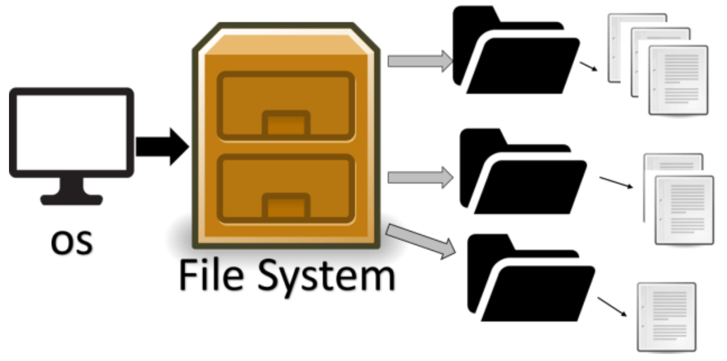
CS340 A2
User space file systems

File Systems

File systems are simply different ways of **organizing** and **storing files** on a **storage device.**

It needs to satisfy these general requirements.



- We need some way of storing information
  - independently from a running program
  - so it can be used at a later time
  - permanently (or an approximation to it)
  - non-volatile storage
  - so it can be shared with other programs or users
- An infinite variety of data is to be stored
  - The more information the OS knows about the data the more it can facilitate use of the data.
  - e.g. Executable files
- Naming the data
  - The data needs to be stored and retrieved easily. We need a way to name the data.
  - We must then be able to locate the data using its name.

File Operations

On most systems these commands need security authorisation to perform and they work on the file as a whole.

Create

- Need to specify information about the file:
  - the name
  - the file type (or some representation of the program associated with this file)
- Do we need to specify the size of the file? (certainly helps with keeping storage contiguous but is usually regarded as an unnecessary restriction)
- Creation needs to do something to the associated device - at least write to some structure (sometimes a directory).
- Some systems allow transitory files to not be recorded permanently in secondary storage.

File Operations

Delete

- Remove the file (or the directory entry). Release the space used by the file, so it could be reused by other files.

Move

- Moving a file can be performed in different ways depending on the before and after locations.
- If both locations are on the same device the data doesn’t have to be copied and then the original deleted. Instead change information about the file.

Open

- The OS evaluates the name, check the access permission.

File Operations

Read/Write

These operations work on the file's contents.

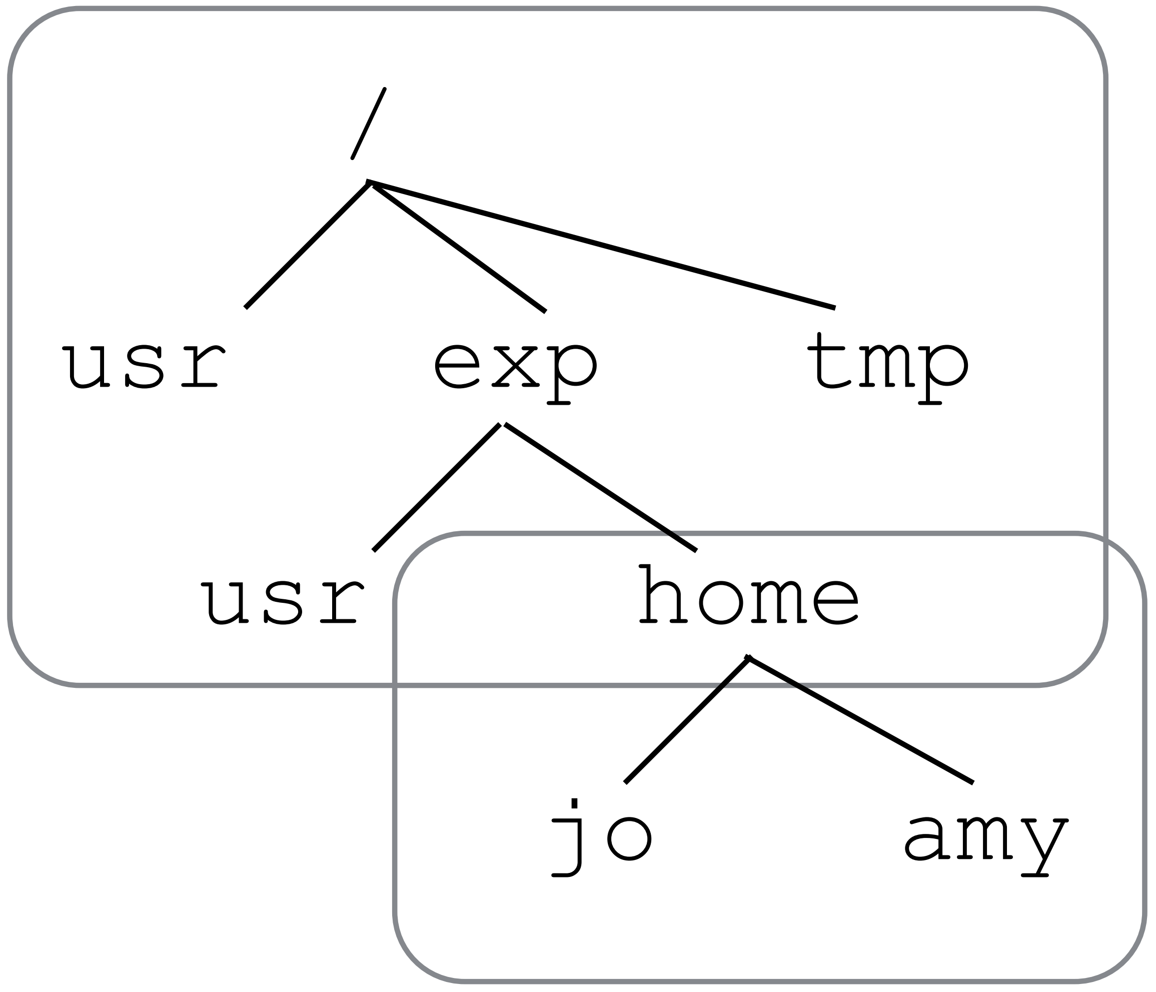
- We need to know where the information is on the device.
- Must specify what data to read/write, how much, and where to put it.
- Write requires the allocation of extra space.

File system in USEr space - FUSE

- Linux has a library which allows file systems to be written and used without root privileges - libfuse.
- The library is used in conjunction with a kernel module which provides the privileged operations.
- The user’s file system gets mounted on to an existing directory.

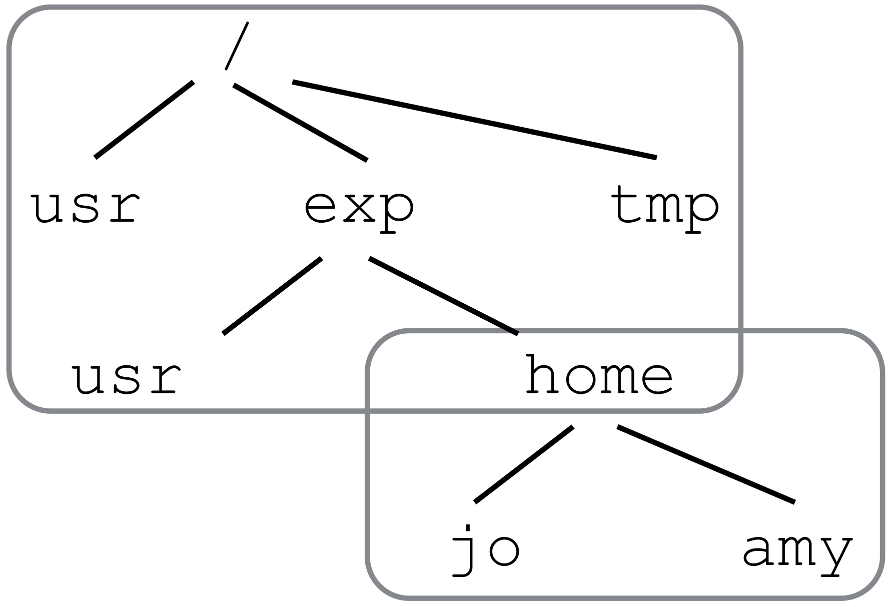
Mounting a file system

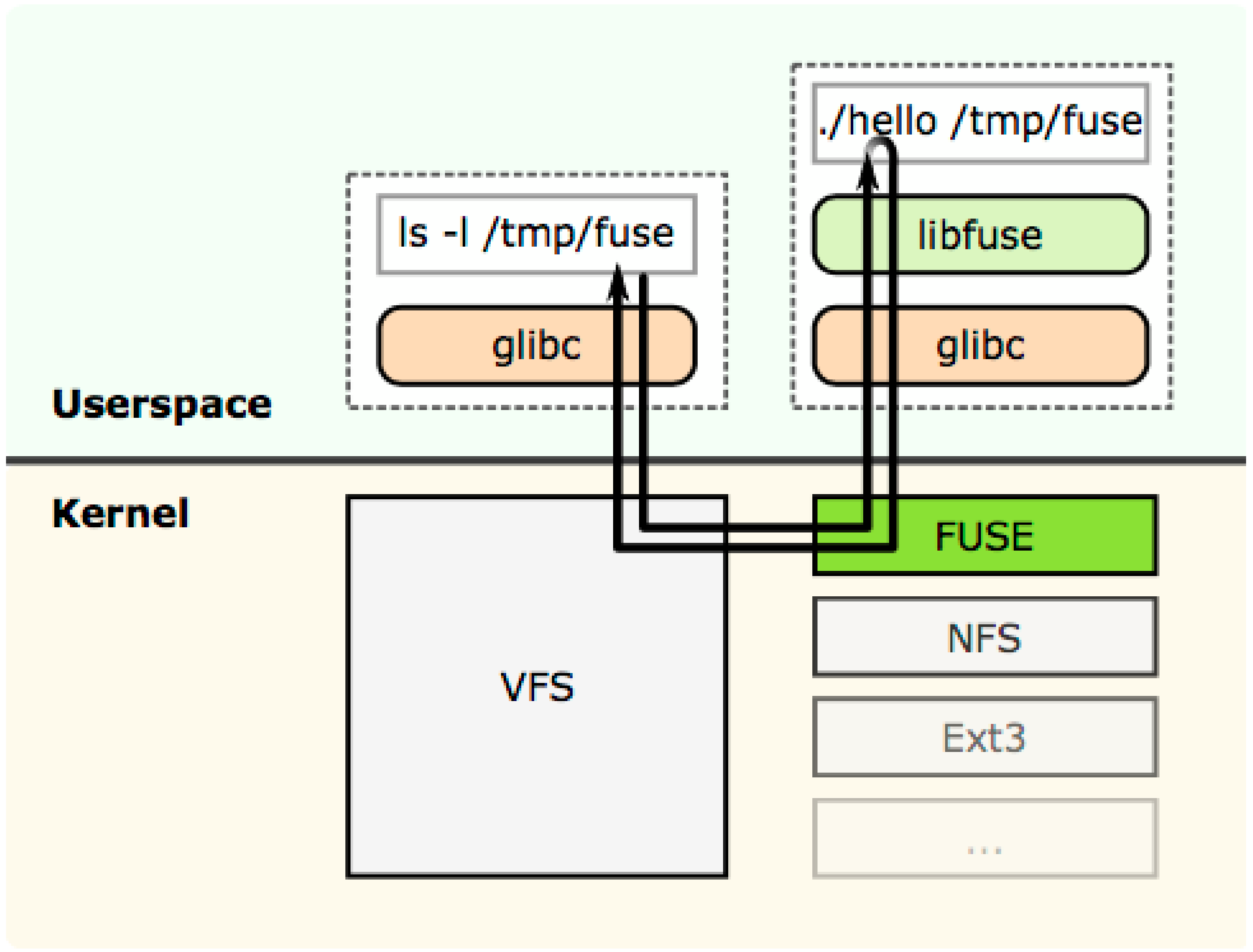
- A file system (in Unix context) can be thought of as a device or partition which can be connected into the standard hierarchical file system starting from the root "/".



Mounting a file system

- We need a mount point: a directory where we plant the file system
    - e.g., `/exp/home`
- Any files which are already in the mount point directory then disappear
- They get replaced by the files/directories in the new file system





How it works

from Wikipedia

`./hello` is mounted on `/tmp/fuse`

All the operations on `/tmp/fuse` and its files can be handled by user level code.

How it works

- Need a handler program which gets linked to the FUSE library.
- The handler program specifies how the files respond to read/write/stat requests.
- At the time of mounting the new file system (handler gets registered with the kernel).
- Mainly useful for virtual file systems.

Setup

- Ubuntu in the labs or on your own machine.
  - Virtual machines, macOS or Windows Subsystem for Linux version 2
  - Ubuntu 18.04 image has been made available at FlexIT (flexit.auckland.ac.nz)
    - Sign in to SSO which will take you to the sign-in page for FlexIT
    - Look for the Ubuntu 18.04 desktop icon

- The Markers will use Ubuntu in the labs.

Setup

- Download fuse.py, memory.py and passthrough.py from the A2 files section on Canvas.
    - fuse.py is from
        - https://github.com/fusepy/fusepy/blob/master/fuse.py
    - memory.py is from
        - https://github.com/fusepy/fusepy/blob/master/examples/memory.py
    - passthrough.py is from
        - https://github.com/skorokithakis/python-fuse-sample

- You can also get examples from the GitHub site at
    - https://github.com/fusepy/fusepy/tree/master/examples

OMG Python

- Much (much) simpler to do this assignment in Python.
- If you don't know how to do something in Python.
  - Ask one of the tutors/lecturers for help
  - Forum: feel free to ask on Piazza or Canvas
  - Python documentation: https://docs.python.org/3.6/
  - StackOverflow, Google
- The standard python 2 or 3.6 on Ubuntu works.

passthrough.py

- Neatly enapsulates the methods you may need to override (not all of them)
- Two types of methods
  1. filesystem methods which deal with directories and files
  2. file methods which deal with the contents of files

Logging

- There is some python magic which logs information to the screen as the file system works.

```
class LoggingMixIn:
    log = logging.getLogger('A2')

    def __call__(self, op, path, *args):
        self.log.debug('-> path:%s %s%s', path, op, repr(args))
        ret = '[Unhandled Exception]'
        try:
            ret = getattr(self, op)(path, *args)
            return ret
        except OSError as e:
            ret = str(e)
            raise
```

Part1

- Setup
  - Put `fuse.py, passthrough.py` and `a2fuse1.py` in the same directory
  - Create a directory called "`source`"
  - Download files "`one`", "`two`", and "`three`" from Canvas into the "source" directory.
  - Create a directory called "`mount`"
- Run "`python a2fuse1.py source mount`"
  - Observe logs and highlight the main functionality, for example

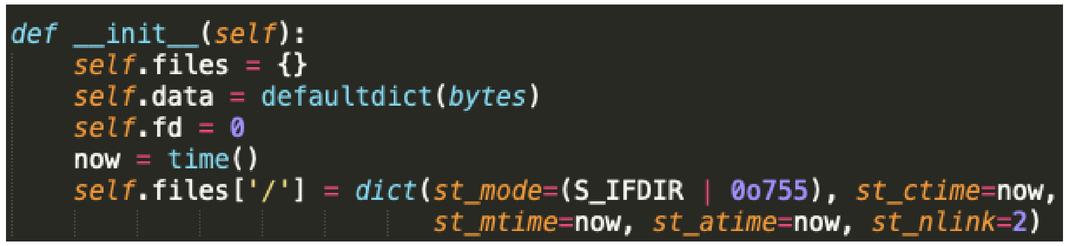
```
DEBUG:fuse.log-mixin:-> getattr /newfile (None,)
DEBUG:fuse.log-mixin:<- getattr {'st_atime': 1599037397.463398, 'st_ctime':
1599037379.543504, 'st_gid': 20, 'st_mode': 33188, 'st_mtime':
1599037379.543504, 'st_nlink': 1, 'st_size': 12, 'st_uid': 501}
```

Gets attributes for "newfile".

- To stop
  - "`fusermount -u mount`" (Ubuntu) or `umount mount`" (macOS)

Part2

- In the `Memory` class, explain what each method does
  - `__init__, getattr, readdir, open, create, unlink, write, read`



```
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)
```

- Example: `__init__`
  - Creates an empty dictionary self.files for the files, using 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.

Part3 – Requirement 1

- Setup
  - Create two directories, `source1` and `source2`
  - Put file `one` in `source1`
  - Put file `two` in `source2`
  - Make sure that `mount` is initially empty

- Enable your FUSE to mount two source directories into a mount point
  - Override the "`_full_path`" method to add logic to return multiple source directories
  - Override the "`readdir`" function to add logic s.t.
    - "`ls`" command in your `mount` can list the content of both `source1` and `source2`
  - Override the "`main`" method because we need an extra parameter
- Run "`python a2fuse2.py source1 source2 mount`"

Part3 – Requirement 2

- Create your own user space file system
- The file system has two classes of files 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.
  - Override the File methods to create a totally cached file system
  - When a file is opened read all of the contents into memory
  - Any reads are extracted directly from memory with no disk access
  - Any writes go to memory
  - When the file is closed the changes get written to disk

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