SE370 Assignment 2 - Answers

Full Name	Ernest Pui Hong Wong
UPI	ewon746
ID	178687569

Question 1

Running the commands yields:

```
ernwong@ErnWong-Latitude-E6230:/tmp/se370-scratch
$ mkdir a2tasks
ernwong@ErnWong-Latitude-E6230:/tmp/se370-scratch
$ ls -al a2tasks
total 8
drwxr-xr-x 2 ernwong ernwong 4096 Oct  3 22:38 .
drwxr-xr-x 3 ernwong ernwong 4096 Oct  3 22:38 ..
ernwong@ErnWong-Latitude-E6230:/tmp/se370-scratch
$ echo "something" > a2tasks/some.txt
ernwong@ErnWong-Latitude-E6230:/tmp/se370-scratch
$ ls -al a2tasks
total 12
drwxr-xr-x 2 ernwong ernwong 4096 Oct  3 22:38 .
drwxr-xr-x 3 ernwong ernwong 4096 Oct  3 22:38 ..
-rw-r--r-- 1 ernwong ernwong   10 Oct  3 22:38 some.txt
ernwong@ErnWong-Latitude-E6230:/tmp/se370-scratch
$ mkdir a2tasks/d1
ernwong@ErnWong-Latitude-E6230:/tmp/se370-scratch
$ ls -al a2tasks
total 16
drwxr-xr-x 3 ernwong ernwong 4096 Oct  3 22:38 .
drwxr-xr-x 3 ernwong ernwong 4096 Oct  3 22:38 ..
drwxr-xr-x 2 ernwong ernwong 4096 Oct  3 22:38 d1
-rw-r--r-- 1 ernwong ernwong   10 Oct  3 22:38 some.txt
ernwong@ErnWong-Latitude-E6230:/tmp/se370-scratch
$ mkdir a2tasks/d2
ernwong@ErnWong-Latitude-E6230:/tmp/se370-scratch
$ ls -al a2tasks
total 20
drwxr-xr-x 4 ernwong ernwong 4096 Oct  3 22:39 .
drwxr-xr-x 3 ernwong ernwong 4096 Oct  3 22:38 ..
drwxr-xr-x 2 ernwong ernwong 4096 Oct  3 22:38 d1
drwxr-xr-x 2 ernwong ernwong 4096 Oct  3 22:39 d2
-rw-r--r-- 1 ernwong ernwong   10 Oct  3 22:38 some.txt
ernwong@ErnWong-Latitude-E6230:/tmp/se370-scratch
$
```

The number of hard links for any particular file represents the number of entries across all directories that point to that particular file. For directories, this includes the `.` and `..` entries. Since all directories must have a `.` entry that points to itself, and all directories (apart from the root directory) are stored in exactly one parent

directory, it means that all directories first start off with 2 hard links upon a fresh creation as we can see for `a2tasks` at the beginning, and `d1` and `d2`.

As every directory has a `..` entry to its parent, adding each new child directory would increment the number of hard links in the parent directory. This is why we have 4 hard links by the time we added `d1` and `d2`. Note that adding regular files such as `some.txt` does not increment the number of hard links of the parent directory since it does not add any extra `..` links to the filesystem.

In general, since directories may only have one parent, the general formula for the number of hard links for a directory is `= 2 + (Number of immediate child directories)`.

Question 2

Hard links are used to implement filesystem navigation via `.` and `...` However, this is done behind the scenes by the filesystem and is the only way to link directories in order to protect the tree-like properties of the directories themselves. This cannot create a problem of cycles since the implementation of the filesystem syscalls prevent arbitrary userspace programs to create such hard links to directories. As long as the filesystem implementation itself always links directories to a single parent, and disallows directories to be moved into a subdirectory itself, then cycles can be prevented.

Question 3

When change-directory is requested to a new directory, the `getattr` is called for every directory that is needed to traverse from the root of the filesystem to the desired directory. This is to check that:

- The directories exist,
- The destination is indeed a directory (by checking the mode bits),
- Every intermediate directory has execute permissions (which represent the permission to enter the directory and access the files inside) allowing the use to access the subdirectories inside it.

Question 4

A lock for the inode needs to be taken when modifying the file's size and when performing file-write operations. Without locking of the file's size, write-appending can overwrite concurrently added data because it would have been trying to append using the old file size before another process has resized the file. The process trying to append to the file would have first read the file's size into memory, after which a second process changes the file size, finally after which the original process writes data into somewhere that is not the file's end position.

File permission information also needs to be locked to prevent time-of-check-to-time-of-use problems, where the permissions settings has elevated after the permissions has checked, but before the operation requiring the permissions is performed. Without locking, it allows operations to be performed even when they are not permitted to.

In general, inode stats information needs to be locked when accessed. For modifications, an exclusive lock needs to be used. For purely reads, a shared lock can be used so that it cannot read partially written stats that are not consistent.

Question 5

Locating file contents are easier. In this C implementation, the file contents are located by a pointer to a contiguous region of memory from the point of view of the program. This makes it easy to arbitrarily locate bytes in the file given an offset, as the program can simply perform pointer arithmetic to find the byte's address.

However, many block-based devices map the file to multiple non-contiguous blocks on the disk with several layers of indirection to improve their performances. These levels of indirections increases the complexity of resolving where a given byte of a file at an arbitrary offset is actually located on the disk.

For instance, the inode may directly point to specific data blocks out of order, or it may point to index blocks that further point to more data blocks.

If a requested region of data spans across multiple blocks, the filesystem needs to reconstruct the data from those blocks back into a single buffer.

Question 6

It is impossible to create files with slashes in their names since the FUSE API that we bind our filesystem operations to all uses the absolute path to the file to infer its filename. While the internal directory-entry data structures that this filesystem uses may allow slashes in its name string, any attempt by the user to add a slash into the filename will be treated as having another subdirectory before the slash.

Filenames longer than 255 characters are also impossible since the filenames are stored in directory-entry structs that are of limited size. Moreover, explicit length-checking is done while copying the filename strings. If the filename is too long, a `ENAMETOOLONG` error is returned.

Question 7

When tab-completion is being executed, the parent directory is first opened for reading, followed by a `getattr` check for its size, followed by `readdir` to read the directory-entries to compile a list of match candidates in that directory. Once reading is done, the directory is closed via `releasedir`. After that, `getattr` is called for every matching file, to gather context about what the files are. For instance, files without the execute mode bit cannot be run, so it will not be tab-completed when the user is typing a program to run.