370 A2 Answer
elee353, 840454023

Question 1

Output:

```
st970703@ubuntu:~/Desktop/370_A2$ ls -l source
total 700
-rw-rw-r-- 1 st970703 st970703 700001 Sep 14 05:31 hundredthousand
-rw-rw-r-- 1 st970703 st970703     31 Sep 14 05:31 oneten
-rw-rw-r-- 1 st970703 st970703   3001 Sep 14 05:31 onethousand
-rw-rw-r-- 1 st970703 st970703   6001 Sep 14 05:31 twothousand
```

This lists all the files in the source directory. The file attributes and permissions are shown.
The total disk allocation for all files in this directory is 700 blocks.

```
st970703@ubuntu:~/Desktop/370_A2$ ls -l mount
total 0
-rw-rw-r-- 1 st970703 st970703 700001 Sep 14 05:31 hundredthousand
-rw-rw-r-- 1 st970703 st970703     31 Sep 14 05:31 oneten
-rw-rw-r-- 1 st970703 st970703   3001 Sep 14 05:31 onethousand
-rw-rw-r-- 1 st970703 st970703   6001 Sep 14 05:31 twothousand
```

This lists all the files in the mount directory. The file attributes and permissions are shown.
The total disk allocation for all files in this directory is 0 blocks.
This is because the actual mount directory on disk is empty. A virtual mount directory is created when a2fuse1.py is run. The source directory is mounted to this virtual directory.

The files in the source directory are mounted to the mount directory after the mount command is run.

There are two processes running. One process provides the file system functionalities, while the other handles user-level code. The a2fuse1.py run is a handler program linked to the libfuse library. It specifies actions required for the reading, writing, and mounting requests.

When the user uses the mount command, the handler is registered with the kernel and the command is passed through the GNU C library. This then makes a system call into the kernel. When the user issues read/write/stat requests for this newly mounted file system, the kernel forwards these IO-requests to the handler. The virtual file system and FUSE modules inside the kernel handle the requests and then sends the handler's response back to the user.

The file system here is a virtual file system. A virtual file system is an abstraction layer on top of a more concrete file system. It acts as a view or translation of an existing file system.

Question 2

cat oneten

```
DEBUG:fuse.log-mixin:-> getattr / (None,)
DEBUG:fuse.log-mixin:<- getattr {'st_ctime': 1506549172.211756, 'st_mtime':
1506549172.211756, 'st_nlink': 2, 'st_mode': 16893, 'st_size': 4096, 'st_gid': 1000, 'st_uid':
1000, 'st_atime': 1506549172.2197568}
DEBUG:fuse.log-mixin:-> access / (1,)
DEBUG:fuse.log-mixin:<- access None
DEBUG:fuse.log-mixin:-> getattr /oneten (None,)
DEBUG:fuse.log-mixin:<- getattr {'st_ctime': 1506549163.5910938, 'st_mtime':
1506502314.0, 'st_nlink': 1, 'st_mode': 33188, 'st_size': 31, 'st_gid': 1000, 'st_uid': 1000,
'st_atime': 1506502314.0}
DEBUG:fuse.log-mixin:-> open /oneten (32768,)
DEBUG:fuse.log-mixin:<- open 4
DEBUG:fuse.log-mixin:-> read /oneten (4096L, 0, 4L)
DEBUG:fuse.log-mixin:<- read 'oneoneoneoneoneoneoneoneoneone\n'
DEBUG:fuse.log-mixin:-> getattr /oneten (None,)
DEBUG:fuse.log-mixin:<- getattr {'st_ctime': 1506549163.5910938, 'st_mtime':
1506502314.0, 'st_nlink': 1, 'st_mode': 33188, 'st_size': 31, 'st_gid': 1000, 'st_uid': 1000,
'st_atime': 1506550568.0480227}
DEBUG:fuse.log-mixin:-> flush /oneten (4L,)
DEBUG:fuse.log-mixin:<- flush None
DEBUG:fuse.log-mixin:-> release /oneten (4L,)
DEBUG:fuse.log-mixin:<- release None
```

- getattr / (None,) - gets the file attributes associated with the mount directory.
  - The file attributes are returned.
- access / (1,) - checks the accessibility of the mount directory.
  - It is accessible, hence None is returned.
- getattr /oneten (None,) - gets the file attributes associated with the file oneten.
  - The file attributes are returned.
- open /oneten - opens the file oneten for reading.
  - 4 is the file descriptor value returned. It is an index value.
- read /oneten (4096L, 0, 4L) - reads the content of the opened file oneten.
  - The file content read is returned.
- getattr /oneten (None,) - gets the file attributes associated with the file oneten.
  - The file attributes are returned.
- flush /oneten (4L,) - flushes the cached data
  - The returned value None means success.
- release /oneten (4L,) - releases and closes the opened file, oneten.
  - The file is successfully released and closed.

Cat > newFile

```
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_ctime': 1506550625.420613, 'st_mtime':
1506550625.420613, 'st_nlink': 1, 'st_mode': 33204, 'st_size': 0, 'st_gid': 1000, 'st_uid':
1000, 'st_atime': 1506550625.420613}
DEBUG:fuse.log-mixin:-> flush /newfile (4L,)
DEBUG:fuse.log-mixin:<- flush None
```

- getattr /newfile (None,) - gets the file attributes associated with the file newfile.
  - newFile doesn’t exist yet. Hence an error, ‘No such file or directory’ is returned.
- create /newfile (33204L,) - creates and opens a file.
  - The file is created with the specified mode, and then it is opened.
  - The file descriptor,4 is returned.
- getattr /newfile (None,) - gets the file attributes associated with the file newfile.
  - The file attributes are returned.
- flush /newfile (4L,) - flushes the cached data.
  - Flush is successfully completed.

```
hello world
```

```
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 ('hello world\n', 0, 4L)
DEBUG:fuse.log-mixin:<- write 12
```

- getxattr /newFile (u'security.capability',) - retrieves the value of the extended attributes.
  - getxattr() hasn't been implemented yet.
  - This raises the error, '[Errno 95] Operation not supported'.
- write /newFile ('hello world\n', 0, 4L) - writes the string, ‘hello world’ to the opened file.
  - Write returns 12 bytes as requested.

```
Ctrl + D
```

```
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
```

- flush /newFile (4L,) - flushes the cached data
  - Flush is successfully completed.
- release /newFile (4L,) - releases and closes the opened file, newFile.
  - The file is successfully released and closed.

```
ls
```

```
DEBUG:fuse.log-mixin:-> opendir / ()
DEBUG:fuse.log-mixin:<- opendir 0
```

```
DEBUG:fuse.log-mixin:-> getattr / (None,)
DEBUG:fuse.log-mixin:<- getattr {'st_ctime': 1506550884.3201988, 'st_mtime':
1506550884.3201988, 'st_nlink': 2, 'st_mode': 16893, 'st_size': 4096, 'st_gid': 1000,
'st_uid': 1000, 'st_atime': 1506551053.1984487}
DEBUG:fuse.log-mixin:-> readdir / (0L,)
DEBUG:fuse.log-mixin:<- readdir <generator object readdir at 0x7f8a17ef0870>
DEBUG:fuse.log-mixin:-> getattr /twothousand (None,)
DEBUG:fuse.log-mixin:<- getattr {'st_ctime': 1506549163.5910938, 'st_mtime':
1506502314.0, 'st_nlink': 1, 'st_mode': 33188, 'st_size': 6001, 'st_gid': 1000, 'st_uid': 1000,
'st_atime': 1506502314.0}
DEBUG:fuse.log-mixin:-> getattr /newFile (None,)
DEBUG:fuse.log-mixin:<- getattr {'st_ctime': 1506550982.6352646, 'st_mtime':
1506550982.6352646, 'st_nlink': 1, 'st_mode': 33204, 'st_size': 12, 'st_gid': 1000, 'st_uid':
1000, 'st_atime': 1506550819.1166415}
DEBUG:fuse.log-mixin:-> getattr /oneten (None,)
DEBUG:fuse.log-mixin:<- getattr {'st_ctime': 1506549163.5910938, 'st_mtime':
1506502314.0, 'st_nlink': 1, 'st_mode': 33188, 'st_size': 31, 'st_gid': 1000, 'st_uid': 1000,
'st_atime': 1506550568.0480227}
DEBUG:fuse.log-mixin:-> getattr /hundredthousand (None,)
DEBUG:fuse.log-mixin:<- getattr {'st_ctime': 1506549163.5870934, 'st_mtime':
1506502314.0, 'st_nlink': 1, 'st_mode': 33188, 'st_size': 700001, 'st_gid': 1000, 'st_uid':
1000, 'st_atime': 1506502314.0}
DEBUG:fuse.log-mixin:-> getattr /onethousand (None,)
DEBUG:fuse.log-mixin:<- getattr {'st_ctime': 1506549163.5910938, 'st_mtime':
1506502314.0, 'st_nlink': 1, 'st_mode': 33188, 'st_size': 3001, 'st_gid': 1000, 'st_uid': 1000,
'st_atime': 1506502314.0}
DEBUG:fuse.log-mixin:-> releasedir / (0L,)
DEBUG:fuse.log-mixin:<- releasedir 0
```

- opendir / () - opens the directory.
    - opens a directory stream whose elements are directory entries.
    - A pointer to the directory stream is returned.
    - The stream is positioned at ‘/’, which is the mount directory.
    - The returned value 0 means success.
- getattr / (None,) - gets the file attributes associated with the directory.
    - The file attributes are returned.
- readdir / (0L,) - reads the directory.
    - The pointer to an object of type DIR is returned, which is indicated by <generator object readdir at 0x7f8a17ef0870>. This is the memory address of the mount directory.
    - The DIR object pointed represents the next directory entry in the directory stream.
- getattr /twothousand (None,) - gets the file attributes associated with the file twothousand.
    - The file attributes are returned.
- This getattr process is repeated for every file in the directory.
- releasedir / (0L,) - releases and closes the mounted directory.
    - The returned value 0 means success.

Rm newFile

```
DEBUG:fuse.log-mixin:-> getattr /newFile (None,)
DEBUG:fuse.log-mixin:<- getattr {'st_ctime': 1506551269.4314816, 'st_mtime':
1506551269.4314816, 'st_nlink': 1, 'st_mode': 33204, 'st_size': 6, 'st_gid': 1000, 'st_uid':
1000, 'st_atime': 1506551265.8235352}
DEBUG:fuse.log-mixin:-> access /newFile (2,)
DEBUG:fuse.log-mixin:<- access None
DEBUG:fuse.log-mixin:-> unlink /newFile ()
DEBUG:fuse.log-mixin:<- unlink None
```

- getattr /newFile (None,) - gets the file attributes associated with the newfile file.
  - The file attributes are returned.
- access /newFile (2,) - Check the file access permissions.
  - It is accessible, hence None is returned.
- unlink /newFile () - removes the file.
  - None is returned on success.

Part 2

Question 3

For the following list of methods in the Memory class explain exactly what each method does.
Include a statement by statement explanation.

__init__

This method is a constructor for instantiating a Memory object.

1. self.files = {}
   a. The files field is an empty dictionary.
2. self.data = defaultdict(bytes)
   a. The defaultdict initialises a dictionary with they type bytes.
   b. The new dictionary is called ‘data’.
3. self.fd = 0
   a. The file descriptor field is initialised with 0.
4. now = time()
   a. The now field has a value of the current time in seconds since the Epoch. This variable is an instance level variable and has a floating point value.
5. self.files['/'] = dict(st_mode=(S_IFDIR | 0o755), st_ctime=now, st_mtime=now, st_atime=now, st_nlink=2)
   a. There is another dictionary stored inside the files dictionary.
   b. St_mode is set to S_IFDIR.
   c. The sequences of the inner dictionary’s key-value pairs include st_mode, st_ctime, st_mtime, etc.
   d. S_IFDIR is a directory file type. Its value is OR-ed using the value 0o755.
   e. St_ctime: the creation time is the current time.
   f. St_mtime: the modified time is the current time.

g. St_atime: the last accessed time is the current time.
h. St_nlink: the number of hard links linked to the file is two.
i. Stores the inner dictionary inside the outer dictionary using root directory, ‘/’ as the key.

getattr

This method gets the file attributes associated with the input path.

1. if path not in self.files:
    a. Checks if the path is stored in the files dictionary.
2. raise FuseOSError(ENOENT)
    a. Raises a Fuse error. This means no such file or directory.
3. return self.files[path]
    a. Otherwise, returns the file found in the files dictionary field.

readdir

This method reads the directory specified.

1. return ['.', '..'] + [x[1:] for x in self.files if x != '/']
    a. Creates a list containing '.' and '..'.
    b. Extracts results from the files field.
    c. Loops through them, starting from index 1, using the results that are not ‘/’, the root directory.
    d. Prepends the dots to the results.

open

This method opens the file specified.

1. self.fd += 1
    a. Increments file descriptor field value.
2. return self.fd
    a. Return the incremented value of the file descriptor field.

create

This method creates a new file at the location specified by path.

1. self.files[path] = dict(st_mode=(S_IFREG | mode), st_nlink=1, st_size=0, st_ctime=time(), st_mtime=time(), st_atime=time())
    a. Creates a new dictionary and stores it into the files dictionary with path as the key.
    b. S_IFREG means the file type is regular. Its value is OR-ed with the mode argument given. This value is used for the ‘file type and mode’ attribute.
    c. The number of hard links is one.

    d. St_size is the total size in bytes. It is set to zero.
    e. st_ctime, time of last status change is set to the current time.
    f. St_mtime, time of last modification is set to the current time.
    g. St_atime, time of last access is set to the current time.
    h. These times are set to the current time because this is a new file.
    i. Stores the path dictionary into the files dictionary
2. self.fd += 1
    a. Increments file descriptor field value.
3. return self.fd
    a. Returns the incremented file descriptor value.

unlink

This method removes the file specified.

1. self.files.pop(path)
    a. The path is used as the key to remove the corresponding stored file from the files dictionary.

write

This method writes data to a file specified.

1. self.data[path] = self.data[path][:offset] + data
    a. Extracts data from the data field with the path variable as key.
    b. Selects only the range from index 0 to offset.
    c. Appends the data variable to the end of the selected range.
    d. Stores the results back.
2. self.files[path]['st_size'] = len(self.data[path])
    a. st_size is size in bytes of a plain file.
    b. Extracts result from the path field with the path variable as key.
    c. Extracts result from the data field with the path variable as key.
    d. Edits the st_size property with the new data size.
3. return len(data)
    a. Returns the number of items of the data variable.

read

This method reads the content of the file specified.

1. return self.data[path][offset:offset + size]
    a. Extracts data from the data field with the path variable as key.
    b. Only returns the range from offset to offset + size.