SOFTENG 370: Assignment 2

Kinan Cheung | kche356 | 495324665

# Question 1

cat > hello

DEBUG:fuse.log-mixin:-> getattr /hello (None,)

DEBUG:fuse.log-mixin:<- getattr '[Errno 2] No such file or directory'

DEBUG:fuse:FUSE operation getattr raised a <class 'fuse.FuseOSError'>, returning errno 2.

Traceback (most recent call last): ...

DEBUG:fuse.log-mixin:-> create /hello (33188,)

DEBUG:fuse.log-mixin:<- create 1

DEBUG:fuse.log-mixin:-> getattr /hello (1,)

DEBUG:fuse.log-mixin:<- getattr {'st\_mode': 33188, 'st\_nlink': 1, 'st\_size': 0, 'st\_ctime': 1620343432.740972, 'st\_mtime': 1620343432.7409723, 'st\_atime': 1620343432.7409723}

DEBUG:fuse.log-mixin:-> flush /hello (1,)

DEBUG:fuse.log-mixin:<- flush 0

*getattr /hello (None,)* gets the file attributes associated with *hello* but outputs *get attr ‘[Errno 2] No such file or directory’* as the file does not exist. This raises the *FUSE* error *ENOENT* which displays the previous calls.

*create /hello (33188,)* creates the file *hello* with a mode of *33188*. This is confirmed by *create 1* where *1* is the file ID

*getattr /hello (1,)* gets the dictionary output of *hello* as shown by the values after. This is where *st\_mode­* is the file access mode, *st\_nlink* is the number of hard links, *st\_size* is the total size of the file in bytes, *st\_ctime* is the creation time, *st\_mtime* is the modified time and *st\_atime* is the last accessed time.

*flush /hello (1,)* flushes the internal buffer for the file and *flush 0* indicates there are no errors.

hello world

DEBUG:fuse.log-mixin:-> getxattr /hello ('security.capability',) DEBUG:fuse.log-mixin:<- getxattr '' DEBUG:fuse.log-mixin:-> write /hello (b'hello world\n', 0, 1) DEBUG:fuse.log-mixin:<- write 12

*getxattr /hello (‘security.capability’,)* tries to get the security information for file *hello* and is successful by returning an empty output.

*write /hello (b’hello world\n’, 0, 1)* writes *hello world* into the file and the size of the input of the message is returned at 12 bytes as shown by *write 12.*

^D

DEBUG:fuse.log-mixin:-> flush /hello (1,)

DEBUG:fuse.log-mixin:<- flush 0

DEBUG:fuse.log-mixin:-> release /hello (1,)

DEBUG:fuse.log-mixin:<- release 0

*flush /hello (1,)* flushes the internal buffer for the file and *flush 0* indicates there are no errors.

*release /hello (1,)* releases the file descriptor for */hello* and outputs *release 0* indicating there are no errors

ls -al

DEBUG:fuse.log-mixin:-> opendir / ()

DEBUG:fuse.log-mixin:<- opendir 0

DEBUG:fuse.log-mixin:-> readdir / (0,)

DEBUG:fuse.log-mixin:<- readdir ['.', '..', 'hello']

DEBUG:fuse.log-mixin:-> getattr / (None,)

DEBUG:fuse.log-mixin:<- getattr {'st\_mode': 16877, 'st\_ctime': 1620344034.604667, 'st\_mtime': 1620344034.604667, 'st\_atime': 1620344034.604667, 'st\_nlink': 2}

DEBUG:fuse.log-mixin:-> getxattr / ('security.selinux',)

DEBUG:fuse.log-mixin:<- getxattr ''

ERROR:fuse:Uncaught exception from FUSE operation getxattr, returning errno.EINVAL.

Traceback (most recent call last): ...

DEBUG:fuse.log-mixin:-> getattr /hello (None,)

DEBUG:fuse.log-mixin:<- getattr {'st\_mode': 33188, 'st\_nlink': 1, 'st\_size': 12, 'st\_ctime': 1620344041.600293, 'st\_mtime': 1620344041.6002932, 'st\_atime': 1620344041.6002932}

DEBUG:fuse.log-mixin:-> getxattr /hello ('system.posix\_acl\_access',)

DEBUG:fuse.log-mixin:<- getxattr ''

DEBUG:fuse.log-mixin:-> releasedir / (0,)

DEBUG:fuse.log-mixin:<- releasedir 0

*opendir / ()* opens the current root directory and returns *opendir 0* which means it was a successful operation.

*readdir / (0,)* reads the files in the root directory and returns *readdir [‘.’, ‘..’, ‘hello’]* where *.* is the directory entry for the current directory, *..* is the directory entry for the directory one level higher in the hierarchy (*A2)* and *hello* is the file that was just created.

*getattr / (None)* gets the file attributes associated with the root directory and returns a dictionary output as shown by values after. This is where *st\_mode­* is the file access mode, *st\_ctime* is the creation time, *st\_mtime* is the modified time and *st\_atime* is the last accessed time and *st\_nlink”* is the number of hard links.

*getxattr / (‘security.selinux’,)* tries to get the security for SELinux context information for the root directoryand is successful by returning an empty output. From this operation, a *FUSE* error *EINVAL* is raised due to an invalid argument and displays the previous calls.

*getattr /hello (None,)* gets the dictionary output dictionary of *hello* as shown by the values after. This is where *st\_mode­* is the file access mode, *st\_nlink* is the number of hard links, *st\_size* is the total size of the file in bytes, *st\_ctime* is the creation time, *st\_mtime* is the modified time and *st\_atime* is the last accessed time.

*getxattr /hello (‘system.posix\_acl\_access’,)* tries to get the POSIX ACLs context information for *hello* and succeeds as shown by the empty return value.

*releasedir / (0,)* releases the current directory and is done successfully as shown by *releasedir 0.*

rm hello

DEBUG:fuse.log-mixin:-> getattr / (None,)

DEBUG:fuse.log-mixin:<- getattr {'st\_mode': 16877, 'st\_ctime': 1620344034.604667, 'st\_mtime': 1620344034.604667, 'st\_atime': 1620344034.604667, 'st\_nlink': 2}

DEBUG:fuse.log-mixin:-> getattr /hello (None,)

DEBUG:fuse.log-mixin:<- getattr {'st\_mode': 33188, 'st\_nlink': 1, 'st\_size': 12, 'st\_ctime': 1620344041.600293, 'st\_mtime': 1620344041.6002932, 'st\_atime': 1620344041.6002932}

DEBUG:fuse.log-mixin:-> access /hello (2,)

DEBUG:fuse.log-mixin:<- access 0

DEBUG:fuse.log-mixin:-> unlink /hello ()

DEBUG:fuse.log-mixin:<- unlink None

*getattr / (None)* gets the file attributes associated with the root directory and returns a dictionary output as shown by values after. This is where *st\_mode­* is the file access mode, *st\_ctime* is the creation time, *st\_mtime* is the modified time and *st\_atime* is the last accessed time and *st\_nlink”* is the number of hard links.

*getattr /hello (None,)* gets the dictionary output of *hello* as shown by the values after. This is where *st\_mode­* is the file access mode, *st\_nlink* is the number of hard links, *st\_size* is the total size of the file in bytes, *st\_ctime* is the creation time, *st\_mtime* is the modified time and *st\_atime* is the last accessed time.

*access /hello (2,)* checks whether the current calling process can access the file */hello* and succeeds as shown by *access 0*

*unlink /hello ()* then deletes the file */hello* from the current directory and succeeds as shown by *unlink None*

# Question 2

Python has a module called *os* which includes functions *getuid()* and *getgid()*. These 2 functions get the true user ids and group ids of the users creating the files. If you wanted to show this in memory.py, you would set *st\_uid*  and *st\_gid* ­meta-data in createwith the *os* functions as the input parameters.

# Question 3

As the *access()* function is not implemented within *memory.py,* the default method from *fuse.py* is used instead. This method in *fuse.py* has a default return value of *0* which allows any user to access/ work with files. Hence the ordinary user can work with the file *hello.*

# Question 4

To keep track of free blocks, in block 0 which holds the root meta-data, there are 4 bytes allocated to two bitmaps. These bitmaps are essentially integers which are used to check whether blocks are occupied by using bit manipulation. The helper class *bit.py* hold the functions to change a specific bit to 1 (indicating the block is not free) or 0 (indicating the block is free). Looping through and checking each value allows me to determine if a block is free or not. One bitmap holds the information for all blocks, including meta-data and data which is to find any free block, while the other bitmap only records meta-data which is used for reading directories.

# Question 5

To keep track of blocks allocated to a file, the first byte in each file meta data holds a value pointing to the block with data in it. The first byte of these blocks also holds a value pointing to another block if data that belongs to that file is in it. If the value is not 0, the pointer value is used to read the next block in the file, and this is repeated until there are no more blocks.

# Question 6

When the file name/ path is given as a parameter input, I manually search through the disk by reading the bitmap that holds every block. I read the specific bytes in each block in which the name of a file is located and match it with the path. If the block name matches the path, in that block is the meta-data/ attributes for the files to return.