# 340 Assignment 2 Write Up

Name: Etienne Naude

UPI : enau831

ID : 768485633

## Part 1

### Question 1

`ls -l source`

```
total 12
-rw-rw-r-- 1 etinaude etinaude 31 Sep 28 11:16 one
-rw-rw-r-- 1 etinaude etinaude 51 Sep 28 11:16 three
-rw-rw-r-- 1 etinaude etinaude 41 Sep 28 11:16 two
```

`ls -l mount`

```
total 0
-rw-rw-r-- 1 etinaude etinaude 31 Sep 28 11:16 one
-rw-rw-r-- 1 etinaude etinaude 51 Sep 28 11:16 three
-rw-rw-r-- 1 etinaude etinaude 41 Sep 28 11:16 two
```

`ls` lists files which match certain parameters and locations, in this case the first command listed those found in the source directory and the second in the mount directory.

the `-l` option shows a more detailed view of the files, it shows the permissions, number of links, owner, groups, file size, last edits and the name of the file.

The total at the top of the ls output shows the total number of blocks which are used to store the data on the disk.

`ls mount` shows 0 because the files are not being stored in mount but are stored in source and links created to mount, therefore they aren't taking up any blocks in mount.

### Question 2

```
DEBUG:fuse.log-mixin:<- release None
DEBUG:fuse.log-mixin:-> getattr /newfile (none,)
DEBUG:fuse.log-mixin:<- getattr "[Errno 2] No such file or directory: 'source/newfile'"
DEBUG:fuse.log-mixin:-> create /newfile (33204L,)
DEBUG:fuse.log-mixin:<- create 4
DEBUG:fuse.log-mixin:-> getattr /newfile (none,)
DEBUG:fuse.log-mixin:<- getattr {'st_atime': 1601249824.6497111, 'st_ctime': 1601249831.9377408,
'st_gid': 1000,
'st_mode': 33204, 'st_mtime': 1601249831.9377408, 'st_nlink': 1, 'st_size': 12, 'st_uid': 1000}
DEBUG:fuse.log-mixin:-> flush /newfile (4L,)
DEBUG:fuse.log-mixin:<- flush None

DEBUG:fuse.log-mixin:-> getxattr /newfile (u'security.capability',)
DEBUG:fuse.log-mixin:<- getxattr '[Errno 95] Operation not supported'
DEBUG:fuse.log-mixin:-> write /newfile (b'hello world\n', 0, 4L)
DEBUG:fuse.log-mixin:<- write 12
```

```
DEBUG:fuse.log-mixin:-> flush /newfile (4L,)
DEBUG:fuse.log-mixin:<- flush None
DEBUG:fuse.log-mixin:-> release /newfile (4L,)
DEBUG:fuse.log-mixin:<- release None
```

Note:
for ease of reading I removed the error stack traces but left in the first line to signify the error.

First it tries to read the attributes of newfile but since it doesn't exist this fails, so instead it creates the newfile and then tries again to the get attributes (successfully this time). An explanation of the attributes is below.

Then when the content is added it tries to get a specific attribute but fails, it then writes hello world to the file and closes the file.

- `st_atime` - this is the time the file was last accessed
- `st_ctime` - this is the time the files contents or attributes were last changed
- `st_gid` - this is the group identifier of the file
- `st_mode` - this is the mode of the file, which includes permissions and other details
- `st_mtime` - this is the time the file contents were lsat changed
- `st_nlink` - this is the number of direct links the file has
- `st_size` - this is the size of the file
- `st_uid` - this is the unique identifier of the file

The mount directory is empty but the source folder has a the 4 files newfile, one, two and three.

## Part 2

### Question 3

`def __init__(self):`

Initializes the object and makes a blank dictionary to hold the files which holds a dictionary as the value while the file path as the key.
`__init__` also creates a dictionary named data which also uses pathname as the key but uses the files data as the value.
It also initializes the file descriptor to 0.
Lastly the init creates the root directory by adding the key "/" in files with a dictionary value and the current time for all time variables.

`def create(self, path, mode):`

This creates a new file by adding it to the files dictionary with the specified path for the key and sets the mode to the mode passed to the method and the current time for all the time variables.

`def getattr(self, path, fh=None):`

This returns an item from the files dictionary if it exists else it raises an error. (the file dictionary holds the attributes for the file)

`def open(self, path, flags):`

Open simply increases the unique file descriptor and returns it.

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

This reads data from a file starting at the offset and ending at offset+length.
It does this by using the file path as a key for the data dictionary and using the offset as indices

`def readdir(self, path, fh):`

This reads the contents of a directory

`def unlink(self, path):`

This simply removes an item from the files dictionary but not the data dictionary so it the data is still able to be read but there is no record of it or it's attributes.

`def write(self, path, data, offset, fh):`

This appends a the value of the file in the data dictionary, it then also updates the length of the file in the files dictionary and returns the length of the new data.

## Part 3

The majority of this part was making os calls to ensure a file existed and to find where it existed.