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 |
| 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 |

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

There are two processes running. One to provide the file system functionalities, the other to handle 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 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 cached data   + The returned value None means success. * release /oneten (4L,) - releases the opened directory.   + The returned value None means success. |

|  |
| --- |
| 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. So 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.   + A file descriptor is returned. * getattr /newfile (None,) - gets the file attributes associated with the file newfile.   + The file attributes are returned. * flush /newfile (4L,) - flushes 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 attribute.   + getxattr() hasn't been implemented yet.   + This raises the error, '[Errno 95] Operation not supported'. * write /newFile ('hello world\n', 0, 4L) - writes data 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 cached data   + Flush is successfully completed. * release /newFile (4L,) - releases the opened file.   + The file is successfully released. |

|  |
| --- |
| 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.   + 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.   + A pointer to an object of type DIR is returned. * 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 the opened 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 = {}    1. The files field is an empty dictionary. 2. self.data = defaultdict(bytes)    1. The defaultdict initialises a dictionary with they type bytes. 3. self.fd = 0    1. The fd field is initialised with 0. 4. now = time()    1. The now field has a value of the current time in seconds since the Epoch.   The value is a floating point value.  This variable is an instance level variable.   1. self.files['/'] = dict(st\_mode=(S\_IFDIR | 0o755), st\_ctime=now, st\_mtime=now, st\_atime=now, st\_nlink=2)    1. There is another dictionary stored inside the files dictionary.    2. St\_mode is set to S\_IFDIR.    3. The sequences of the inner dictionary’s key-value pairs include st\_mode, st\_ctime, st\_mtime, etc.    4. S\_IFDIR is a directory file type. Its value is OR-ed using the value 0o755.    5. St\_ctime: the creation time is current time.    6. St\_mtime: the modified time is current time.    7. St\_atime: the last accessed time is current time.    8. St\_nlink: the number of hard links linked to the file is two.    9. Stores the inner dictionary inside the outer dictionary using ‘/’ as the key. |

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

|  |
| --- |
| readdir |
| 1. return ['.', '..'] + [x[1:] for x in self.files if x != '/']    1. Extracts results from the files field.    2. Loops through them, starting from index 1, using the results that are not ‘/’, the root directory.    3. Prepends the dots to the results. ‘.’ or ‘..’ is prepended depending the method call. |

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

|  |
| --- |
| create |
| 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())    1. Creates a new dictionary and stores it into the files dictionary with path as the key.    2. 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.    3. The number of hard links is one.    4. St\_size is the total size in bytes. It is set to zero.    5. st\_ctime, time of last status change is set to current time.    6. St\_mtime, time of last modification is set to current time.    7. St\_atime, time of last access is set to current time.    8. Stores the path dictionary into the files dictionary 2. self.fd += 1    1. Increments file descriptor field value. 3. return self.fd    1. Returns the incremented file descriptor value. |

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

|  |
| --- |
| write |
| 1. self.data[path] = self.data[path][:offset] + data    1. Extracts data from the data field with the path variable as key.    2. Selects only the range from index 0 to offset.    3. Appends the data variable to the end of the selected range.    4. Stores the results back. 2. self.files[path]['st\_size'] = len(self.data[path])    1. st\_size is size in bytes of a plain file.    2. Extracts data from the path field with the path variable as key.    3. Extracts data from the data field with the path variable as key.    4. Edits the st\_size property with the new data size. 3. return len(data)    1. Returns the number of items of the data variable. |

|  |
| --- |
| read |
| 1. return self.data[path][offset:offset + size]    1. Extracts data from the data field with the path variable as key.    2. Only returns the range from offset to offset + size. |