Homework Assignment 2

William Jagels

November 4, 2016

I. Q&A

I.1.

1. x.dat is an empty file because it was created with the `O_CREAT` flag
2. x.dat is still empty because it was just opened
3. x.dat now contains `AAAAAA` because it was written to from the beginning
4. x.dat now contains `BBBAAA` because `fd2` still writes from the beginning which overwrites the previous bytes
5. x.dat does not change, but `fd2` will be closed and become a copy of `fd1`
6. x.dat now contains `BBBAAABBB` because `fd2` is a duplicate of `fd1`, with the same file offset, so it will write to the end.
7. x.dat will not change, but it will now no longer be marked in the open table.

I.2.

`stdout` is normally line-buffered, so every time a newline character is encountered, the buffer is flushed and output is visible in the terminal. Normal redirection is fully-buffered, so after a certain number of bytes, the buffer is flushed, at which point it becomes visible on the terminal. The first instance of `t` has fully buffered output thanks to the `|` operator, so the second instance of `t` gets everything in batches, which it then echoes into a line-buffered output.

I.3.

1. `dir_name` is not large enough for the null terminator.
2. `stat()` gets symlink targets, not the link itself. Should use `lstat()`.
3. The call to `sprintf()` is wrong, `dir_name` needs to be before the other arguments.
4. `statP` is an uninitialized pointer, this will crash the call to `stat()`. Instead of a pointer, statP should be declared as an automatic variable, and its address should be passed to `stat()`. This also requires updating the call to `S_ISLNK()`.
5. `dir` is redeclared, either rename the string or the `DIR *`.
6. `readdir()` returns a `struct dirent *`, so `dP` must be declared as `struct dirent *dP`.

I.4.

	data1 R	data1 W	data2 R	data2 W
u1 runs exec1	Y	N	Y	Y
u2 runs exec1	Y	Y	N	Y
u1 runs exec2	Y	N	Y	Y
u2 runs exec2	Y	Y	Y	Y

The first column is easy, `data1` will always be readable because it has the read bit on for owner, group, and everyone. In the second column, `u1` is unable to write as a consequence of having auxiliary group `g2`. For the third column, `u2` will not be able to read unless they run `exec2` which gives them an effective uid of `u1`. And for the fourth column, `data2` is only non-writable when the gid is `g1` and the uid is not `u1`. This state is impossible, so all can write.

I.5.

1. `cd` must be a built-in command because since shells fork before executing a program, changes to the working directory will be lost as soon as a theoretical `cd` binary finishes.

2. `pwd` can be an external binary because it just has to read the working directory and print it, and the working directory is inherited from the shell.

3. `exec` must be a built-in command because a binary would not be able to replace the shell since it would only replace the forked instance.

4. `exit` must be a built-in command because the shell forks first, so any attempts by an external process to exit or crash the shell will have no effect. However, if the PID of the shell could be determined, the binary could send a termination signal to the shell. This would still require some built-in shell support for passing along the PID of the main shell process.

I.6.

1. This is valid unless the writing failed, because the offset is incremented when bytes are successfully written to files. Non-empty data can be written, but fail, in which case the statement is invalid. There's also the case where you use `pwrite()` which will write and then restore the file offset atomically.

2. As long as the user has ownership over that file, they can delete it even if the permissions are set to none. However, if the parent directory changes to non-executable for that user, the file will be non-accessible and consequentially, non removable. Since the question only restricted changes to the file itself, this statement is invalid in the case when the parent directories are manipulated.

3. The user id part of this statement is valid, but the group part is not. In the event that set-group-ID is active on the parent directory, then the gid is set as the gid of the parent directory. When the bit is not set, the gid is set to the effective gid of the calling process. (Taken from `open(2)`).

4. This statement is invalid. Moving a file should only require updating directory listings to point to the inode that points to the file. A bad programmer could theoretically copy and remove, but using `rename()` is easier and more efficient.