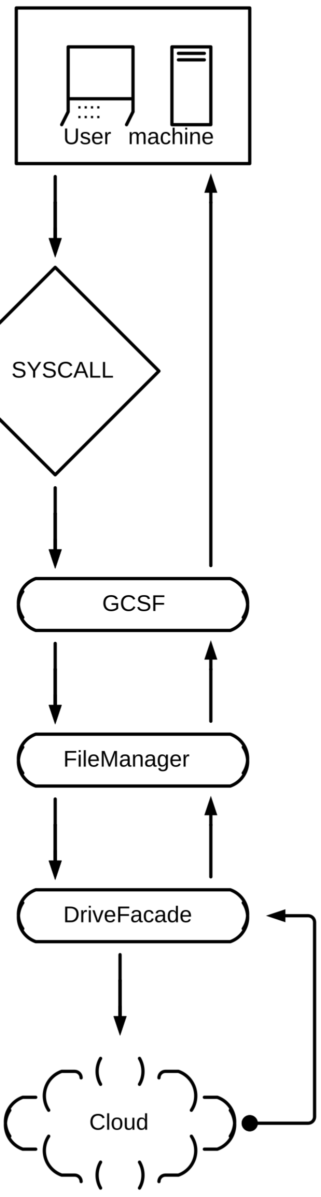
A configuration file can be used in order to set different parameters for GCSF. One example can be found in listing 3.5.

3.4.6 Caching and laziness

Because internet connections tend to be slow and imperfect, GCSF aims reduce unnecessary network requests. The main way of achieving this is by caching file contents. Considering that the content of a given file is unlikely to be modified right after being accessed, it can be stored locally for a small amount of time for faster access. The effect of such a strategy can be observed by measuring the execution time of successive read operations on the same file (listing 3.6). From a technical standpoint, this feature involves the use of an LRU cache, which is incidentally a remarkably popular programming interview question. Similarly, the reported file system size and capacity values are also cached for a small amount of time. In addition, the file tree is permanently maintained in-memory because of its insignificant size. This allows both local and remote changes to be applied quickly.

The second strategy for reducing network requests consists of only uploading new data when necessary. In a UNIX environment, the action of copying a single file might involve a large number of system calls. For instance, the file might have to first be created (using a `create` call), then have its attributes filled in (using a `setattr` call), then have its data written

Figure 3.6: GCSF architecture



```
### This is the configuration file that GCSF uses.
### It should be placed in $XDG_CONFIG_HOME/gcsf/gcsf.toml,
### which is usually defined as $HOME/.config/gcsf/gcsf.toml

# Show debug logs?
debug = true

# How long to cache the contents of a file after it has been
# accessed.
cache_max_seconds = 300

# How how many files to cache.
cache_max_items = 10

# How long to cache the size and capacity of the file system.
# These are the values reported by ‘df’.
cache_statfs_seconds = 1

# How many seconds to wait before checking for remote changes
# and updating them locally.
sync_interval = 10000

# Mount options
mount_options = [
    "fsname=GCSF",
    "allow_root",
    "big_writes",
    "max_write=131072"
]

# If set to true, Google Drive will provide a code after
# logging in and authorizing GCSF. This code must be copied
# and pasted into GCSF in order to complete the process.
# Useful for running GCSF on a remote server.
#
# If set to false, Google Drive will attempt to communicate
# with GCSF directly. This is usually faster and more
# convenient.
authorize_using_code = false
```

Listing 3.5: GCSF configuration file

```
$ time file "/mnt/gcsf/LaTeX Guide.pdf"
LaTeX Guide.pdf: PDF document, version 1.5

real    0m2.751s
user    0m0.005s
sys     0m0.005s
$ time file "/mnt/gcsf/LaTeX Guide.pdf"
LaTeX Guide.pdf: PDF document, version 1.5

real    0m0.023s
user    0m0.017s
sys     0m0.001s
```

Listing 3.6: File caching in action

(using potentially many `write` calls with different offsets and data buffers) and finally be `flush`ed. Uploading every small change that comes with each individual `write` call does not make a lot of sense. For this reason, GCSF decides to be lazy and simply store the `PendingWrite`s in memory. The operations are only performed when a `flush` call is encountered, and the new file content is uploaded as a whole afterwards.

3.5 Problems encountered

As with any sufficiently large project, GCSF involved a series of annoying and obscure problems along the way. I will describe some of the most notable ones in this section.

3.5.1 Shared files

The Google Drive REST API provides the `files.list` endpoint for listing files which meet some criteria [16]. One of the filters that can be applied is the `sharedWithMe` boolean. It instructs the API to include or omit files that are shown in the *Shared with me* collection on Drive.

Early implementations of GCSF did not aim to manage shared files at all, as this feature requires extra work to get right. Excluding shared files is easy – just add `sharedWithMe = false` in the request. Unfortunately, this does not work, as reported by multiple other developers [22, 14, 17]. Instead of returning the requested list of files, the API returns a **500 Internal Server Error**. There is no warning sign for this behavior.

A possible workaround consists of replacing the query with `'me' in owners`. This is intuitive – files that are shared with a user are usually not owned by that user, making the two queries partial substitutes for one another. This workaround is not failproof. Exceptions exist and they lead to inconsistent behavior.

The solution I opted for involves a different querying strategy. Instead of asking for all files which are not shared, GCSF asks for files which are direct children of the *My Drive* directory. This is equivalent to setting the query to `'root' in parents`. Afterwords, the returned files are processed. In order to explore the rest of the file tree, GCSF recursively queries children of any one of the directories obtained at the previous step. For instance, if the children of the `root` directory are `a`, `b` and `c`, then the next query will be `'a' in parents or 'b' in parents or 'c' in parents`. This essentially explores the file tree one level at a time, resulting in $O(\textit{tree depth})$ network requests.

3.5.2 'My Drive' id

Every file on Google Drive has its own associated string identifier. As discussed in 3.5.1, GCSF populates the local file tree starting with the root (*My Drive*) directory. This is achieved by setting the query `'root' in parents`, where `'root'` is a placeholder recognized by the API. However, GCSF needs to know the real identifier for this directory, because files placed in it will use that identifier as their `parent` field.

The question is: how to obtain this identifier? There is no method for retrieving it from the API.

One way is to get all files that match the `'root' in parents` query and check the actual identifier that they have in the `parent` field. GCSF can then use this identifier instead of the `'root'` placeholder from this point onward. This method usually works, but there were some cases in which the application crashed because of it. For instance, if there are absolutely no files on Drive, the root identifier cannot be obtained. The current implementation works around this limitation by postponing the operation until at least one file is added to Drive.

3.5.3 File attributes

The `fuse` crate represents file attributes using the `FileAttr` struct, as seen in listing 3.7. The `perm` field is an encoding which describes a file's permissions, essentially stating what any user on the system can and can't do with that particular file. Any `Filesytem` can alter these permissions using the `setattr` method, as seen in listing 3.8.

```rust
pub struct FileAttr {
    pub ino: u64,
    pub size: u64,
    pub blocks: u64,
    pub atime: Timespec,
    pub mtime: Timespec,
    pub ctime: Timespec,
    pub crtime: Timespec,
    pub kind: FileType,
    pub perm: u16,
    pub nlink: u32,
    pub uid: u32,
    pub gid: u32,
    pub rdev: u32,
    pub flags: u32,
}
```

Listing 3.7: File attributes representation

```rust
fn setattr(
    &mut self,
    _req: &Request,
    _ino: u64,
    _mode: Option<u32>,
    _uid: Option<u32>,
    _gid: Option<u32>,
    _size: Option<u64>,
    _atime: Option<Timespec>,
    _mtime: Option<Timespec>,
    _fh: Option<u64>,
    _crtime: Option<Timespec>,
    _chgtime: Option<Timespec>,
    _bkuptime: Option<Timespec>,
    _flags: Option<u32>,
    reply: ReplyAttr
) { ... }
```

Listing 3.8: Setting file attributes

Notice something missing? There is no argument for changing the file permissions. This defeats the entire purpose of recognizing different user permissions and is the reason why GCSF does not enforce this security feature.

3.5.4 Updating file metadata on Drive

The Rust library that GCSF uses is generated based on a template which is mostly consistent with the API schema that Google provides. However, there are some inconsistent methods which act as limitations for this project.

One of them regards updates on file metadata. GCSF sometimes has to modify information about a file without changing its content. For instance, renaming a file or moving it to a different directory are operations which require this sort of behavior. In order to to this, a `FileUpdateCall` can be used.

For context, most calls exposed by the library follow the builder pattern [46]. The general structure is:

```rust
let result = hub.resource().activity(...).doit();
```

Or specifically:

```rust
let result = hub.files().copy(...).doit();
let result = hub.files().create(...).doit();
let result = hub.files().list(...).doit();
let result = hub.files().delete(...).doit();
```

But the `update` call is different. Instead of exposing a public `doit()` method which does all the work, it exposes two alternatives:

```rust
pub fn upload<RS>(
  self,
  stream: RS,
  mime_type: Mime
) -> Result<(Response, File)>

pub fn upload_resumable<RS>(
  self,
  resumeable_stream: RS,
  mime_type: Mime
) -> Result<(Response, File)>
```

This means that there is no way to update a file's metadata without also providing new content. One terribly inefficient solution would consist of first downloading the current file content, then modifying the relevant file metadata and finally applying the changes by re-uploading the content exactly as it is, along with the new metadata. In fact, this is how earlier versions of GCSF worked around this issue. As of version 0.1.2, GCSF uses a custom fork [5] of the library which also exposes a new method which does not require uploading file content:

```
pub fn doit_without_upload(mut self) -> Result<(Response,
    File)>
```

3.5.5 Detecting remote changes

GCSF aims to achieve data consistency in two directions: any change applied locally should also be applied on Drive, and any change applied remotely should also be reflected locally. Remote changes are detected using the `changes.list` API endpoint [9]. GCSF follows the following polling policy:

- It should ask for remote changes and apply them locally right before serving user requests, so that the user only receives fresh data.
- It should not do this for every request, so that the added latency does not impact the user experience. There should be a user-defined cooldown period between syncs.

This generally works. Files added/removed/modified on the web or mobile client are picked up by GCSF after a short while. No wasteful network traffic is generated, as would be the case with regular polling. The delay perceived by the user is present but not significant as it only occurs every once in a while.

But there is a problem. The API seems to make up its own mind and only return the requested changes when it wants. Sometimes, this happens almost instantly. On other occasions, the user has to manually intervene and remount the file system in order to preserve its consistency. This can lead to problems which impair user experience.

Chapter 4

Performance evaluation

Although comparing GCSF with similar tools is a fuzzy task, I will attempt to construct an appropriate performance analysis. For this purpose I have selected two other projects:

- *dsoprea/GDriveFS* [64], which I personally used prior to starting work on GCSF.
- *astrada/google-drive-ocamlfuse* [73] – the most popular project of those mentioned in this section.

Besides the two, there are many other similar projects. Most of them are either in early stages of development, abandoned, or serve a different purpose:

- *thejinx0r/node-gdrive-fuse* [83] – unmaintained since February 2016.
- *joe42/CloudFusion* [27] – unmaintained since January 2015.
- *S2Games/drivefs* [71] – unmaintained since June 2014.
- *BYVoid/gdrive* [57] – unmaintained since October 2013.
- *jcline/fuse-google-drive* [43] – unmaintained since September 2012.
- *thejinx0r/DriveFS* [82] – undocumented as of June 2018 and still in early stages of development.
- *zond/futon* [41] – unmaintained since December 2014. In addition, it is *"right now, and probably forever, read only"* according to the documentation.
- *dweidenfeld/plexdrive* [79] – only allows read-only access and targets media streaming.

As such, I have decided to exclude these projects from my analysis and benchmarks. Here is a brief comparison of the chosen projects, as of 23 June 2018:

		GCSF	GDriveFS	google-drive-ocamlfuse
GitHub Statistics	Owner	Sergiu Pușcaș	Dustin Oprea	Alessandro Strada
	First Commit	April 2018	August 2012	May 2012
	Commits	130	395	511
	Releases	1	23	71
	Contributors	1	6	12
	Stars	11	491	2086
	Forks	0	81	153
Technical Details	Language	Rust	Python 2.7	OCaml
	LoC	2093	5232	7962

In the next sections I will discuss each of these projects from a technical standpoint and from a user perspective.

4.1 GDriveFS

As of June 2018, *GDriveFS* aims to be *“an innovative FUSE wrapper for Google Drive”* [64]. It has been in development for the past six years, accumulating along the way a total of almost 400 commits, 23 releases, 6 contributors and 81 forks. As a consequence, GDriveFS has more features than GCSF and its longevity makes it a time-proven piece of software. This has been consistent with my personal experience. GDriveFS worked out of the box in most situations where I attempted to use it.

Unfortunately, I also encountered some issues. I will walk through a first time setup of this application in order to illustrate its drawbacks. First, we create a new virtual environment using `virtualenv` in order to avoid conflicts between global Python packages and the local requirements of GDriveFS. We can easily install GDriveFS in this environment as a `pip` package.

```
$ virtualenv2 gdrivefs
New python executable in ./gdrivefs/bin/python2
Also creating executable in ./gdrivefs/bin/python
Installing setuptools, pip, wheel...done.
$ source gdrivefs/bin/activate
(gdrivefs) $ pip install gdrivefs
Collecting gdrivefs
Collecting fusepy==2.0.2 (from gdrivefs)
Collecting httplib2==0.8 (from gdrivefs)
[...]
```

```
Successfully installed gdrivefs-0.14.9 [...]
```

Listing 4.1: Creating a virtual environment and installing GDriveFS

Now we are ready to run into our first problem. The official documentation suggests using `gdfstool auth_automatic` in order to log in with our Google account [65]. However, we encounter an error:

```
(gdrivefs) $ gdfstool auth_automatic
usage: gdfstool [-h] {auth,mount} ...
gdfstool: error: argument command: invalid choice: ‘
    auth_automatic’ (choose from ‘auth’, ‘mount’)
```

Listing 4.2: GDriveFS nonexistent authentication command

It seems that there is an inconsistency between the documentation and the application itself. No problem. We can follow the suggestion provided by the error message and execute `gdfstool auth` instead. We provide the `-o` flag in order to open the authentication form in a browser window. After logging in and allowing the application to access our account, we are ready to feed the access code into `gdfstool`:

```
(gdrivefs) $ gdfstool auth -a /tmp/credentials "$AUTH_CODE"
[...]
gdrivefs.errors.AuthorizationFailureError: Could not do auth-
    exchange (this was either a legitimate error, or the auth-
    exchange was attempted when not necessary): [SSL:
    CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl
    .c:726)
```

Listing 4.3: GDriveFS authentication error

This is strange. After researching the problem, we find a relevant open issue on this subject [12]. According to user *blitz313*, the root cause is one of the dependencies. Manual installation of package `httplib2-0.10.3` (instead of the required version `0.8`) seems to solve this problem and we can finally mount the file system:

```
(gdrivefs) $ gdfstool mount /tmp/credentials /mnt/gdrivefs
(gdrivefs) $ ls /mnt/gdrivefs/
drwxrwxrwx@    - sergiu 11 Jun 18:56 Books
drwxrwxrwx@    - sergiu 11 Jun 19:04 School projects
drwxrwxrwx@    - sergiu 11 Jun 18:59 Stock photo collection
.rw-rw-rw-@ 1.0k sergiu 11 Jun 18:57 Business spreadsheet#
.rw-rw-rw-@ 613k sergiu 11 Jun 19:38 LaTeX Guide.pdf
.rw-rw-rw-@ 1.0k sergiu 12 Jun 15:27 Some document#
.rw-rw-rw-@ 1.0k sergiu 11 Jun 18:57 This presentation#
```

Listing 4.4: GDriveFS filesystem mount

From this point on, most operations perform as expected. My biggest issue with GDriveFS is its async I/O strategy. Files written to the file system are not instantly updated locally or on Drive. Reading a file too soon after writing to it can cause data inconsistency. In my benchmarks, writing files larger than 10 MB and reading them immediately afterwards often resulted in checksum failures. Reading the same file multiple times in a row can also result in different outputs.

In the situations where GDriveFS did work, it required significantly more time than its competitors. This may be attributed to the fact that it is implemented in Python, which is known to be slower than statically-typed compiled languages.

Overall, the experience can only be described as messy. GDriveFS makes it difficult to reason about the state of the operations performed, and what you see is not always what you get.

4.2 google-drive-ocamlfuse

Compared to GDriveFS, google-drive-ocamlfuse is the result of the collective effort of twice as many contributors. As of June 2018, it has been starred by more than 2000 users on GitHub. For comparison, the programming language it is written in only has 1850 stars [37].

This project is implemented in OCaml [30], "an industrial strength programming language supporting functional, imperative and object-oriented styles". According to open-source software developer Thomas Leonard, OCaml can often achieve better performance compared to Python [60]. Whether or not this is the case in general, it is certainly reflected in the case of this project. As it turns out, it is the best performer in several categories described in section 4.3.

From a user perspective, google-drive-ocamlfuse also has its flaws. For once, it does not support authentication using a generated code. This makes it more difficult to use on a headless machine, but it is not a problem for most users. Installation can also be tricky. I have personally run into issues such as [32] while trying to set up the file system. The Arch Linux package [18] does not automatically pull in all dependencies, requiring manual installation of several packages.

However, after setting the file system up, all of these issues disappear. The user experience is unimpaired. In all tests, there were no cases of data inconsistency or file system errors.

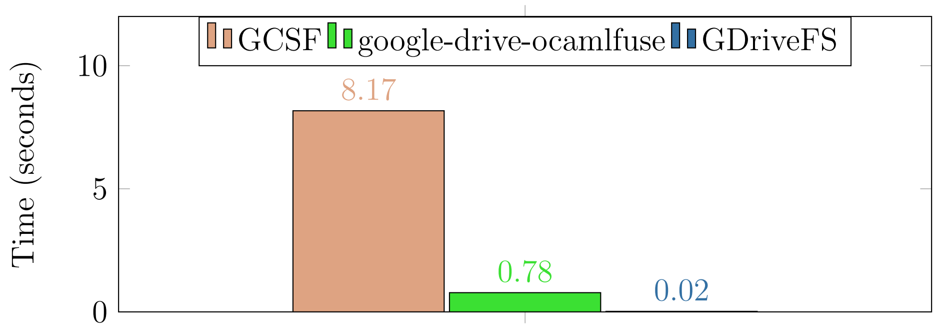


Figure 4.1: Mounting a file system with 2000 files and 100 directories, nested on 4 levels.

4.3 Benchmarks

4.3.1 Methodology

All tests were performed on a machine with the following specifications:

- Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
- SSD storage
- 4 GB RAM
- 3 Gb/s internet connection

Unless stated otherwise, all file systems were mounted using their default configuration. All results reported in the following sections are averaged among 10 successive executions. The test account has been artificially populated with 2000 files and 100 directories, totaling 2.8 GB. The maximum depth of any file in the the file tree is 4.

4.3.2 Mounting and startup

Compared to google-drive-ocamlfuse and GDriveFS, GCSF populates the file tree at mount time. As discussed in 3.5.1, the startup time of GCSF grows linearly with the depth of the file tree. This means that the file system will take longer to load deeply nested directories, but it has no problems with a large number of files in a shallow file tree.

As seen in fig. 4.1, constructing the file tree at mount time increases the loading time of the file system, but it significantly improves subsequent operations. More on this in section 4.3.3.

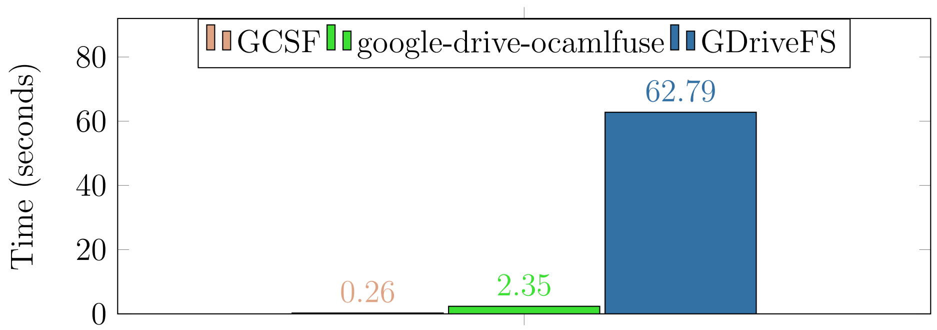


Figure 4.2: Listing all files and directories recursively.

4.3.3 File listing

One of the first operations performed after mounting the file system is in many cases a file listing. Whether this is achieved using a command line tool such as `ls` or a GUI file explorer, the file system has to respond to the same system calls – mainly `readdir` and `lookup`. For this benchmark, I decided to measure the execution time of the `tree` command [39]. This command explores an entire directory recursively, constructing an ASCII tree-like structure of all files and directories.

This is where the longer mount time of GCSF pays off. As seen in fig. 4.2, GCSF is one order of magnitude faster than google-drive-ocamlfuse, and two orders of magnitude faster than GDriveFS. It should be noted that the first listing performed on google-drive-ocamlfuse is considerably slower than subsequent listings – approximately 60 seconds – because of the empty cache. This outlier is not included in the computed average.

4.3.4 Reading – empty cache

The target of this test is to measure the execution time required for computing an MD5 checksum of a randomly generated file. This allows us to check whether or not the file content is retrieved by each file system in its entirety and without errors. Although computing the checksum takes some time in itself, it only affects the measured times marginally.

As seen in fig. 4.3, google-drive-ocamlfuse tends to perform best on small files. GCSF is slower because it makes an additional request to Google Drive: first it creates an empty file and then it updates the file content. However, the performance changes for larger files. In this case, an additional network request becomes insignificant compared to each file system's efficiency of processing operations internally. As it turns out, GCSF performs slightly

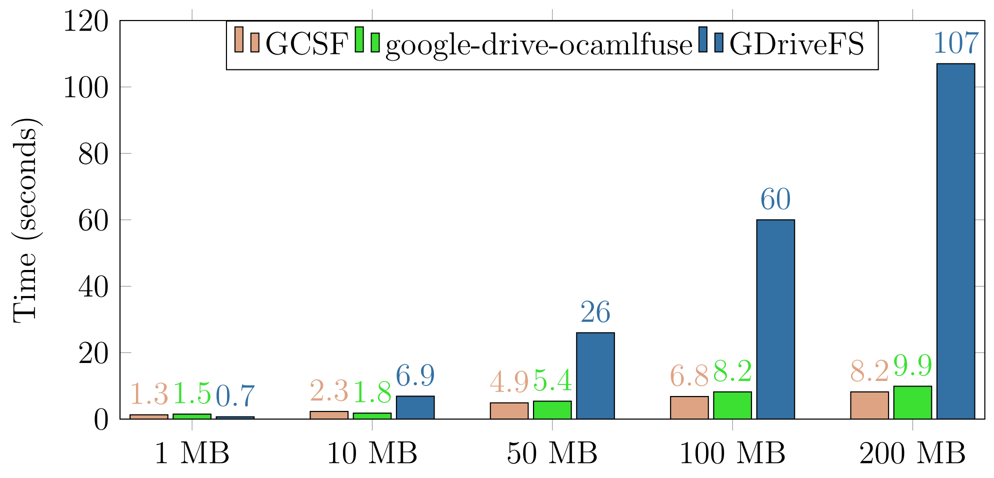


Figure 4.3: Reading a file when the cache is empty.

better than google-drive-ocamlfuse.

Not the same can be said about GDriveFS. Its execution time grows almost linearly with the file size. Moreover, the computed checksums are incorrect in many cases. Waiting for the file system to re-synchronize with Drive may fix these errors, but the inconvenience of not knowing whether or not the file system introduced errors to user data still exists.

File size (MB)	Checksum errors (%)	
	Fresh read	Cached read
1	20	10
10	40	60
50	90	70
100	80	100
200	90	100

Table 4.1: Checksum errors reported when reading from GDriveFS

4.3.5 Reading – cached

Both GCSF and google-drive-ocamlfuse provide a caching mechanism. GCSF uses an in-memory least recently used (LRU) cache in order to preserve file content for some time after retrieving it from Drive, whereas google-drive-ocamlfuse uses an on-disk SQLite 3 database. Both methods have their advantages and disadvantages. Retrieving data from memory is faster but

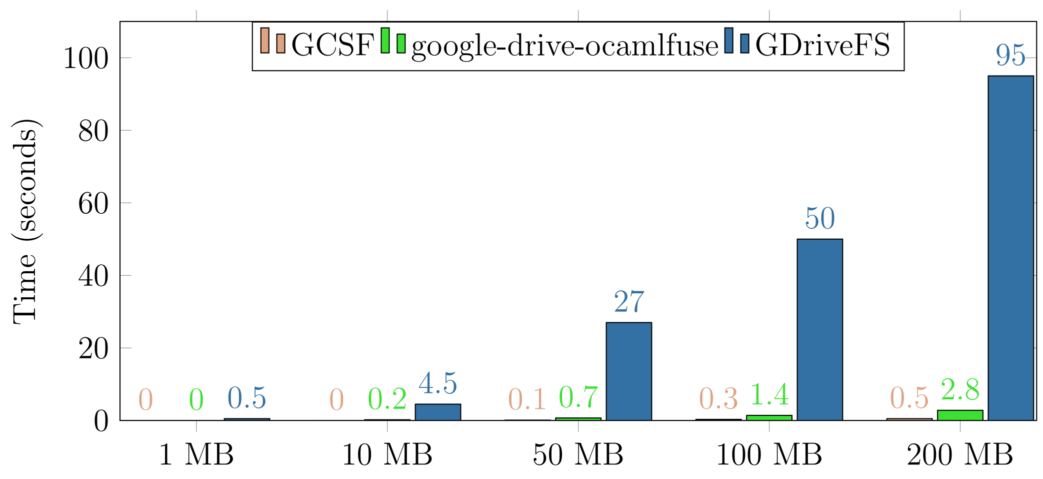


Figure 4.4: Reading a cached file.

the available capacity can be a limiting factor. Caching on disk removes this limitation, at the cost of slower speeds.

As far as I can tell, GDriveFS only caches authentication tokens and the structure of the file tree. This is reflected in fig. 4.4. As was the case in section 4.3.4, checksum errors still occur with GDriveFS. The exact values can be found in table 4.1.

Chapter 5

Conclusion

I personally consider that GCSF is a worthwhile alternative for users who are dissatisfied with existing tools. In many cases, it provides more control compared to the official web platform. As illustrated in Chapter 4, it can also achieve better performance and stability than similar FUSE-based file systems.

This result is especially significant considering the short development cycle of just 11 weeks. The projects I compared it against have been in development for multiple years.

As I plan to continue working on GCSF, I have assembled a list of areas for improvement. In no particular order:

- *Trash directory.* Currently, GCSF shows trashed files and directories but only has limited functionality in this area. Removing files currently moves them to trash, and there is no way to permanently delete files. This will be addressed by introducing context-aware behavior: removing a file from any directory will move it to trash, whereas removing a file from trash will permanently delete it.
- *'Shared with me' collection and team drives.* GCSF only supports the 'My Drive' collection.
- *Faster mount time.* The number of network requests required for populating the file system can be reduced from $O(tree\ depth)$ to $O(1)$, with the addition of a more complex tree building strategy.
- *Extended attributes.* Google Drive stores several custom file attributes in addition to those reported by `lsattr`.
- *Identically named files.* Some operations are not well defined in the case of identically named files because this concept is foreign to traditional file systems. GCSF makes an effort to differentiate them by adding

specific suffixes (e.g `hello.txt.1`, `hello.txt.2` and so on). When a user moves such a file to a different directory, it is not immediately clear what the correct behavior should be. Should the file keep its suffix even if not necessary, just for the sake of consistency? Should the suffix be adapted to match the identical files in the new directory or be removed altogether? A consistent strategy must be determined.

- *User specified MIME type for special Drive files.* GCSF guesses the best export format for Docs, Sheets and Slides. This is the format used by the OpenOffice suite. Users should be able to specify any valid format accepted by Google Drive.
- *Real file size of exportable Drive documents.* GCSF reports a fixed size of 10 MB for Docs, Sheets and Slides, which is not factually accurate. One alternative is to always report the file size that each specific document would have when exported in the default format.
- `gzip` *compression.* Google suggests the use of `gzip` for compressing files before transferring over the network. This requires additional CPU time to decompress the results, but reduces the bandwidth needed in most cases. The trade-off is usually worthwhile.
- *Concurrency.* GCSF can only perform one operation at a time. This is usually not a problem, but in some cases it impedes user experience.
- *Support for symbolic links.*
- *Package releases for multiple Linux distributions.* The de facto method of installing GCSF is via the Cargo package manager. This requires a local installation of both Rust and Cargo and adds some time required for building the project. Packaging binaries for multiple operating systems and architectures would lower the barrier of using the application.
- *File permissions.* Enforcing file permissions would improve user experience in the case of multi-user environments.

5.1 External links

The entire project is hosted on GitHub [68]. It is also published as a Rust crate on crates.io [70]. The latest documentation is accessible on docs.rs [69].

Bibliography

[1] Apple Music passes 11M subscribers as iCloud hits 782M users. http://appleinsider.com/articles/16/02/12/ apple-music-passes-11m-subscribers-as-icloud-hits-782m-users. [Online; accessed 18 June 2018].

[2] Backup and Sync homepage - no Drive app for Linux. https://www.google.com/drive/download/backup-and-sync/. [Online; accessed 21 June 2018].

[3] Box hires Nasdaq exec to head financial services practice. http://fortune.com/2015/08/19/box-financial-services/. [Online; accessed 18 June 2018].

[4] C++11 vs Rust comparison. https://github.com/dmitryikh/rust-vs-cpp-bench. [Online; accessed 21 June 2018].

[5] Custom fork of the Google Drive API library for Rust. https://crates.io/crates/google-drive3-fork.

[6] Developer Survey Results 2018. https://insights.stackoverflow.com/survey/2018/#technology. [Online; accessed 13 June 2018].

[7] Developer Survey Results 2018: most loved, dreaded and wanted technologies. https://insights.stackoverflow.com/survey/2018# most-loved-dreaded-and-wanted. [Online; accessed 13 June 2018].

[8] Docs.rs – an open source project to host documentation of crates for the Rust Programming Language. http://docs.rs. [Online; accessed 21 June 2018].

[9] Drive `changes.list` endpoint. https://developers.google.com/drive/api/v3/reference/changes/list. [Online; accessed 21 June 2018].

[10] Dropbox Android App. https://play.google.com/store/apps/details?id=com.dropbox.android. [Online; accessed 13 June 2018].

[11] `fuse::spawn_mount` unsafe method. https://docs.rs/fuse/0.3.1/fuse/fn.spawn_mount.html. [Online; accessed 23 June 2018].

[12] GDriveFS Issue #195: SSLHandshakeError. https://github.com/dsoprea/GDriveFS/issues/195. [Online; accessed 13 June 2018].

[13] Get 3TB of Lifetime Cloud Storage for Under $70. https://www.thedailybeast.com/get-3tb-of-lifetime-cloud-storage-for-under-dollar70. [Online; accessed 18 June 2018].

[14] Google API NodeJS Client Issue #1136: `files.list` method returns "Bad Request error 400" when `sharedWithMe = false`. https://github.com/google/google-api-nodejs-client/issues/1136. [Online; accessed 13 June 2018].

[15] Google Drive Android App. https://play.google.com/store/apps/details?id=com.google.android.apps.docs. [Online; accessed 13 June 2018].

[16] Google Drive API: `files.list` endpoint. https://developers.google.com/drive/api/v3/reference/files/list. [Online; accessed 13 June 2018].

[17] Google Drive API `sharedWithMe=false` causes "500 Internal Server Error". https://goo.gl/eshSvE. [Online; accessed 13 June 2018].

[18] google-drive-ocamlfuse installation. https://github.com/astrada/google-drive-ocamlfuse/wiki/Installation#archlinux. [Online; accessed 13 June 2018].

[19] Google Drive Pricing Guide. https://www.google.com/drive/pricing/. [Online; accessed 13 June 2018].

[20] Google Drive Reliability. https://support.google.com/googlecloud/answer/6056635?hl=en. [Online; accessed 21 June 2018].

[21] Google Drive REST API Overview. https://developers.google.com/drive/api/v3/about-sdk. [Online; accessed 13 June 2018].

[22] Google Drive SDK `sharedWithMe = false` search query not works. https://stackoverflow.com/questions/24515151/google-drive-sdk-sharedwithme-false-search-query-not-works. [Online; accessed 13 June 2018].

[23] Google Drive UI updates. https://gsuiteupdates.googleblog.com/2018/05/google-drive-ui-updates.html. [Online; accessed 13 June 2018].

[24] How Google created a custom Material theme. https://material.io/articles/how-google-created-a-custom-material-theme.html#01. [Online; accessed 13 June 2018].

[25] How the habit of forgetting became a $10 billion business idea. http://theunvisited.in/how-habit-of-forgetting-became-10-billion-business-idea/. [Online; accessed 18 June 2018].

[26] iCloud – The best place for all your photos, files, and more. https://www.apple.com/lae/icloud/. [Online; accessed 18 June 2018].

[27] joe42/CloudFusion – Linux file system (FUSE) to access Dropbox, Sugarsync, Amazon S3, Google Storage, Google Drive or WebDAV servers. https://github.com/joe42/CloudFusion. [Online; accessed 17 June 2018].

[28] Libfuse repository on GitHub. https://github.com/libfuse/libfuse. [Online; accessed 13 June 2018].

[29] Linux Market Share. https://goo.gl/QPcwGZ. [Online; accessed 13 June 2018].

[30] OCaml. https://ocaml.org/. [Online; accessed 17 June 2018].

[31] OneDrive Android App. https://goo.gl/93Ffot. [Online; accessed 13 June 2018].

[32] OPAM depext issue #75: `opam install depext` fails on macOS. https://github.com/ocaml/opam-depext/issues/75. [Online; accessed 13 June 2018].

[33] Rust FUSE crate - Filesystem in Userspace. https://crates.io/crates/fuse. [Online; accessed 13 June 2018].

[34] Rust Website. https://www.rust-lang.org/en-US/. [Online; accessed 13 June 2018].

[35] SSH File Transfer Protocol. https://tools.ietf.org/html/draft-moonesamy-secsh-filexfer-00. [Online; accessed 13 June 2018].

[36] SSHFS: A network filesystem client to connect to SSH servers. https://github.com/libfuse/sshfs. [Online; accessed 13 June 2018].

[37] The core OCaml system: compilers, runtime system, base libraries. https://github.com/ocaml/ocaml. [Online; accessed 17 June 2018].

[38] The Rust community's crate registry. http://crates.io. [Online; accessed 13 June 2018].

[39] `tree` – Linux man page. https://linux.die.net/man/1/tree. [Online; accessed 18 June 2018].

[40] WikipediaFS. http://wikipediafs.sourceforge.net/. [Online; accessed 13 June 2018].

[41] zond/futon – Google Drive on FUSE. https://github.com/zond/futon. [Online; accessed 17 June 2018].

[42] Blandy, Jim. *Why Rust? Trustworthy, Concurrent Systems Programming*. O'Reilly Media, Inc, 2015.

[43] Cline, James. jcline/fuse-google-drive – A fuse filesystem wrapper for Google Drive. https://github.com/jcline/fuse-google-drive. [Online; accessed 17 June 2018].

[44] Dignan, Larry. Google plans to leverage G Drive for broader enterprise footprint, team management and collaboration. https://goo.gl/DvTfXM. [Online; accessed 13 June 2018].

[45] Gallagher, Sean. Hands-on with the Google Drive for iOS app: mostly read only. https://arstechnica.com/information-technology/2012/06/hands-on-with-the-google-drive-for-ios-app-mostly-read-only/. [Online; accessed 13 June 2018].

[46] Gamma, Erich and Johnson, Ralph and Vlissides, John and Helm Richard. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[47] Graham, Paul. *Hackers & Painters: Big Ideas from the Computer Age*. O'Reilly Media, Inc, 2004.

[48] Griffin, Andrew. Gmail, Google Drive down: many Google services hit by widespread outage. https://goo.gl/DFrM1v. [Online; accessed 13 June 2018].

[49] Gupta, Ankit. 10 years of OneDrive: Microsoft OneDrive celebrates its 10th birthday. https://news.thewindowsclub.com/10-years-of-onedrive-90396/. [Online; accessed 18 June 2018].

[50] Hachman, Mark. Google Drive is being replaced by Backup and Sync: What to expect. https://www.pcworld.com/article/3223136/data-center-cloud/google-drive-is-being-replaced-by-backup-and-sync-what-to-expect.html. [Online; accessed 13 June 2018].

[51] Houston, W. Andrew. Registration statement of Dropbox, Inc. https://www.sec.gov/Archives/edgar/data/1467623/000119312518055809/d451946ds1.htm. [Online; accessed 18 June 2018].

[52] Kincaid, Jason. Dropbox Acquires The Domain Everyone Thought It Had: Dropbox.com. https://techcrunch.com/2009/10/13/dropbox-acquires-the-domain-everyone-thought-it-had-dropbox-com/. [Online; accessed 13 June 2018].

[53] Kincaid, Jason. Dropbox Security Bug Made Passwords Optional For Four Hours. https://techcrunch.com/2011/06/20/dropbox-security-bug-made-passwords-optional-for-four-hours/. [Online; accessed 13 June 2018].

[54] Klabnik, Steve and Nichols, Carol. *The Rust Programming Language*. No Starch Press, 2018.

[55] Klein, Andy. Hard Drive Cost Per Gigabyte. https://www.backblaze.com/blog/hard-drive-cost-per-gigabyte/. [Online; accessed 13 June 2018].

[56] Krumins, Carl. Smartphone Ownership, Usage And Penetration By Country. https://thehub.smsglobal.com/smartphone-ownership-usage-and-penetration. [Online; accessed 13 June 2018].

[57] Kuo, Carbo. BYVoid/gdrive – A user-level filesystem wrapper of Google Drive. https://github.com/BYVoid/gdrive. [Online; accessed 17 June 2018].

[58] Lardinois, Frederic. Google updates Drive with a focus on its business users. https://goo.gl/tUU3CX. [Online; accessed 18 June 2018].

[59] Leighton, Ralph and Feynman, Richard. *Surely You're Joking, Mr. Feynman!* W.W. Norton, 1985.

[60] Leonard, Thomas. Python to OCaml: Retrospective. http://roscidus.com/blog/blog/2014/06/06/python-to-ocaml-retrospective/. [Online; accessed 13 June 2018].

[61] Mendelsohn, Tom. Dropbox hackers stole e-mail addresses, hashed passwords from 68M accounts. https://arstechnica.com/information-technology/2016/08/dropbox-hackers-stole-email-addresses-hashed-passwords-68m-accounts/. [Online; accessed 13 June 2018].

[62] Mihov, Dimitar. Google suffered a meltdown as Gmail, Maps and YouTube went down. https://thenextweb.com/google/2017/09/12/google-down-gmail-youtube-maps/. [Online; accessed 13 June 2018].

[63] Noyes, Katherine. Google Drive for Linux Is on the Way. https://www.pcworld.com/article/254488/google_drive_for_linux_is_on_the_way.html. [Online; accessed 13 June 2018].

[64] Oprea, Dustin. GDriveFS – An innovative FUSE wrapper for Google Drive. https://github.com/dsoprea/GDriveFS. [Online; accessed 17 June 2018].

[65] Oprea, Dustin. GDriveFS documentation. https://github.com/dsoprea/GDriveFS/blob/master/gdrivefs/resources/README.rst. [Online; accessed 17 June 2018].

[66] Perez, Sarah. Google Drive Experiencing Outage. https://techcrunch.com/2013/03/18/google-drive-experiencing-intermittent-issues/. [Online; accessed 13 June 2018].

[67] Pichai, Sundar. Introducing Google Drive. . . yes, really. https://googleblog.blogspot.ro/2012/04/introducing-google-drive-yes-really.html. [Online; accessed 13 June 2018].

[68] Pușcaș, Sergiu. GCSF – a virtual file system based on Google Drive. https://github.com/harababurel/gcsf. [Online; accessed 23 June 2018].

[69] Pușcaș, Sergiu. GCSF Documentation. https://docs.rs/gcsf/latest/gcsf/. [Online; accessed 23 June 2018].

[70] Pușcaș, Sergiu. GCSF Rust Crate. https://crates.io/crates/gcsf. [Online; accessed 23 June 2018].

[71] S2Games. S2Games/drivefs – fuse file system for google drive written in pure go. https://github.com/S2Games/drivefs. [Online; accessed 17 June 2018].

[72] Sahney, Aakash and Loxton, David. Introducing Backup and Sync for Google Photos and Google Drive. https://www.blog.google/products/photos/introducing-backup-and-sync-google-photos-and-google-drive/. [Online; accessed 13 June 2018].

[73] Strada, Alessandro. astrada/google-drive-ocamlfuse – FUSE filesystem over Google Drive. https://github.com/astrada/google-drive-ocamlfuse. [Online; accessed 17 June 2018].

[74] Svetlik, Joe. Google goes down for 5 minutes, Internet traffic drops 40%. https://www.cnet.com/news/google-goes-down-for-5-minutes-internet-traffic-drops-40/. [Online; accessed 13 June 2018].

[75] Tanenbaum, S. Andrew. *Modern Operating Systems*. Pearson Education International, 3rd edition, 2007.

[76] Thiel, Sebastian. A binding and CLI generator for all Google APIs. https://github.com/Byron/google-apis-rs. [Online; accessed 13 June 2018].

[77] Thiel, Sebastian. GitHub profile. https://github.com/Byron. [Online; accessed 13 June 2018].

[78] Voelker, Abe. How long since Google said a Google Drive Linux client is coming. https://abevoelker.github.io/how-long-since-google-said-a-google-drive-linux-client-is-coming/. [Online; accessed 13 June 2018].

[79] Weidenfeld, Dominik. dweidenfeld/plexdrive – mounts your Google Drive FUSE filesystem (optimized for media playback). https://github.com/dweidenfeld/plexdrive. [Online; accessed 17 June 2018].

[80] Woods, Ben. Google Drive and Docs are down for some users, company is investigating. https://thenextweb.com/google/2014/10/27/google-drive-docs-users-company-investigating/. [Online; accessed 13 June 2018].

[81] Woods, Ben. Google Drive and Gmail are down for some users around the world. https://thenextweb.com/google/2016/01/26/google-drive-is-down-for-some-users-around-the-world/. [Online; accessed 21 June 2018].

[82] Yen, Eric. thejinx0r/DriveFS – A google drive fuse filesystem implemented in C++. https://github.com/thejinx0r/DriveFS. [Online; accessed 17 June 2018].

[83] Yen, Eric. thejinx0r/node-gdrive-fuse – a simple filesystem written in NodeJS to mount Google Drive as a local drive. https://github.com/thejinx0r/node-gdrive-fuse. [Online; accessed 17 June 2018].