Project Five Report
Introduction to Operating Systems
New Beginnings Spring 2018

Gavin Megson

1 June 2018

Description

For this assignment, I implemented file permissions for owners, groups, and other categories. This involved changing the `inode` structures to include permission flags in the metadata, as well as modifying the `exec` function to check for permission. The `ls` command was also updated to list the permissions for files. Finally, new user commands and associated system calls were added to modify permissions of files.

Deliverables

The following features were added to xv6:

- All files were given metadata fields to track the UID of the file's owner, the group id of the file, and the permissions (mode) in the `inode`, `dinode`, and `stat` structures. This involved creating a new union stuct, `mode-t`, which tracks flags for read, write, and execute permissions, to be included in the `dinode` structure, and a duplicate stucture `stat-mode-t` to avoid a dependency problem in the stat command. The functions `ilock`, `stati`, `iupdate`, and create were updated to handle the metadata properly.
- New user commands `chmod`, `chown`, and `chgrp` change the mode (permissions), owner, and group of a file, respectively. These commands call similarly named system calls, which check the ranges of the inputs before executing.
- The `ls` command has been updated to show the permissions, in the form of user, group, and other `'rwx'` (read write execute) characters. The first character is either `-` or `D` for a file or directory, respectively. The next 3 are the `rwx` permissions for the user, then group, then others. A `-` indicates no permission. Additionally, the `inode` number is now displayed.
- To enforce permissions, the `exec` system call now check the process UID and GID against the file permissions. If a file's `setuid` flag is set, then the process will change its UID to the file's, if appropriate.

Implementation

`inode`, `dinode`, and `stat` Structures

- The `inode` structure in `fs.h` was updated to include the `mode-t` union (lines 31-46) as the `mode` attribute, allowing bit-level or integer-level manipulation. It now also includes fields for storing the uid and gid (lines 56-58). Similarly, the `stat` stucrture in `stat.h` now includes the `stat-mode-t` union (lines 6-22) and the uid and gid fields (lines 32-24). The number of data blocks directly linked to by an `inode` in `fs.h` was adjusted to account for the increased size of the metadata leading to decreased space (lines 22-26).

  Function `ialloc` in `mkfs.c` now updates the inode's `UID`, `GID`, and `mode` to the defaults (defined in `param.h` line 26) (lines 234-236) when allocating.

  In addition, the following functions in `fs.c` were modified:

  - Function `iupdate` now copies the `UID`, `GID`, and `mode` of an `inode` to a `dinode` (lines 213-215).
  - Function `ilock` now copies the `UID`, `GID`, and `mode` of a `dinode` to an `inode`, in the event it has to read the `inode` from disk (lines 295-297).
  - Function `stati` now copies the new metadata from an `inode` as well (lines 441-443).

New User Commands and Associated System Calls

New user commands `chmod`, `chown`, and `chgrp`, with associated system calls, were added via the usual process: `user.h` lines 47-49 (library function `atoo`'s prototype was also added to `user.h` at line 66), `usys.S` lines 36-38, `syscall.h` lines 33-35, and the files `chmod.c`, `chown.c`, and `chgrp.c` were created.

- The new system call `chmod` was created to implement the associated user command in `fs.c` (lines 727-755), with associated system call handler in `sysfile.c` (lines 567-575). After retrieving the arguments, it checks to make sure the mode argument is in the correct range of octal values. If so, it updates the given file's inode's `mode` field to the correct mode.
- The new system call `chown` was created to implement the associated user command in `fs.c` (lines 669-696), with associated system call handler in `sysfile.c` (lines 547-555). After retrieving the arguments, it checks to make sure the given user id is in a valid range. If so, it updates the given file's `UID`.
- The new system call `chgrp` was created to implement the associated user command in `fs.c` (lines 698-725), with associated system call handler in `sysfile.c` (lines 557-565). After retrieving the arguments, it checks to make sure the given group id is in a valid range. If so, it updates the given file's `GID`.

Updates to `ls` Command

- To facilitate the additional functionality of `ls` printing the permissions of files, the function `print-mode`, implemented in included file `print-mode.c`, is used to print the mode of an `inode` as characters representing the user/group/other process' ability to r(ead)/w(rite)/x(execute) the file. `ls.c` was modified at lines 6, 54, and 79 to reflect this special formatting.
- Additional changes to `ls.c` include printing a new header (line 49) and printing the inode number (lines 55 and 80).

`exec` System Call

The following changes were made to `exec.c`:Files `fs.h` and `file.h` were included (lines 10-11) to facilitate checking permissions of the file against the process calling `exec`. These permissions are checked in the pre-specified order (lines 45-57): user permissions if the UID matches, then group permissions if the GID matches, then other permissions. If the `inode` has its `setuid` bit set, then the calling process will have its UID changed to the file's (lines 24, 60-61, 122-123).

Testing

`p5-test` Suite A

The `Proc UID` test sets a file's `UID` and checks for an error return code. It then checks the new value to make sure it did actually change. Then it tries to set the UID to an invalid value, and makes sure it returns an error code. If all these tests pass, the program simply prints test passed.

This test `PASSES`.

`p5-test` Suite B

The `Proc GID` test sets a file's GID and checks for an error return code. It then checks the new value to make sure it did actually change. Then it tries to set the GID to an invalid value, and makes sure it returns an error code. If all these tests pass, the program simply prints test passed.

This test `PASSES`.

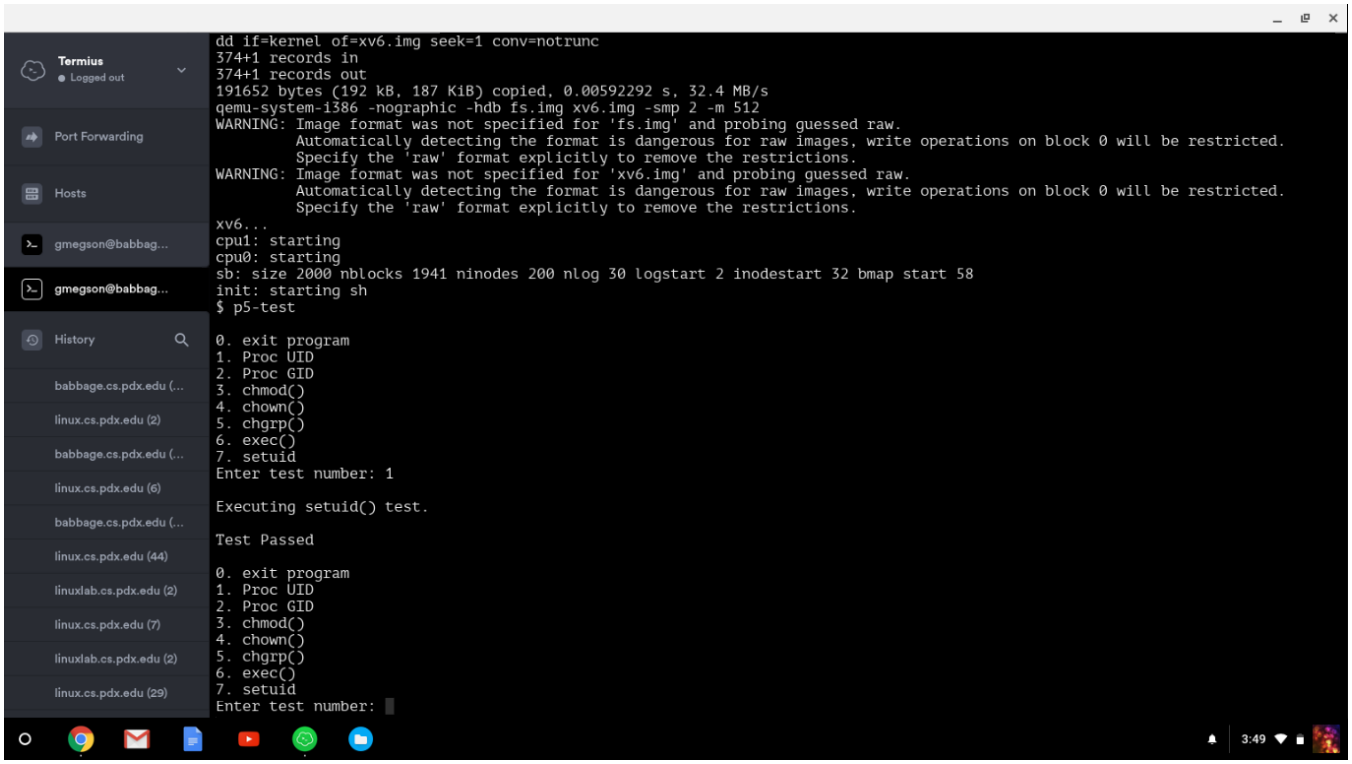

Figure 1: p5-test suite a

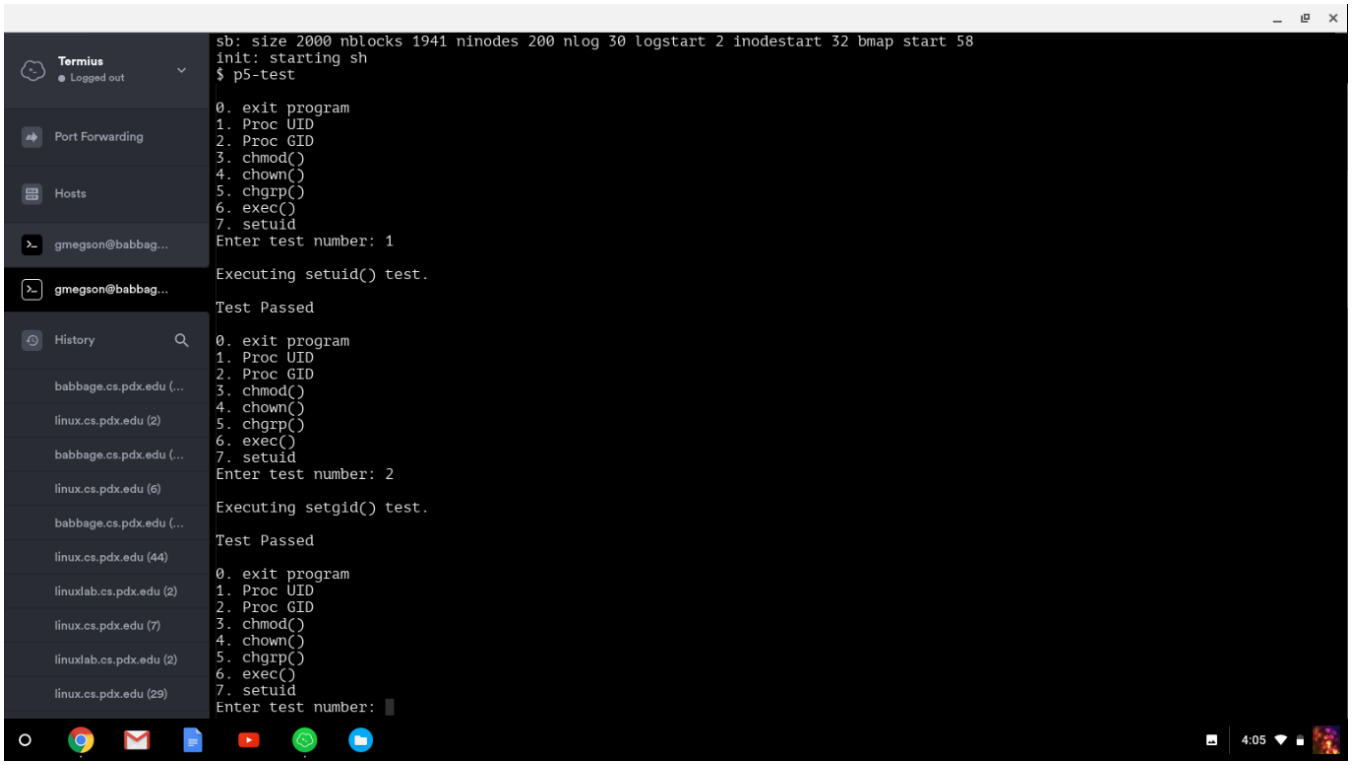

Figure 2: p5-test suite b

`p5-test` Suite C

The `chmod` test runs through a list of values and tries to set a file's permission to each of these. If the mode does not change, the test fails, otherwise it passes.

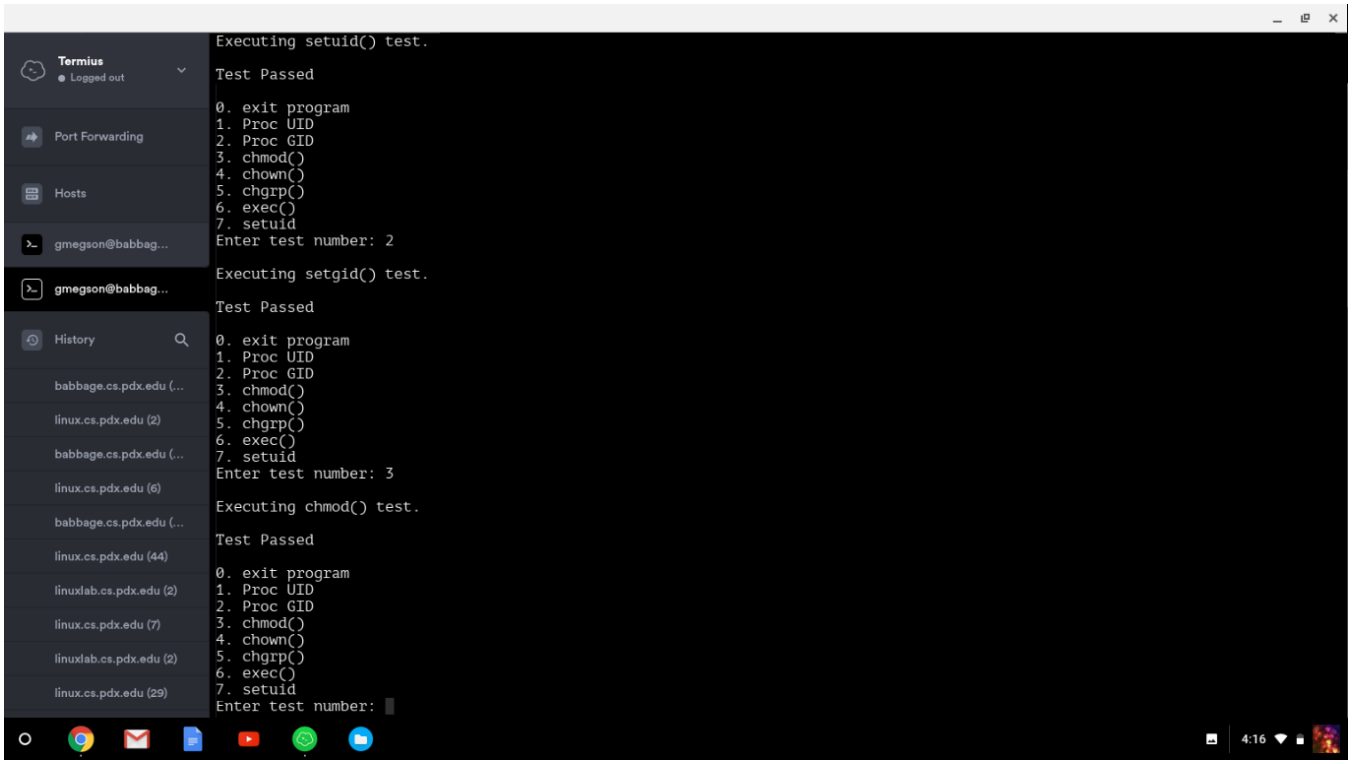

Figure 3: p5-test suite c

This test `PASSES`.

`p5-test` Suite D

The `chown` test runs increments through UID values and tries to set a file's UID to each of these. If `chown` returns an error code, or if the UID did not change, the test fails, otherwise it passes.

This test `PASSES`.

`p5-test` Suite E

The `chgrp` test changes a file's GID value one time. If `chgrp` returns an error code, or if the GID did not change, the test fails, otherwise it passes.

This test `PASSES`.

`p5-test` Suite F

The `exec` test runs through a list of different permissions, and then forks children to call `exec` with those permissions. If the `fork` command fails, if a child with the wrong permissions runs `exec` successfully, or if a child with the right permissions runs `exec` unsuccessfully, the test fails, otherwise it passes.

This test `PASSES`.

`p5-test` Suite G

The `Set UID` test runs through a list of 4 different permissions: where the UID matches, where the GID matches, where neither match, and where neither match and the mode is invalid, and then forks children to call `exec` with those permissions. If the `fork` command fails, if a child with the wrong permissions runs

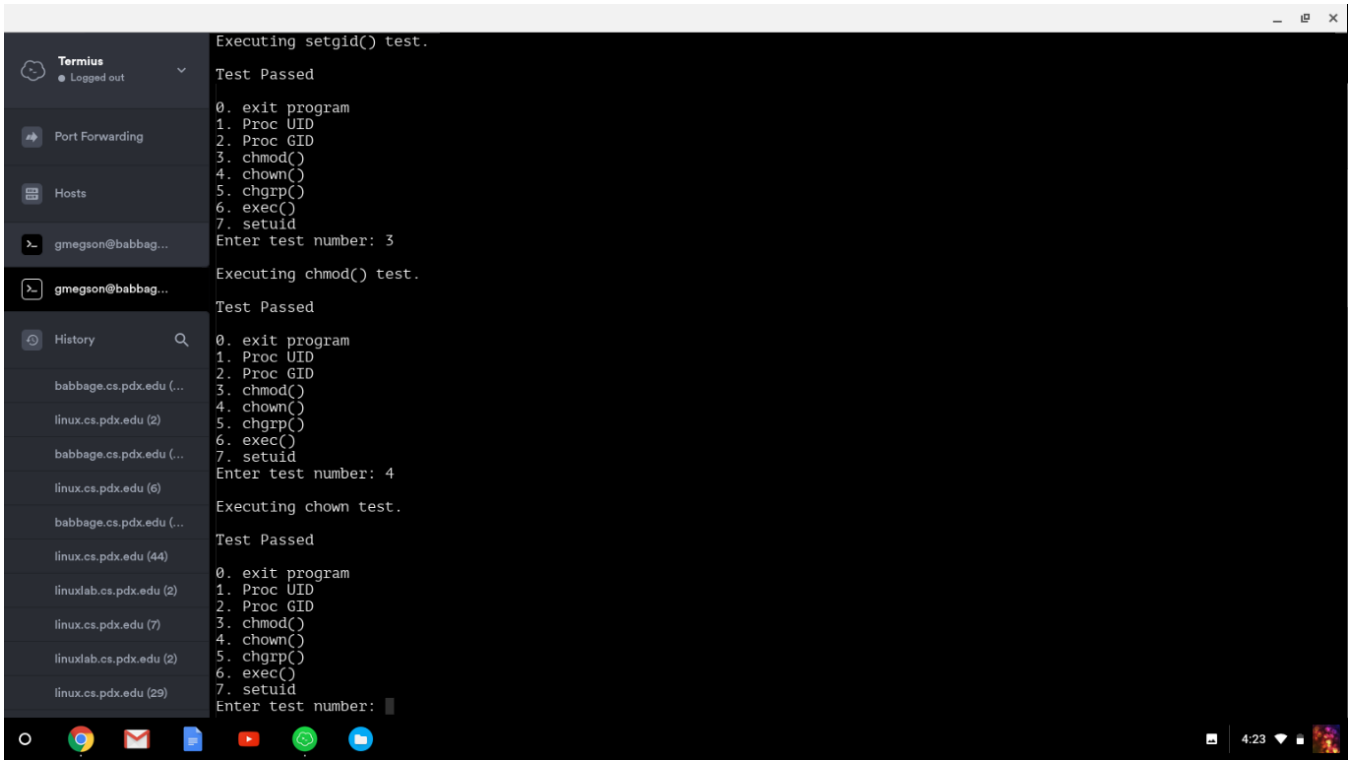

Figure 4: p5-test suite d

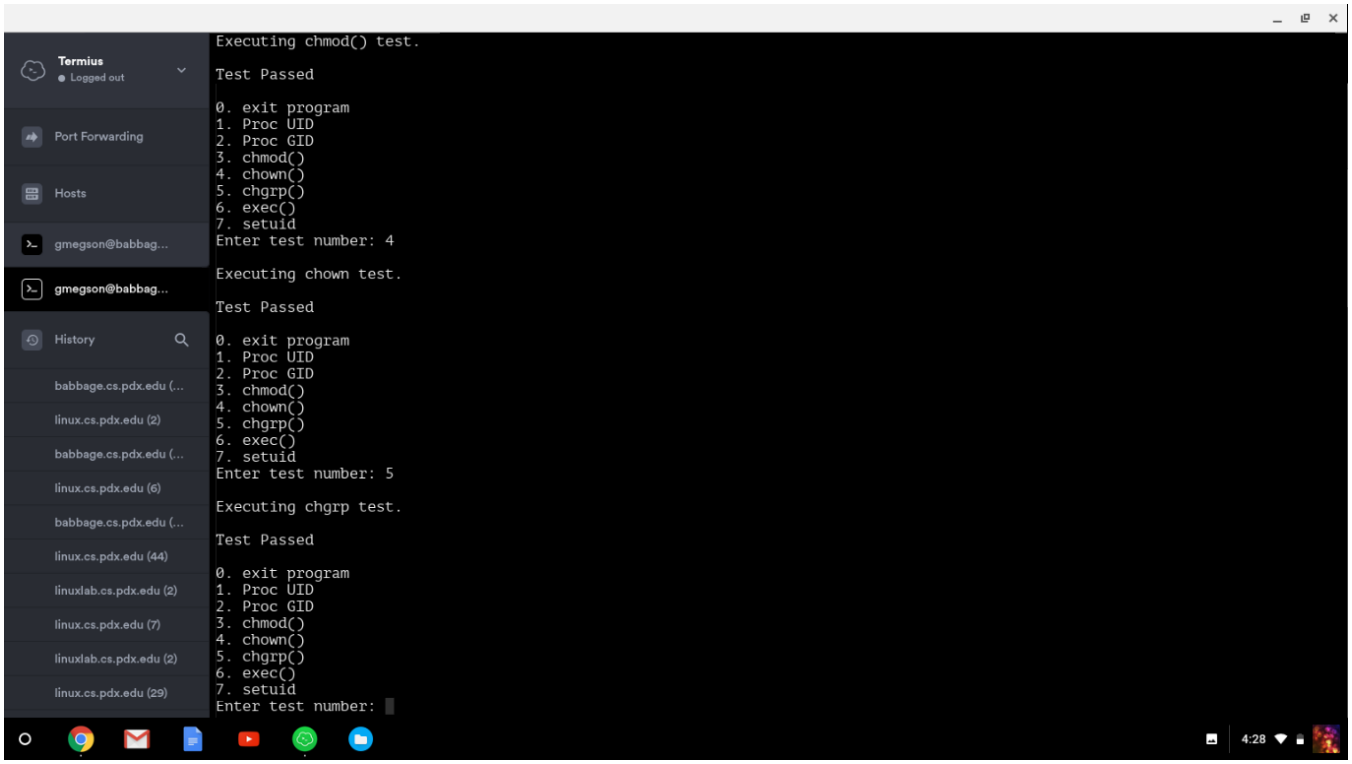

Figure 5: p5-test suite e

`exec` successfully, or if a child with the right permissions runs `exec` unsuccessfully, the test fails, otherwise it passes.

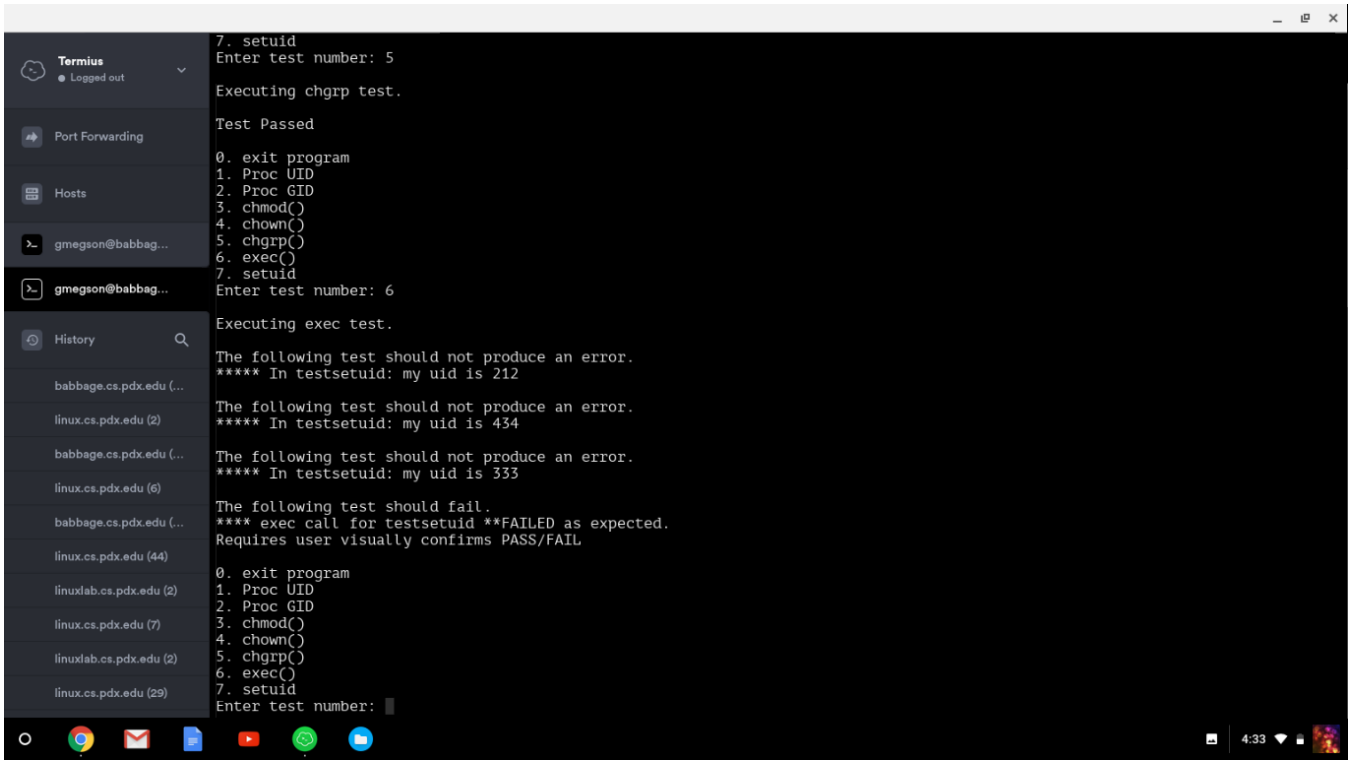

Figure 6: p5-test suite f

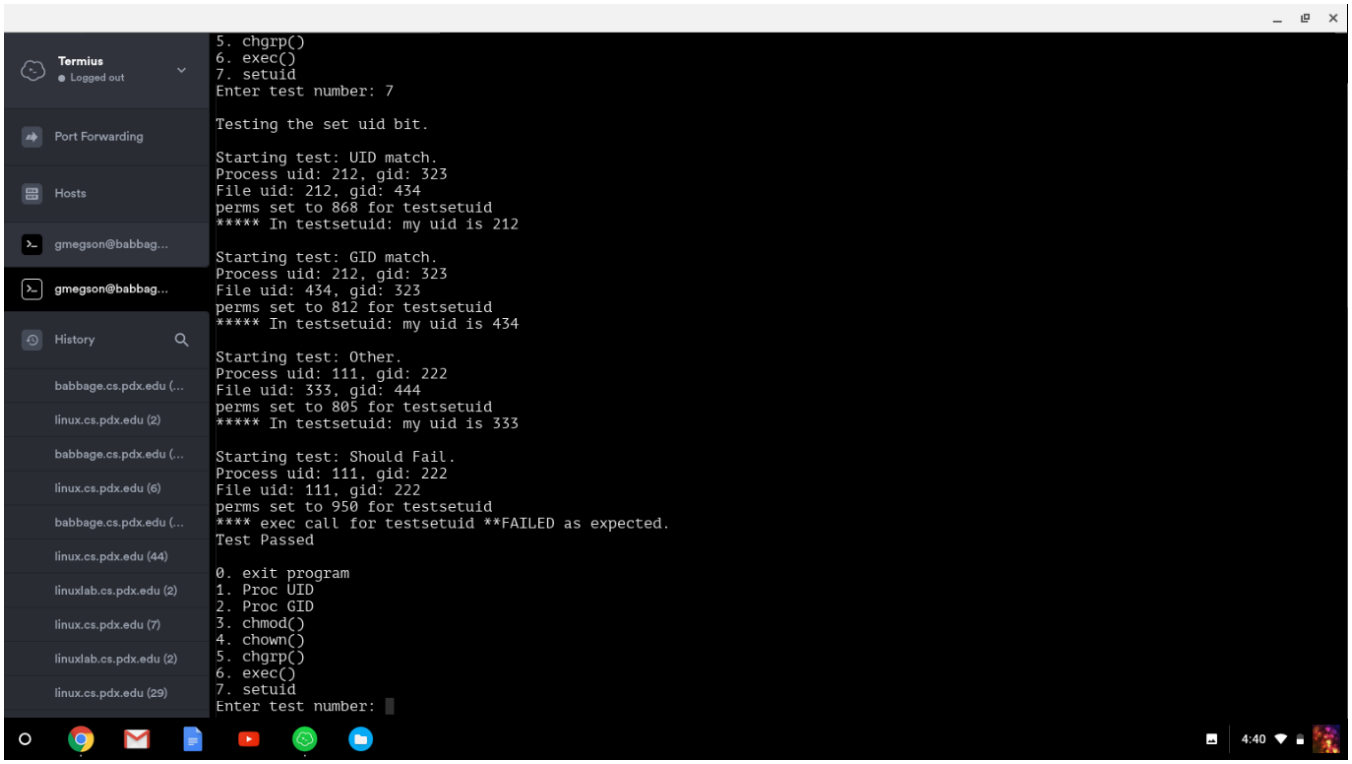

Figure 7: p5-test suite g

This test `PASSES`.

`chmod`, `chown`, `chgrp`, `ls`

In these tests, I will use `chmod`, `chown`, and `chgrp` on the `README` file, and use `ls` to show they behave as expected. This should satisfy tests 2, 3, 4, 5, and 6.

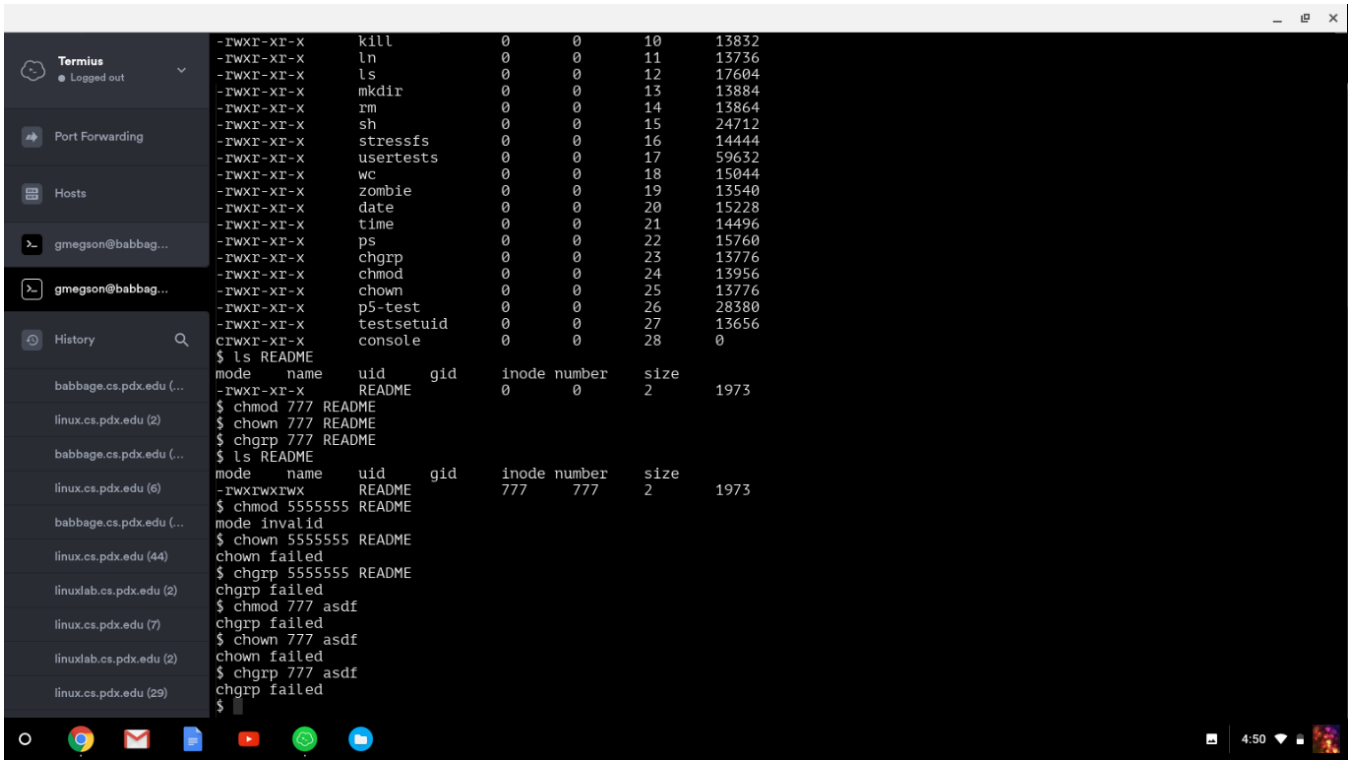

Figure 8: Other tests

This test `PASSES`.

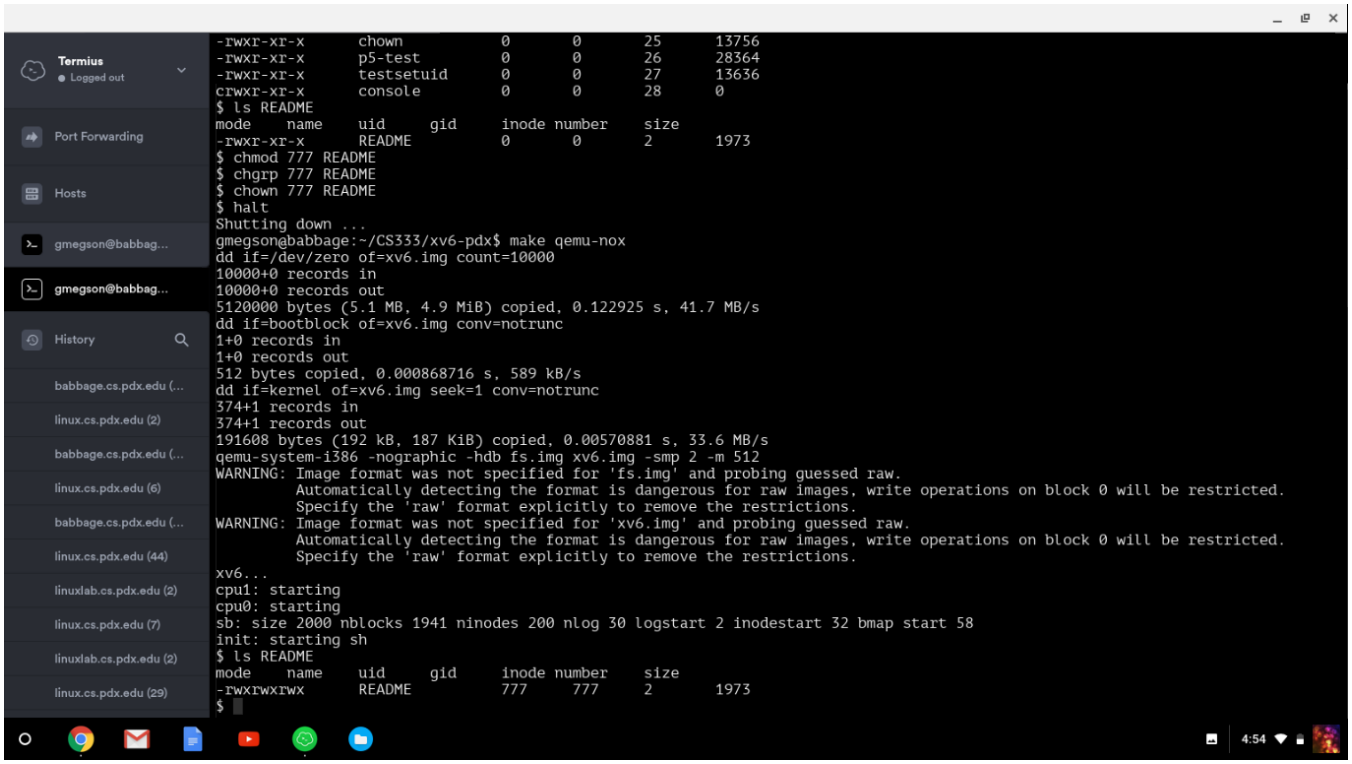

Figure 9: Other tests