COMPSCI 340 / SOFTENG 370
Operating Systems
Assignment 2 - User space file system
Worth 10%
Due date: 09:30pm 8th of October 2020

Introduction
These are the specifications for the assignment. In week 7, the FUSE (file system in user space) library will be introduced and explained. In this assignment you have to first of all work with an existing user space file system. Then you need to write your own. You can start working on this now using A2_Help slides that can be downloaded from Canvas.

Setup
Do the assignment either on Ubuntu in the labs or on your own machine. You can use virtual machines, macOS or Windows Subsystem for Linux version 2. If you have difficulty or are not able to work with above mentioned implementations; as a result of COVID'19, the image has also been made available at FlexIT (`flexit.auckland.ac.nz`). You'd need to sign in to SSO which will take you to the sign-in page for FlexIT. If you succeed with the access, look for the Ubuntu 18.04 desktop icon. The markers will use Ubuntu in the labs.

Download the files `fuse.py`, `passthrough.py`, `memory.py` and `a2fuse1.py` from the A2 files section of Canvas into a directory.
`memory.py` originally came from
`https://github.com/fusepy/fusepy/blob/master/examples/memory.py`
`fuse.py` originally came from `https://github.com/terencehonles/fusepy`
`passthrough.py` originally came from
`https://github.com/skorokithakis/python-fuse-sample`

Part 1
Make two directories, one called `source` and one called `mount` in the directory. Put some files in the `source` directory (download the files `one`, `two`, and `three` from Canvas).

You will need two terminal windows open: one to run the user space file system and display the work it is doing, and one to work with files from the command line. I will refer to these as terminal one and terminal two.

In terminal one run the program: `python a2fuse1.py source mount`

In terminal two do:
```
ls -l source
ls -l mount
```

For all the questions put the answers (and requested output) into a file called `A2.txt`.

Question 1
Explain the output you have just seen in terminal two. What did you see and why was it like that?
[2 marks]

Question 2
For each of the following commands you perform in terminal two, copy the output generated by the user space file system in terminal one into your answer file and explain each method called. You can get some information from the Python documentation and more using `man`.

Here is the set of commands to be performed:

```
cd mount
cat > newfile
hello world
^D  (this is control-D)
cd ../    (move up out of that directory)
fusermount -u mount (Ubuntu) or umount mount  (macOS)
```

Check the contents of the source and mount directories.

[6 marks]

Example:

```
DEBUG:fuse.log-mixin:-> getattr /newfile (None,)
DEBUG:fuse.log-mixin:<- getattr {'st_atime': 1599037397.463398, 'st_ctime':
1599037379.543504, 'st_gid': 20, 'st_mode': 33188, 'st_mtime':
1599037379.543504, 'st_nlink': 1, 'st_size': 12, 'st_uid': 501}
```

Gets attributes for "newfile".

Part 2

The `Operations` class in `fuse.py` is the one which does the work we are interested in. This is subclassed in both `passthrough.py` and `memory.py`.
As we have seen `passthrough.py` provides a copy of one directory mounted in a different location and passes all requests back to the original directory. Whereas `memory.py` implements an entirely separate file system in memory. This means when the file system is shut down those files are lost.

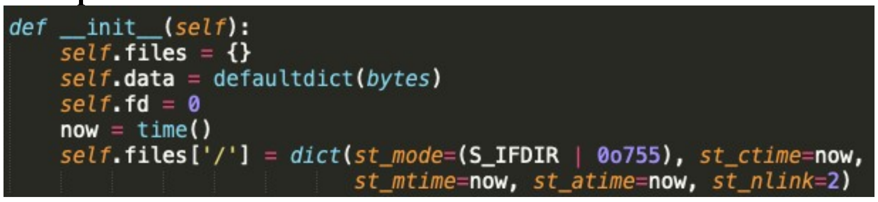
Question 3

Answer these questions about `memory.py`.
For the following list of methods in the `Memory` class explain exactly what each method does.
Include a statement by statement explanation.

```
__init__, getattr, readdir
open, create, unlink
write, read
```

[6 marks]

Example:



```
def __init__(self):
    self.files = {}
    self.data = defaultdict(bytes)
    self.fd = 0
    now = time()
    self.files['/'] = dict(st_mode=(S_IFDIR | 0o755), st_ctime=now,
                           st_mtime=now, st_atime=now, st_nlink=2)
```

Creates an empty dictionary self.files for the files. This will use the path names as the keys. Each value in the dictionary will be another dictionary.
self.data is a dictionary for the files' data. The path names are the keys. The values are the data of that file.
Sets the starting value for the file descriptors, these are going to be used as unique file identifiers.
Grabs the current time and sets the file attributes for the root of this file system. It is a directory, with creation, modified and accessed times set to now. It has two links.

Part 3
You now have to create your own user space file system. It works a little bit like a combination of `passthrough.py` and `memory.py`. Call your program `a2fuse2.py`. You can subclass `Passthrough` or `Memory` if you want. You will probably have to implement at least the same methods you described in Question 3 to enable the following command.

Run your program with `python a2fuse2.py source1 source2 mount`

Settings: Create two source directories, `source1` and `source2`. Create file `one` in `source1`, and create file `two` in `source2` respectively. Make sure that `mount` is initially empty. The file system works very much like `memory.py` but it starts with some real files from the `source1` and `source2` directories. So, the file system has two classes of files which are in the `mount` directory. One consists of real files from the `source1` and `source2` directories and the other of files which only exist in memory. Specifically, create your FUSE to meet the following two requirements.

Requirement 1: Enable your FUSE to mount two source directories into a mount point.
Requirement 2: Any changes which happen to files in the `mount` directory which have been created only in memory (including creating or deleting files, or writing to files) only happen in the `mount` directory. However, if the file was initially in the `source1` and `source2` directories then changes get passed back to that directory just as with `passthrough.py`.

Example:
Start your FUSE in terminal one: `python a2fuse2.py source1 source2 mount`
Start in the same directory in terminal two.

```
------------------------------------------------------------------------------------------------
(base) ubuntu@q2b:~/part3$ ls -l source1
total 4
-rw-r--r-- 1 ubuntu ubuntu 5 Sep  2 05:12 one
(base) ubuntu@q2b:~/part3$ ls -l source2
total 4
-rw-r--r-- 1 ubuntu ubuntu 10 Sep  2 05:12 two
(base) ubuntu@q2b:~/part3$ cd mount/
(base) ubuntu@q2b:~/part3/mount$ ls -l
total 0
-rw-r--r-- 1 ubuntu ubuntu  5 Sep  2 05:12 one
-rw-r--r-- 1 ubuntu ubuntu 10 Sep  2 05:12 two

------------------------------------------------------------------------------------------------
(base) ubuntu@q2b:~/part3/mount$ cat one two > three
(base) ubuntu@q2b:~/part3/mount$ ls -l
total 0
-rw-rw-r-- 1 ubuntu ubuntu 15 Sep  2 05:16 three
-rw-r--r-- 1 ubuntu ubuntu  5 Sep  2 05:12 one
-rw-r--r-- 1 ubuntu ubuntu 10 Sep  2 05:12 two
(base) ubuntu@q2b:~/part3/mount$ ls -l ../source1
total 4
-rw-r--r-- 1 ubuntu ubuntu 5 Sep  2 05:12 one
(base) ubuntu@q2b:~/part3/mount$ ls -l ../source2
total 4
-rw-r--r-- 1 ubuntu ubuntu 10 Sep  2 05:12 two

------------------------------------------------------------------------------------------------
(base) ubuntu@q2b:~/part3/mount$ rm three
(base) ubuntu@q2b:~/part3/mount$ ls -l
total 0
-rw-r--r-- 1 ubuntu ubuntu  5 Sep  2 05:12 one
-rw-r--r-- 1 ubuntu ubuntu 10 Sep  2 05:12 two
```

For working correctly with `cat, ls, rm` on a variety of files. Files in the `source1` and `source2` directories can be modified but any new files created only exist in the `mount` directory (in memory).

[12 marks]

Submission

Use the Canvas submission system to submit your assignment. Zip together A2.txt and a2fuse2.py.

Extra marks

1 mark for including your name and login in both files.
1 mark for any files created by the file system having the correct user and group ids.

[2 marks]

Hints:

1. You can put logging output directly into your code for debugging purposes, as long as you model your code on that in a2fuse1.py. E.g. logging.debug("whatever you want to print") this will then appear as output in terminal one.
2. To make this assignment easier it only tests positively. i.e. Any command executed by the markers will only be ones that should execute without causing an error. E.g. You do not need to worry about files not existing or having the wrong privileges. You do not need to worry about symbolic links. You do not need to worry about sparse files. You do not need to consider nested directories.
3. For those of you who have never programmed in Python, feel free to come for help or ask on Piazza. The language itself is simple but learning the libraries (or modules as they are called in Python) requires time. Google and StackOverflow are really helpful here and you will eventually become confident with the Python documentation https://docs.python.org/2/.