MegaFS, a FUSE filesystem wrapper for Mega. Part 2: Reading and writing files.

In the previous article, we implemented the beginning of a filesystem where we can list our Mega files and navigate through them. It is now time to read/download them, and create/upload new ones! In order to to that, we need to implement 5 FUSE callbacks:

- *mknod(path, mode, dev)*: called to create a new, empty file ;
- *open(path, flags)*: called to open a file ;
- *read(path, size, offset, fh)*: called to read *size* bytes from a file at a given *offset ;*
- *write(path, buf, offset, fh)*: called to write the contents of *buf* to a file at a given *offset ;*
- *release(path, flags, fh)*: called to release a file (the opposite of *open*, see the FUSE FAQ to know the relation between *release* and the *close* system call).

Reading files

The *read* implementation can be tricky because of the *size* and *offset* parameters. They can take any value, but since the contents of the file contents are encrypted, we can only deal with data starting and ending on an AES block boundary (16 bytes). We could handle that by downloading the blocks containing the requested range (the Mega API supports partial downloads via the HTTP Range header and/or a suffix appended to the download URL), decrypting them and returning only the part really requested.

But for now, we will follow a simpler approach: just download the entire file and store it in a temporary file when *open* is called. Then, we can just seek/read this temporary file whenever *read* is called.

```
def open(self, path, flags):
  if path not in self.files:
    return -errno.ENOENT

  if (flags &amp;amp; 3) == os.O_RDONLY:
    (tmp_f, tmp_path) = tempfile.mkstemp(prefix='megafs')
    os.close(tmp_f)
    if self.client.downloadfile(self.files[path], tmp_path):
      return open(tmp_path, "rb")
    else:
      return -errno.EACCESS
  else:
    return -errno.EINVAL
```

We currently only support files opened in read-only mode (flag O_RDONLY), so we start by checking that (the access mode is stored in the two least significant bits of *flags*, which explains the "*& 3*" mask). We then create the temporary file, download the entire Mega file to it (the *downloadfile* method is pretty much the same as in my first article, with an additional parameter to specify the target path), and check that the download succeeded by checking the meta-MAC (a thing we wouldn't be able to do with partial downloads because we don't know the MACs of each chunk). We finally return a handle to this temporary file, which will be used by *read* and *release*:

```
def read(self, path, size, offset, fh):
  fh.seek(offset)
  return fh.read(size)
```

Our approach makes the *read* implementation very simple: just seek to the desired offset and read the temporary file. We are now almost done, and only need to delete the temporary file when the file is released:

```
def release(self, path, flags, fh):
  fh.close()
  os.unlink(fh.name)
```

That's all! We can now move to the *write* implementation.

Writing files

The writing part is even more tricky than the reading part: we have not only the encryption problem (we can only handle data aligned on AES block boundaries to be able to encrypt it, so we would need to buffer the data), but we also have to know the final size of the file before writing it! It is indeed required by the API '*u*'

method, used to request an upload URL. Moreover, partial uploads must start and end on a chunk boundary (see the Mega developer's guide, section 5 – "File encryption and integrity checking", but this is in fact pretty much the same problem as the encryption one, with largest boundaries).

We will address these problems while keeping our code as simple as possible by using quite the same technique as earlier: we will first write all the data to a temporary file, and then upload it when the file is released. That way, we will know its final size when requesting an upload URL from the API.

Let's start with the changes to the *open* method. The only writing mode allowed is *O_WRONLY | O_CREAT | O_EXCL* (see the Mega developer's guide, section 1.3 – "Storage model"), that is, write-only with creation of the file: we can't write to an existing file, nor can we open a file for both reading and writing. But in fact, *open* is not in charge of creating the file: a call to *open(path, flags | O_CREAT)* will be turned into a call to *mknod* (to create the file) followed by a call to *open(path, flags)*. Thus, we have to implement *mknod*:

```
def mknod(self, path, mode, dev):
  if path in self.files:
    return -errno.EEXIST

  dirname, basename = os.path.split(path)
  self.files[dirname]['children'].append(basename)
  self.files[path] = {'t': 0, 'ts': int(time.time()), 's': 0}
```

When *mknod(path)* is called, we add a new, empty file to our filesystem at *path*, so that it can be found by the *getattr* and *open* methods (they are called just after *mknod* and need to see that the file has been created). It obviously has no Mega ID ('*h*' field), and this will enable us to distinguish these "special" files from the others. We can now make changes to *open* to handle openings in write-only mode:

```
def open(self, path, flags):
  if path not in self.files:
    return -errno.ENOENT

  if (flags &amp;amp; 3) == os.O_RDONLY:
    (tmp_f, tmp_path) = tempfile.mkstemp(prefix='megafs')
    os.close(tmp_f)
    if 'h' not in self.files[path]:
      return open(tmp_path, "rb")
    elif self.client.downloadfile(self.files[path], tmp_path):
      return open(tmp_path, "rb")
    else:
      return -errno.EACCESS
  elif (flags &amp;amp; 3) == os.O_WRONLY:
    if 'h' in self.files[path]:
```

```
      return -errno.EEXIST
    (tmp_f, tmp_path) = tempfile.mkstemp(prefix='megafs')
    os.close(tmp_f)
    return open(tmp_path, "wb")
  else:
    return -errno.EINVAL
```

For the *O_RDONLY* case, we just add a condition to check that the file is a "real" file (with a Mega ID, and not an empty file created by *mknod*) before downloading it.

For the *O_WRONLY* case, we first check that the file is an empty file created by *mknod*: if it's not the case, that means that we are trying to write to an existing file, and that is not allowed by the API. Then we create the temporary file we are going to write all the data into, and we return a handle to this file that will be used by *write* and *release*:

```
def write(self, path, buf, offset, fh):
  fh.seek(offset)
  fh.write(buf)
  return len(buf)
```

As for the *read* method, our approach makes the *write* implementation very simple: just seek to the desired offset and write the contents of the buffer to the temporary file.

Now, let's add some code to the *release* method to effectively upload the file when we are done writing to it:

```
def release(self, path, flags, fh):
  fh.close()
  if fh.mode == "wb":
    dirname, basename = os.path.split(path)
    uploaded_file = self.client.uploadfile(fh.name, self.files[dirname]['h'], basename)
    if 'f' in uploaded_file:
      uploaded_file = self.client.processfile(uploaded_file['f'][0])
      self.files[path] = uploaded_file
  os.unlink(fh.name)
```

If the temporary file was opened for writing, we upload it and create the new node on Mega. The *uploadfile* method is the same as in my first article, with an additional parameter to specify the source path. It returns the information on the new node, but with its key and attributes still encrypted. Thus, we give it to a *processfile* method that is just the main loop of the *getfiles* method (from the first article) extracted to a method (you can find the complete code on GitHub):

```
def getfiles(self):
  files = self.api_req({'a': 'f', 'c': 1})
  files_dict = {}
  for file in files['f']:
    files_dict[file['h']] = self.processfile(file)
  return files_dict

def processfile(self, file):
  if file['t'] == 0 or file['t'] == 1:
    key = file['k'][file['k'].index(':') + 1:]
    key = decrypt_key(base64_to_a32(key), self.master_key)
    if file['t'] == 0:
      file['k'] = (key[0] ^ key[4], key[1] ^ key[5], key[2] ^ key[6], key[3] ^ key[7])
      file['iv'] = key[4:6] + (0, 0)
      file['meta_mac'] = key[6:8]
    else:
      file['k'] = key
    attributes = base64urldecode(file['a'])
    attributes = dec_attr(attributes, file['k'])
    file['a'] = attributes
  elif file['t'] == 2:
    self.root_id = file['h']
    file['a'] = {'n': 'Cloud Drive'}
  elif file['t'] == 3:
    self.inbox_id = file['h']
    file['a'] = {'n': 'Inbox'}
  elif file['t'] == 4:
    self.trashbin_id = file['h']
    file['a'] = {'n': 'Rubbish Bin'}
  return file
```

We are now done! Let's test the new version of our filesystem:

```
julienm@rchand:~$ python MegaFS/megafs.py ~/megafs
Email [go.julienm@gmail.com]:
Password:
julienm@rchand:~$ cd megafs/Cloud\ Drive/
julienm@rchand:~/megafs/Cloud Drive$ cp ~/MegaFS/megafs.py .
julienm@rchand:~/megafs/Cloud Drive$ head megafs.py
from megaclient import MegaClient
```

```
import errno
import fuse
import getpass
import os
import stat
import tempfile
import time

fuse.fuse_python_api = (0, 2)
julienm@rchand:~/megafs/Cloud Drive$ tail megafs.py

if __name__ == '__main__':
  email = raw_input("Email [%s]: " % getpass.getuser())
  if not email:
    email = getpass.getuser()
  password = getpass.getpass()
  client = MegaClient(email, password)
  fs = MegaFS(client)
  fs.parse(errex=1)
  fs.main()
julienm@rchand:~/megafs/Cloud Drive$ ls -l megafs.py
-rw-rw-rw- 1 root root 3821 févr.  2 22:42 megafs.py
julienm@rchand:~/megafs/Cloud Drive$ cd
julienm@rchand:~$ sudo umount ~/megafs
julienm@rchand:~$
```

In the next article, we will see how to rename/move files, change their attributes, and delete them. Meanwhile, you can find the complete source code on the MegaFS project page on GitHub: https://github.com/CyberjujuM/MegaFS!

This entry was posted in Uncategorized on February 2, 2013 [/web/20140402040809/http://julien-marchand.fr/blog/megafs-a-fuse-filesystem-wrapper-for-mega-part-2-reading-and-writing-files/] .

10 thoughts on “MegaFS, a FUSE filesystem wrapper for Mega. Part 2: Reading and writing files.”

eternauta2001
February 2, 2013 at 11:35 pm

You rock!

Davide
February 4, 2013 at 10:49 am

A huge THANKS!
Very interesting, very useful.
Can’t wait for the mkdir command…

Laurens
February 4, 2013 at 3:46 pm

Great articles! Very interesting & really filling a need

There’s some cases where & is listed in the examples above, fyi.

ProlyX
February 4, 2013 at 9:52 pm

Nice article,
waiting for mkdir aswell – tried to build it myself without success currently.

While i tried to make a folder i created a non-existing folder, pretty weird… to get MegaFs working again with my account i had to modify megafs.py:
line:48
path = self.getpath(files, files[hash]['p']) + “/” + files[hash]['a']['n']

to smt like:
if files[hash]['a'] == False:
return ”
path = self.getpath(files, files[hash]['p']) + “/” + files[hash]['a']['n']

ProlyX
February 5, 2013 at 12:37 am

ok got it..
megaclient.py:

```
def createdir(self, target, dirname):
key = [random.randint(0, 0xFFFFFFFF) for _ in xrange(4)]
attributes = {'n': dirname}
enc_attributes = enc_attr(attributes, key)
return self.api_req({'a': 'p', 't': target, 'n': [{'h': 'xxxxxxxx', 't': 1, 'a': base64urlencode(enc_attributes), 'k':
a32_to_base64(encrypt_key(key, self.master_key))}]})
```

megafs.py

```
def mkdir (self, path, mode):
dirname, basename = os.path.split(path)
created_folder = self.client.createdir(self.files[dirname]['h'], basename)
if 'f' in created_folder:
created_folder = self.client.processfile(created_folder['f'][0])
self.files[dirname]['children'].append(basename)
self.files[path] = created_folder
self.files[path]['children'] = []
```

sorry for the poor code, never used python before..

ProlyX
February 5, 2013 at 12:39 am

uhm ok code view isnt very good it here, for correct intends check http://pastebin.com/BF2uFY0y

---

kionez
February 5, 2013 at 9:31 pm

Hi, I've created a small patch with your mkdir code and a workaround to handle shared directories: https://github.com/CyberjujuM/MegaFS/issues/2

k.

Julien Marchand Post author
February 6, 2013 at 12:25 am

Hey! Sorry for the lack of updates, I've just started my internship @Google and have been a bit busy these days

Thanks a lot kionez & ProlyX for your contributions! I'll add mkdir, rename and unlink tomorrow, based on that.

---

wiwi60
March 4, 2013 at 3:39 pm

how i can test MegaFS on windows ? Please!!

Ingo Marsch
April 7, 2013 at 7:42 am

Using megafs on a empty mega-Account, I get the following error message

Traceback (most recent call last):
File “MegaFS/megafs.py”, line 130, in
fs = MegaFS(client)
File “MegaFS/megafs.py”, line 24, in __init__
path = self.getpath(files, file_h)
File “MegaFS/megafs.py”, line 40, in getpath
path = self.getpath(files, files[hash]['p']) + “/” + files[hash]['a']['n']
TypeError: ‘bool’ object has no attribute ‘__getitem__’