Project 5 Report
CS333 - Intro to Operating Systems
Winter 2018

Ryan Hoover

3/11/18

0.1 Description

0.1.1 New File Permissions & Modifications For `exec`

File permissions will be added to xv6 by implementing *modes* for files/directories/devices. Mode will be interpreted as both a numeric value and a 12-bit binary vector, depending on the functionality. In mode, values at different bits correspond to users, groups, and other permissions.
The `exec` system call will require two changes. First, the file should not be read unless the process has execute permission for the file. Second, when the new image is committed, if the file is `setuid` then the new `uid` will need to be set.

0.1.2 Updated `ls` Command

The `ls` command has been modified to display the new `uid`, `gid`, and `mode` fields.

- The first column in the `ls` print will display the mode bits for the file/directory/device. The first character indicates if the item is a regular file ('-'), directory ('d'), or device file ('c'). If the file is `setuid`, then the 'x' in the user permissions will be displayed as 'S'.
- `name`. The name column will continue to show the name of the file or directory.
- `uid`. The user identifier (owner) of the file or directory will be displayed.
- `gid`. The group identifier (owning group) of the file or directory will be displayed.
- `inode`. Inode number of the file or directory.
- `size`. Size in bytes of the file or directory.

0.1.3 New Commands & System Calls to Modify Permissions

New System Calls:

- The `chmod()` system call sets the mode, or permission, bits for the specified target.
- The `chown()` system call sets the user UID for the specified target.
- The `chgrp()` system call sets the group GID for the specified target.

New Commands:
Three new commands allow the user to set the UID, GID, or mode for a file or directory.

- `chown`: sets the owner (UID) for a file or directory.
- `chgrp`: sets the group (GID) for a file or directory.
- `chmod`: sets the mode bits (permissions) for a file or directory.

0.2 Deliverables

The following section contains detailed descriptions of the features added to xv6 for Project 5.

0.2.1 `inode` / `dnode` / `stat`

New File Permissions (user, group, & other)

The new *user*, *group*, and *other* permission modes will determine the level of access a process will have in the system: read, write, and execute. Fields for User ID, Group ID, and Mode will be added to the `inode` and `dinode` structures. User and Group IDs of a file or directory will be checked against the calling process's UID and GID fields to determine permissions. Mode is a number meant to represent different values for permissions.

New `mode_t` and `stat_mode_t` Unions & Their Usage

Permission modes will be represented as bits, and defined as a `union`, called `mode_t`. This `union` will be added to the `inode` and `dinode` structures alongside `uid` and `gid`. Another `union` will be added to the `stat` structure, the same format as `mode_t`, but renamed `stat_mode_t`. This renaming is due to how the kernel handles `#include` files.
The mode bits for user/group/other will be interpreted as "read write execute" permissions (rwx). Bits 0-2 are the mode bits for "other"; bits 3-5 are for "group"; bits 6-8 are for "user"; bit 9 is the "setuid" indicator; and the remaining bits in the integer field are unused.

Persistence of New File Permissions

The `inode` and `dinode` structures will maintain metadata about files stored to the disk. The `mode` bits we added to these structures will contain information about the file permissions, thus ensuring that the permissions persist when the OS is booted and run again. The xv6 file system uses a log to help maintain file system consistency. Operations to disk are very slow. As a result, all file system operations are logged before being committed to the disk. In this way, if the disk commit process is interrupted, the actions may be replayed from the log. This also means that there must be a way to ensure that the log is not corrupted. The xv6 file system uses a *transactional* approach. All operations that result in a file system modification are wrapped in a transaction.

0.2.2 `chmod` / `chown` / `chgrp`

New System Calls `chmod`, `chown`, & `chgrp`

The prototypes for the three new system calls are as follows:

1. `int chmod(char *pathname, int mode);`
   The `chmod()` system call sets the mode, or permission, bits for the target specified by `pathname`. The return value is '0' on success and '-1' on failure.

2. `int chown(char *pathname, int owner);`
   The `chown()` system call sets the UID for the target specified by `pathname`. The return value is '0' on success and '-1' on failure.

3. `int chgrp(char *pathname, int group);`
   The `chgrp()` system call sets the GID for the target specified by `pathname`. The return value is '0' on success and '-1' on failure.

New Commands `chmod`, `chown`, & `chgrp`

Three new commands will allow the user to set the UID, GID, or mode for a file or directory, described as follows:

1. `chown`: sets the owner (UID) for a file or directory.
   Usage: `chown OWNER TARGET`
   where `OWNER` is the numeric UID to set as the owner of `TARGET` and `TARGET` is the name of a file or directory.

2. `chgrp`: sets the group (GID) for a file or directory.
   Usage: `chgrp GROUP TARGET`
   where `GROUP` is the numeric GID to set as the group of `TARGET` and `TARGET` is thr name of a file or directory.

3. `chmod`: sets the mode bits (permissions) for a file or directory.
   Usage: `chmod MODE TARGET`
   where `MODE` is a string of octal values specifying mode bits to set for `TARGET` and `TARGET` is the name of a file or directory.
   The numeric `MODE` is four octal digits (0 - 7), derived by adding up the bits with values 4, 2, and 1. No digit may be omitted, and any digit can be the value 0. The first digit selects the set UID (1) attribute. The second digit selects permissions for the user who owns the file: read (4), write (2), and execute (1). The third selects permissions for other users in the file's group. The fourth digit is for other users not in the file's group, with the same values. It is permissible for one of user / group / other to be 0, which indicates no access to the file, e.g. `chmod 0000 TARGET` is valid.
   Mode bits are evaluated in this order:

   (a) User: Check against the user permissions. If the UID of the invoking process and the UID of the file are the same, use these permissions.
   (b) Group: Check against the group permission. If the GID of the invoking process and the GID of the file are the same, use these permissions
   (c) Other: If neither the user or group permissions are used, apply these permissions.
   (d) Setuid: If the permissions allow the invoking process to execute the file, set the UID of the process executing the file to be the same as the UID for the file.

Valid Integer Ranges

The `chmod`, `chown`, and `chgrp` user commands do not modify anything when passed in an invalid filename or numeric argument (mode digits for `chmod`, UID for `chown`, and GID for `chgrp`).

0.2.3 `ls` Command

The `ls` command must be modified to display the new fields. A new routine, `print_mode` is provided in the file `print_mode.c` to handle the formatting. Since output is getting crowded, we provide a header that labels each column in the output. The source code file `print_mode.c` can be included directly into the `ls.c` source code.

Changes to Header, Columns

The header will denote the information present in each of the columns.

`mode`. The first column will display the mode bits for the file/directory/device.

`name`. Name of the file or directory.

`uid`. User identifier (owner) of the file or directory.

`gid`. Group identifier (owning group) of the file or directory.

`inode`. Inode number of the file or directory.

`size`. Size in bytes of the file or directory.

Mode Column Values

The first character of the mode bits indicates if the item is a regular file ('-'), directory ('d'), or device file ('c'). If the file is setuid, then the 'x' in the user permissions will be displayed as 'S'.

0.2.4 `exec` System Call

The `exec()` system call now takes into account the execute permissions of user / group / other, only executing the process if proper permissions exist, otherwise will return failure.

New `exec` Permission Check

The `exec` system call will check permissions before executing a process. The file (`inode`) UID and GID are matched against the waiting-to-execute process (`proc`), and the execute mode bits for user, group, and other are checked. If the file permissions exist and the proper execute bits are valid (execute flag is on), then `exec()` proceeds as usual.

`setuid` Functionality

After permission is determined, and right before the context switch takes place to run the process, if the `setuid` flag is on, the process's UID is updated to the UID of the file (`inode`).

0.3 Implementation

This section describes all files and line numbers for modifications, without including screenshots of code.

0.3.1 `inode` / `dnode` / `stat`

- *New* `mode_t` *union in* `fs.h` *and addition of* `uid`, `gid`, *and* `mode` *fields to* `inode` *and* `dnode`
  - `fs.h`.
    - (Lines 23 – 27). Constant `N_DIRECT` is redefined from 12 to 10 if the `CS333_P5` flag is on.
    - (Lines 31 – 48). `union mode_t` is defined.
    - (Lines 56 – 60). `uid`, `gid`, and `mode` fields are added to the `dinode` structure.
  - `file.h`. (Lines 23 – 27). `uid`, `gid`, and `mode` fields are added to the `inode` structure.
- *New* `stat_mode_t` *union in* `stat.h` *and addition of* `uid`, `gid`, *and* `mode` *fields to stat*
  In `stat.h`:
  - (Lines 5 – 22). `union stat_mode_t` is defined; identical to `union mode_t` in `fs.h`.
  - (Lines 29 – 33). `uid`, `gid`, and `mode` fields are added to `struct stat`.
- *New default* `mode` *define in* `param.h`
  (Lines 23 – 25). `mode` is given a default value of octal `0755`. Additional `#define`s of default UID and GID values are defined here as well, to not interfere with the default UID and GID from Project 2.
- *Updated functions in* `fs.c` *to initialize / copy new* `uid`, `gid`, *and* `mode` *fields*
  - `ialloc()` (Lines 189 – 193). The `uid`, `gid`, and `mode` fields of the `inode` are initialized to their default values, as defined in `param.h`.
  - `iupdate()` (Lines 216 – 220). The `uid`, `gid`, and `mode` fields are copied from a modified in-memory `inode` to disk (`dinode`).
  - `ilock()` (Lines 298 – 302). Read the `inode` from disk if necessary.
  - `stati()` (Lines 445 – 449). Copy `stat` information from `inode`.
- *Updated function in* `mkfs.c` *to initialize* `uid`, `gid`, *and* `mode` *fields*
  `ialloc()` (Lines 232 – 236). The `uid`, `gid`, and `mode` fields of the `dinode` are initialized to their default values, as defined in `param.h`.

0.3.2 `chmod` / `chown` / `chgrp` System Calls

- *System call handlers is* `sysfile.c`
  - `sys_chmod` (Lines 447 – 464). Pulls arguments from the stack, performs a bounds check on the octal `mode` argument (`0 <= mode <= 1023`), and passes arguments off the user-side implementation in `fs.c`.
  - `sys_chown` (Lines 466 – 483). Pulls arguments off the stack, performs a bounds check on the integer `uid` argument (`0 <= uid <= 32767`), and passes arguments off to the user-side implementation in `fs.c`.
  - `sys_chgrp` (Lines 485 – 502). Pulls arguments off the stack, performs a bounds check on the integer `gid` argument (`0 <= gid <= 32767`), and passes arguments off to the user-side implementation in `fs.c`.

- *Implementation in* `fs.c`*, including locking and updating of* `inode`
  - `int chmod(char *pathname, int mode);` (Lines 677 – 691).
    A `struct inode` is declared, and `begin_op()` is called to begin the transaction. The temporary `inode` is populated by passing the argument `pathname` to the `namei()` function (unless the supplied path is invalid, which will end the transaction and return failure). We then lock the `inode` with a call to `ilock()`, and assign the `inode`'s `mode` field to the `mode` argument value. The `inode` is updated and committed to disk with calls to `iupdate()` and `iunlockput()`, the transaction is ended with a call to `end_op()`, and the function returns success.
  - `int chown(char *pathname, int owner);` (Lines 695 – 709).
    A `struct inode` is declared, and `begin_op()` is called to begin the transaction. The temporary `inode` is populated by passing the argument `pathname` to the `namei()` function (unless the supplied path is invalid, which will end the transaction and return failure). We then lock the `inode` with a call to `ilock()`, and assign the `inode`'s `uid` field to the `owner` argument value. The `inode` is updated and committed to disk with calls to `iupdate()` and `iunlockput()`, the transaction is ended with a call to `end_op()`, and the function returns success.
  - `int chgrp(char *pathname, int group);` (Lines 713 – 727).
    A `struct inode` is declared, and `begin_op()` is called to begin the transaction. The temporary `inode` is populated by passing the argument `pathname` to the `namei()` function (unless the supplied path is invalid, which will end the transaction and return failure). We then lock the `inode` with a call to `ilock()`, and assign the `inode`'s `gid` field to the `group` argument value. The `inode` is updated and committed to disk with calls to `iupdate()` and `iunlockput()`, the transaction is ended with a call to `end_op()`, and the function returns success.
- *Bounds checking for mode in* `sys_chmod`
  The `mode` argument value is represented as an octal integer, and its bounds must be checked as the highest permissions the machine will support, `1777`. This octal number represents the `setuid` bit set, with read/write/execute permissions set for uid/gid/other. Octal `1777` converted to decimal is `1023`, which `sys_chmod` checks for before passing the arguments off to the user-side implementation in `fs.c` (Lines 459 – 461).
- *Correct use of integer portion of mode union for setting in* `sys_chmod`
  As discussed above, the integer equivalent of the octal number is used. The `mode.asInt` field is used to assign `mode` values for the user-side function calls.

0.3.3 `chmod` / `chown` / `chgrp` User Commands

- *New commands* `chmod`, `chown`, *and* `chgrp`
  New user commands for the `chmod`, `chown`, and `chgrp` system calls were added to xv6, as 3 files:

  ```
  chmod.c
  chown.c
  chgrp.c
  ```

  Additionally, the following existing files were edited to add the system call functionality to xv6:
  - `usys.S` (Lines 41 – 43). Added user-side stubs for the new system calls.
  - `syscall.h` (Lines 35 – 37). The `chmod`, `chown`, and `chgrp` system call numbers were created by appending to the existing list.
  - `syscall.c`. Modified to include the kernel-side function prototype (Lines 117 – 119); an entry in the function dispatch table `syscalls[]` (Lines 160 – 162); and an entry into the `syscallnames[]` array to print the system call name when the `PRINT_SYSCALLS` flag is defined (Lines 207 – 209).

  - `user.h` (Lines 49 – 51). Added the user-side function prototypes for the system calls. Prototypes are defined in section 0.2.2 (p.2) of this document. Added the prototype for the predefined `atoo()` function, for use with the `chmod` user command (Line 68).
  - `defs.h` (Lines 58 – 60). Added function prototypes for the new system calls.

- *Parsing of user input string and conversion to int for* `uid`*,* `gid`*, and* `mode`
  All 3 user commands take 2 arguments from the xv6 command line, a `pathname`, and an integer associated with either UID, GID, or mode. The main functionality of all three programs are largely the same, to the point where explaining one can explain all of them. If given more time, perhaps it would be advantageous to roll the common functionality into a helper function that can be shared between all three.
  With one major difference, which will be pointed out, the following implementation describes all three user command programs (`chmod / chown / chgrp`): The user commands begin by checking the that the number of supplied arguments is equal to 3 (one for the command name, one for the `pathname`, and one for `owner / group / mode`). The command fails if there are not exactly 3 arguments (Lines 13 – 16). Temporary variables for `pathname` and `uid / gid / mode`, and the arguments are parsed from the command line.
  This is where the major difference lies: In `chown` and `chgrp`, the `atoi()` function is used to convert the argument string into an integer, for the `uid` and `gid`; while in `chmod`, the `atoo()` function is used to convert the string into an *octal* integer, for the `mode` (Line 32). There is a series of checks in `chmod` that don't exist in `chown` or `chgrp`, which checks the values of the `argv[1]` string at each index to ensure that it falls somewhere between `0000` and `1777` (`chmod.c`, Lines 19 – 29). An integer, `value`, is set to 1 if the string is determined to be valid, and `atoo` is called to convert the string to an octal. If `valid` was not set, `chmod` exits without setting the values (Line 35). The arguments are then passed to the system call (Line 21), the return value is checked for errors (Lines 22 – 29), and the program exits (`chmod()`, Lines 38 – 47).

0.3.4 `ls` Command

- *Inclusion and usage of* `print_mode`
  The supplied `print_mode` program is included in `ls.c` (Line 6), and used to print out the mode bits in the reformatted `ls` output (Lines 53 & 79).

- *Modification to print statements and new header print*
  The original print statements were wrapped in an `#ifndef CS333_P5` statement, and the updated header follows the associated `#else` (Lines 45 – 51; 75 – 77). The new print statements follow the `print_mode()` function calls, and add in the new information to fill out each column (Lines 54 & 80). The new `ls` header is defined in the program's `main()` (Line 95).

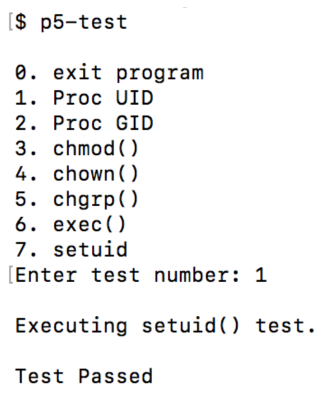
0.3.5 `exec` System Call

In `exec.c`:

- *Copying to* `stat` *struct to read execute permissions for file's* `inode`
  A `stat` struct is defined and relevant data from the `inode` is copied (Lines 35, 36).

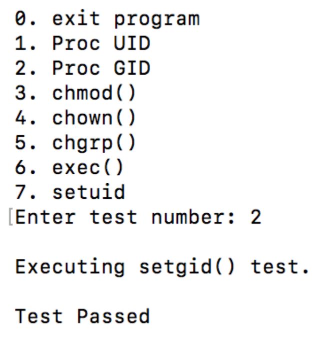
- *Checking if process has execute permissions based on appropriate user, group, or other execute flag*
  The `stat` structure's UID is checked against the current `proc`'s UID alongside checking the validity of the `stat`'s user execute permission bit (`u_x`); if that fails, the process is then repeated for matching GID's and validity of the group execute permission bit (`g_x`); if that too fails, then the validity of the other permission bit (`o_x`)is checked (Lines 37 – 45). If any of these conditions are true, a `goto` statement, "`good`", directs control to the rest of the `exec` function (Line 51). If all of the conditions fail, then the process does not have permission to execute, and `exec` returns failure (Line 47).

- *Checking* `setuid` *flag and changing the process uid if necessary*
  Before the context switch occurs to put the process into the CPU, the `stat` structure's `setuid` bit is checked for validity, and if true, sets `proc->uid` to the value of `stat`'s current UID (Lines 122 – 124).



```
[$ p5-test

0. exit program
1. Proc UID
2. Proc GID
3. chmod()
4. chown()
5. chgrp()
6. exec()
7. setuid
[Enter test number: 1

Executing setuid() test.

Test Passed
```

Figure 1: `p5-test` Proc UID test.



```
0. exit program
1. Proc UID
2. Proc GID
3. chmod()
4. chown()
5. chgrp()
6. exec()
7. setuid
[Enter test number: 2

Executing setgid() test.

Test Passed
```

Figure 2: `p5-test` Proc GID test.

0.4 Testing

0.4.1 `p5-test` Command

1. *Proc UID*
   Tests proper functionality of the `setuid()` system call (Figure 1).
   This sub-test `PASSES`.

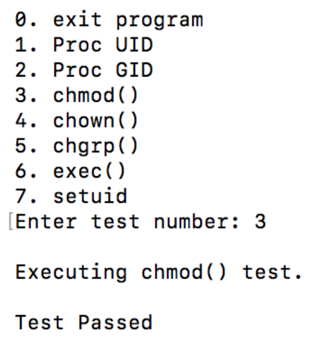
2. *Proc GID*
   Tests proper functionality of the `setgid()` system call (Figure 2).
   This sub-test `PASSES`.

3. *chmod*
   Tests proper functionality of the `chmod()` system call (Figure 3).
   This sub-test `PASSES`.

4. *chown*
   Tests proper functionality of the `chown()` system call (Figure 4).
   This sub-test `PASSES`.

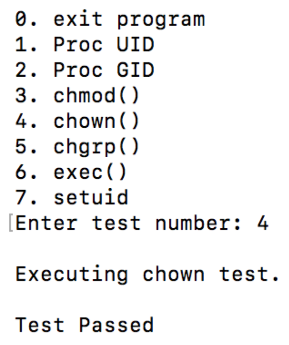
5. *chgrp*
   Tests proper functionality of the `chgrp()` system call (Figure 5).
   This sub-test `PASSES`.

6. *exec*
   Tests for proper permissions, and if the permission checks work, it forks and runs a few processes.



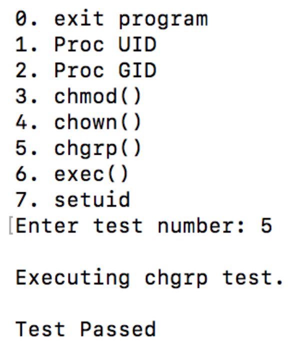
```
0. exit program
1. Proc UID
2. Proc GID
3. chmod()
4. chown()
5. chgrp()
6. exec()
7. setuid
[Enter test number: 3

Executing chmod() test.

Test Passed
```

Figure 3: `p5-test` chmod test.



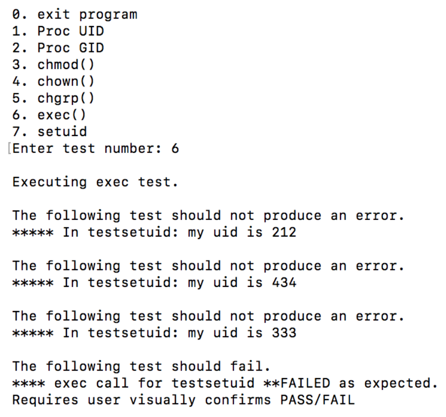
```
0. exit program
1. Proc UID
2. Proc GID
3. chmod()
4. chown()
5. chgrp()
6. exec()
7. setuid
[Enter test number: 4

Executing chown test.

Test Passed
```

Figure 4: `p5-test` chown test.



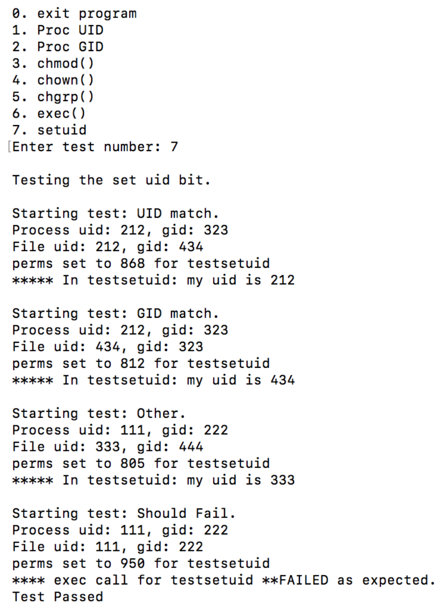
```
0. exit program
1. Proc UID
2. Proc GID
3. chmod()
4. chown()
5. chgrp()
6. exec()
7. setuid
[Enter test number: 5

Executing chgrp test.

Test Passed
```

Figure 5: `p5-test` chgrp test.



```
0. exit program
1. Proc UID
2. Proc GID
3. chmod()
4. chown()
5. chgrp()
6. exec()
7. setuid
Enter test number: 6

Executing exec test.

The following test should not produce an error.
***** In testsetuid: my uid is 212

The following test should not produce an error.
***** In testsetuid: my uid is 434

The following test should not produce an error.
***** In testsetuid: my uid is 333

The following test should fail.
**** exec call for testsetuid **FAILED as expected.
Requires user visually confirms PASS/FAIL
```

Figure 6: `p5-test` Exec test.



```
0. exit program
1. Proc UID
2. Proc GID
3. chmod()
4. chown()
5. chgrp()
6. exec()
7. setuid
Enter test number: 7

Testing the set uid bit.

Starting test: UID match.
Process uid: 212, gid: 323
File uid: 212, gid: 434
perms set to 868 for testsetuid
***** In testsetuid: my uid is 212

Starting test: GID match.
Process uid: 212, gid: 323
File uid: 434, gid: 323
perms set to 812 for testsetuid
***** In testsetuid: my uid is 434

Starting test: Other.
Process uid: 111, gid: 222
File uid: 333, gid: 444
perms set to 805 for testsetuid
***** In testsetuid: my uid is 333

Starting test: Should Fail.
Process uid: 111, gid: 222
File uid: 111, gid: 222
perms set to 950 for testsetuid
**** exec call for testsetuid **FAILED as expected.
Test Passed
```

Figure 7: `p5-test` setuid test.

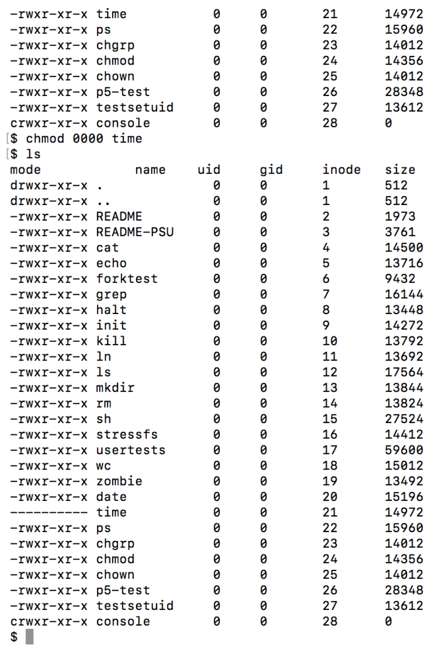
If the processes with the proper permissions are executed, and the test without proper permission is not executed, the user must inspect the outputs to determine pass/fail (Figure 6).
This sub-test `PASSES`.

7. *setuid*
This tests the functionality of the `mode.setuid` bit, and if the exec process will update its UID when run (Figure 7).
This sub-test `PASSES`.

Result: All sub-tests pass; Test 1 `PASSES`

0.4.2 `chmod` Command

1. *Changes mode when given valid parameters*
The first test is expected to pass, given several valid parameters. First, `chmod` is given a mode value of `0000` for the `time` command (Figure 8), eliminating all permissions. Next, the value given is the



```
-rwxr-xr-x time          0     0      21      14972
-rwxr-xr-x ps            0     0      22      15960
-rwxr-xr-x chgrp         0     0      23      14012
-rwxr-xr-x chmod         0     0      24      14356
-rwxr-xr-x chown         0     0      25      14012
-rwxr-xr-x p5-test       0     0      26      28348
-rwxr-xr-x testsetuid    0     0      27      13612
crwxr-xr-x console       0     0      28      0
$ chmod 0000 time
$ ls
mode            name    uid     gid     inode   size
drwxr-xr-x .             0     0      1       512
drwxr-xr-x ..            0     0      1       512
-rwxr-xr-x README        0     0      2       1973
-rwxr-xr-x README-PSU    0     0      3       3761
-rwxr-xr-x cat           0     0      4       14500
-rwxr-xr-x echo          0     0      5       13716
-rwxr-xr-x forktest      0     0      6       9432
-rwxr-xr-x grep          0     0      7       16144
-rwxr-xr-x halt          0     0      8       13448
-rwxr-xr-x init          0     0      9       14272
-rwxr-xr-x kill          0     0      10      13792
-rwxr-xr-x ln            0     0      11      13692
-rwxr-xr-x ls            0     0      12      17564
-rwxr-xr-x mkdir         0     0      13      13844
-rwxr-xr-x rm            0     0      14      13824
-rwxr-xr-x sh            0     0      15      27524
-rwxr-xr-x stressfs      0     0      16      14412
-rwxr-xr-x usertests     0     0      17      59600
-rwxr-xr-x wc            0     0      18      15012
-rwxr-xr-x zombie        0     0      19      13492
-rwxr-xr-x date          0     0      20      15196
---------- time          0     0      21      14972
-rwxr-xr-x ps            0     0      22      15960
-rwxr-xr-x chgrp         0     0      23      14012
-rwxr-xr-x chmod         0     0      24      14356
-rwxr-xr-x chown         0     0      25      14012
-rwxr-xr-x p5-test       0     0      26      28348
-rwxr-xr-x testsetuid    0     0      27      13612
crwxr-xr-x console       0     0      28      0
$
```

Figure 8: `chmod` given value of 0000 for `time` command.

maximum, `1777` , setting all permissions and the setuid bit (Figure 9). Next, the value `0111` is given, changing the `time` permissions to execute only for user, group, and other (Figure 10).
This sub-test `PASSES`.

2. *Fails and does not change mode when given invalid filename*
   For this test, `chmod` was given a pathname for a file that does not exist within the system, "biscuit". A valid mode is given, the defalut 0755. The nonsensical word is used to demonstrate the expected failure of the `chmod` command (Figure 11).
   This sub-test `PASSES`.

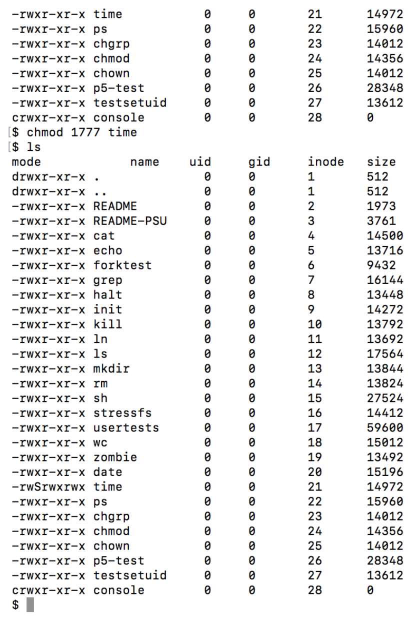
3. *Fails and does not change mode when given invalid mode*
   For this test, `chmod` was given a value far outside the scope of what a valid mode is. Expecting a failure, a value of 8,888,888. A valid path name is provided, `time`. The test fails as expected and the `time` file's permissions are unaffected (Figure 12).
   This sub-test `PASSES`.

Result: All sub-tests pass; Test 2 `PASSES`.

0.4.3 `chown` Command

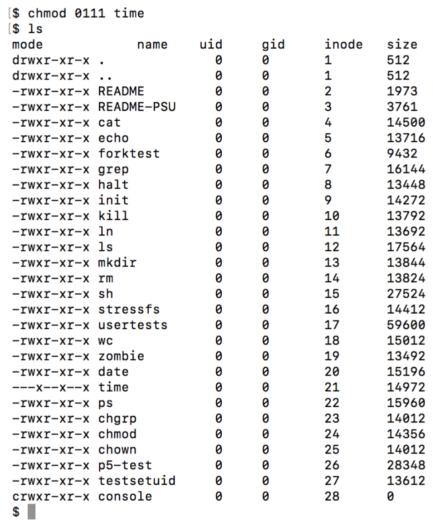
1. *Changes UID when given valid parameters*
   This test is expected to pass, given a valid parameter. To conduct this test, I entered a valid UID for the `time` command (Figure 13).
   This sub-test `PASSES`.

2. *Fails and does not change UID when given invalid filename*
   This test is expected to fail, when given an invalid filename. To conduct this test, I entered a filename for a file that doesn't exist: "biscuit" (Figure 14).
   This sub-test `PASSES`.



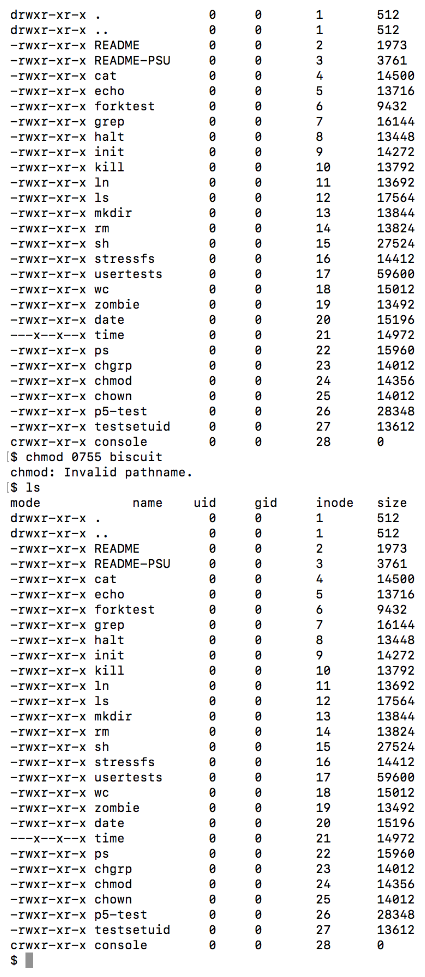
```
-rwxr-xr-x time          0     0       21      14972
-rwxr-xr-x ps            0     0       22      15960
-rwxr-xr-x chgrp         0     0       23      14012
-rwxr-xr-x chmod         0     0       24      14356
-rwxr-xr-x chown         0     0       25      14012
-rwxr-xr-x p5-test       0     0       26      28348
-rwxr-xr-x testsetuid    0     0       27      13612
crwxr-xr-x console       0     0       28      0
$ chmod 1777 time
$ ls
mode            name    uid     gid     inode   size
drwxr-xr-x .             0     0       1       512
drwxr-xr-x ..            0     0       1       512
-rwxr-xr-x README        0     0       2       1973
-rwxr-xr-x README-PSU    0     0       3       3761
-rwxr-xr-x cat           0     0       4       14500
-rwxr-xr-x echo          0     0       5       13716
-rwxr-xr-x forktest      0     0       6       9432
-rwxr-xr-x grep          0     0       7       16144
-rwxr-xr-x halt          0     0       8       13448
-rwxr-xr-x init          0     0       9       14272
-rwxr-xr-x kill          0     0       10      13792
-rwxr-xr-x ln            0     0       11      13692
-rwxr-xr-x ls            0     0       12      17564
-rwxr-xr-x mkdir         0     0       13      13844
-rwxr-xr-x rm            0     0       14      13824
-rwxr-xr-x sh            0     0       15      27524
-rwxr-xr-x stressfs      0     0       16      14412
-rwxr-xr-x usertests     0     0       17      59600
-rwxr-xr-x wc            0     0       18      15012
-rwxr-xr-x zombie        0     0       19      13492
-rwxr-xr-x date          0     0       20      15196
-rwSrwxrwx time          0     0       21      14972
-rwxr-xr-x ps            0     0       22      15960
-rwxr-xr-x chgrp         0     0       23      14012
-rwxr-xr-x chmod         0     0       24      14356
-rwxr-xr-x chown         0     0       25      14012
-rwxr-xr-x p5-test       0     0       26      28348
-rwxr-xr-x testsetuid    0     0       27      13612
crwxr-xr-x console       0     0       28      0
$
```

Figure 9: `chmod` given value of 1777 for `time` command.



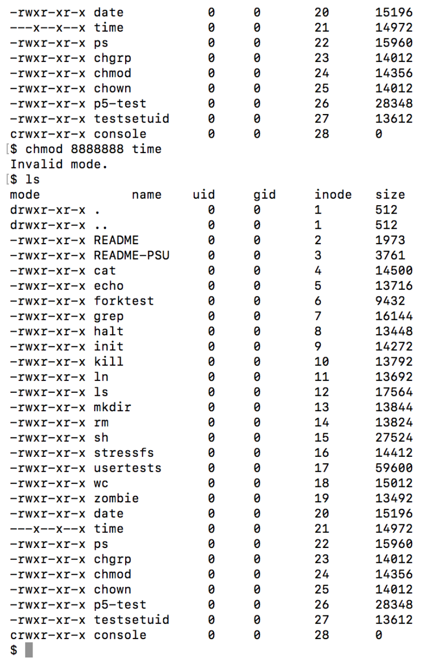
```
$ chmod 0111 time
$ ls
mode            name    uid     gid     inode   size
drwxr-xr-x .             0     0       1       512
drwxr-xr-x ..            0     0       1       512
-rwxr-xr-x README        0     0       2       1973
-rwxr-xr-x README-PSU    0     0       3       3761
-rwxr-xr-x cat           0     0       4       14500
-rwxr-xr-x echo          0     0       5       13716
-rwxr-xr-x forktest      0     0       6       9432
-rwxr-xr-x grep          0     0       7       16144
-rwxr-xr-x halt          0     0       8       13448
-rwxr-xr-x init          0     0       9       14272
-rwxr-xr-x kill          0     0       10      13792
-rwxr-xr-x ln            0     0       11      13692
-rwxr-xr-x ls            0     0       12      17564
-rwxr-xr-x mkdir         0     0       13      13844
-rwxr-xr-x rm            0     0       14      13824
-rwxr-xr-x sh            0     0       15      27524
-rwxr-xr-x stressfs      0     0       16      14412
-rwxr-xr-x usertests     0     0       17      59600
-rwxr-xr-x wc            0     0       18      15012
-rwxr-xr-x zombie        0     0       19      13492
-rwxr-xr-x date          0     0       20      15196
---x--x--x time          0     0       21      14972
-rwxr-xr-x ps            0     0       22      15960
-rwxr-xr-x chgrp         0     0       23      14012
-rwxr-xr-x chmod         0     0       24      14356
-rwxr-xr-x chown         0     0       25      14012
-rwxr-xr-x p5-test       0     0       26      28348
-rwxr-xr-x testsetuid    0     0       27      13612
crwxr-xr-x console       0     0       28      0
$
```

Figure 10: `chmod` given value of 0111 for `time` command.



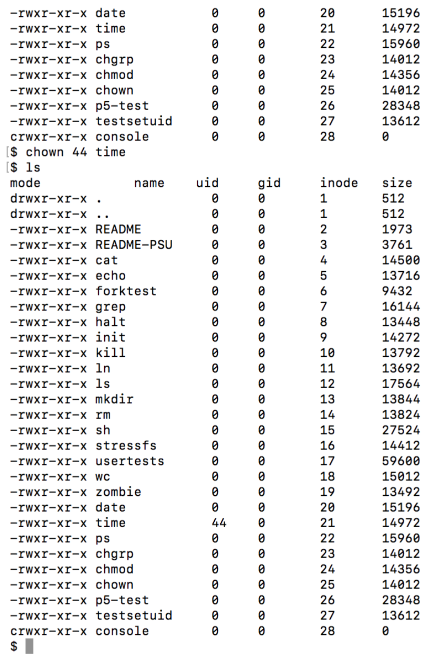
```
drwxr-xr-x .            0     0      1      512
drwxr-xr-x ..           0     0      1      512
-rwxr-xr-x README       0     0      2      1973
-rwxr-xr-x README-PSU   0     0      3      3761
-rwxr-xr-x cat          0     0      4      14500
-rwxr-xr-x echo         0     0      5      13716
-rwxr-xr-x forktest     0     0      6      9432
-rwxr-xr-x grep         0     0      7      16144
-rwxr-xr-x halt         0     0      8      13448
-rwxr-xr-x init         0     0      9      14272
-rwxr-xr-x kill         0     0      10     13792
-rwxr-xr-x ln           0     0      11     13692
-rwxr-xr-x ls           0     0      12     17564
-rwxr-xr-x mkdir        0     0      13     13844
-rwxr-xr-x rm           0     0      14     13824
-rwxr-xr-x sh           0     0      15     27524
-rwxr-xr-x stressfs     0     0      16     14412
-rwxr-xr-x usertests    0     0      17     59600
-rwxr-xr-x wc           0     0      18     15012
-rwxr-xr-x zombie       0     0      19     13492
-rwxr-xr-x date         0     0      20     15196
---x--x--x time         0     0      21     14972
-rwxr-xr-x ps           0     0      22     15960
-rwxr-xr-x chgrp        0     0      23     14012
-rwxr-xr-x chmod        0     0      24     14356
-rwxr-xr-x chown        0     0      25     14012
-rwxr-xr-x p5-test      0     0      26     28348
-rwxr-xr-x testsetuid   0     0      27     13612
crwxr-xr-x console      0     0      28     0
$ chmod 0755 biscuit
chmod: Invalid pathname.
$ ls
mode            name    uid     gid     inode   size
drwxr-xr-x .            0     0      1      512
drwxr-xr-x ..           0     0      1      512
-rwxr-xr-x README       0     0      2      1973
-rwxr-xr-x README-PSU   0     0      3      3761
-rwxr-xr-x cat          0     0      4      14500
-rwxr-xr-x echo         0     0      5      13716
-rwxr-xr-x forktest     0     0      6      9432
-rwxr-xr-x grep         0     0      7      16144
-rwxr-xr-x halt         0     0      8      13448
-rwxr-xr-x init         0     0      9      14272
-rwxr-xr-x kill         0     0      10     13792
-rwxr-xr-x ln           0     0      11     13692
-rwxr-xr-x ls           0     0      12     17564
-rwxr-xr-x mkdir        0     0      13     13844
-rwxr-xr-x rm           0     0      14     13824
-rwxr-xr-x sh           0     0      15     27524
-rwxr-xr-x stressfs     0     0      16     14412
-rwxr-xr-x usertests    0     0      17     59600
-rwxr-xr-x wc           0     0      18     15012
-rwxr-xr-x zombie       0     0      19     13492
-rwxr-xr-x date         0     0      20     15196
---x--x--x time         0     0      21     14972
-rwxr-xr-x ps           0     0      22     15960
-rwxr-xr-x chgrp        0     0      23     14012
-rwxr-xr-x chmod        0     0      24     14356
-rwxr-xr-x chown        0     0      25     14012
-rwxr-xr-x p5-test      0     0      26     28348
-rwxr-xr-x testsetuid   0     0      27     13612
crwxr-xr-x console      0     0      28     0
$
```

Figure 11: `chmod` fails when given an invalid path name.



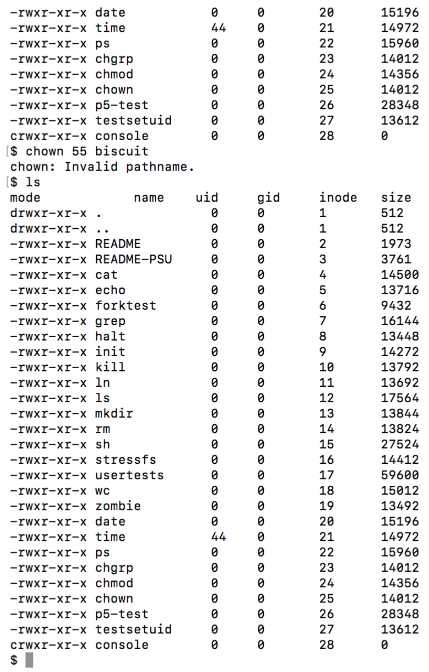
```
-rwxr-xr-x date         0     0      20     15196
---x--x--x time         0     0      21     14972
-rwxr-xr-x ps           0     0      22     15960
-rwxr-xr-x chgrp        0     0      23     14012
-rwxr-xr-x chmod        0     0      24     14356
-rwxr-xr-x chown        0     0      25     14012
-rwxr-xr-x p5-test      0     0      26     28348
-rwxr-xr-x testsetuid   0     0      27     13612
crwxr-xr-x console      0     0      28     0
$ chmod 8888888 time
Invalid mode.
$ ls
mode            name    uid     gid     inode   size
drwxr-xr-x .            0     0      1      512
drwxr-xr-x ..           0     0      1      512
-rwxr-xr-x README       0     0      2      1973
-rwxr-xr-x README-PSU   0     0      3      3761
-rwxr-xr-x cat          0     0      4      14500
-rwxr-xr-x echo         0     0      5      13716
-rwxr-xr-x forktest     0     0      6      9432
-rwxr-xr-x grep         0     0      7      16144
-rwxr-xr-x halt         0     0      8      13448
-rwxr-xr-x init         0     0      9      14272
-rwxr-xr-x kill         0     0      10     13792
-rwxr-xr-x ln           0     0      11     13692
-rwxr-xr-x ls           0     0      12     17564
-rwxr-xr-x mkdir        0     0      13     13844
-rwxr-xr-x rm           0     0      14     13824
-rwxr-xr-x sh           0     0      15     27524
-rwxr-xr-x stressfs     0     0      16     14412
-rwxr-xr-x usertests    0     0      17     59600
-rwxr-xr-x wc           0     0      18     15012
-rwxr-xr-x zombie       0     0      19     13492
-rwxr-xr-x date         0     0      20     15196
---x--x--x time         0     0      21     14972
-rwxr-xr-x ps           0     0      22     15960
-rwxr-xr-x chgrp        0     0      23     14012
-rwxr-xr-x chmod        0     0      24     14356
-rwxr-xr-x chown        0     0      25     14012
-rwxr-xr-x p5-test      0     0      26     28348
-rwxr-xr-x testsetuid   0     0      27     13612
crwxr-xr-x console      0     0      28     0
$
```

Figure 12: `chmod` fails when given an invalid mode value.



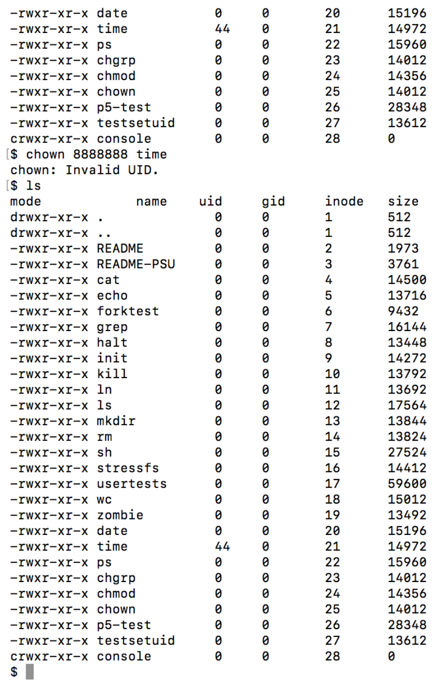
```
-rwxr-xr-x date          0     0       20      15196
-rwxr-xr-x time          0     0       21      14972
-rwxr-xr-x ps            0     0       22      15960
-rwxr-xr-x chgrp         0     0       23      14012
-rwxr-xr-x chmod         0     0       24      14356
-rwxr-xr-x chown         0     0       25      14012
-rwxr-xr-x p5-test       0     0       26      28348
-rwxr-xr-x testsetuid    0     0       27      13612
crwxr-xr-x console       0     0       28      0
$ chown 44 time
$ ls
mode            name    uid     gid     inode   size
drwxr-xr-x .             0     0       1       512
drwxr-xr-x ..            0     0       1       512
-rwxr-xr-x README        0     0       2       1973
-rwxr-xr-x README-PSU    0     0       3       3761
-rwxr-xr-x cat           0     0       4       14500
-rwxr-xr-x echo          0     0       5       13716
-rwxr-xr-x forktest      0     0       6       9432
-rwxr-xr-x grep          0     0       7       16144
-rwxr-xr-x halt          0     0       8       13448
-rwxr-xr-x init          0     0       9       14272
-rwxr-xr-x kill          0     0       10      13792
-rwxr-xr-x ln            0     0       11      13692
-rwxr-xr-x ls            0     0       12      17564
-rwxr-xr-x mkdir         0     0       13      13844
-rwxr-xr-x rm            0     0       14      13824
-rwxr-xr-x sh            0     0       15      27524
-rwxr-xr-x stressfs      0     0       16      14412
-rwxr-xr-x usertests     0     0       17      59600
-rwxr-xr-x wc            0     0       18      15012
-rwxr-xr-x zombie        0     0       19      13492
-rwxr-xr-x date          0     0       20      15196
-rwxr-xr-x time          44    0       21      14972
-rwxr-xr-x ps            0     0       22      15960
-rwxr-xr-x chgrp         0     0       23      14012
-rwxr-xr-x chmod         0     0       24      14356
-rwxr-xr-x chown         0     0       25      14012
-rwxr-xr-x p5-test       0     0       26      28348
-rwxr-xr-x testsetuid    0     0       27      13612
crwxr-xr-x console       0     0       28      0
$
```

Figure 13: `chown` changes the UID for the `time` file.



```
-rwxr-xr-x date          0     0       20      15196
-rwxr-xr-x time          44    0       21      14972
-rwxr-xr-x ps            0     0       22      15960
-rwxr-xr-x chgrp         0     0       23      14012
-rwxr-xr-x chmod         0     0       24      14356
-rwxr-xr-x chown         0     0       25      14012
-rwxr-xr-x p5-test       0     0       26      28348
-rwxr-xr-x testsetuid    0     0       27      13612
crwxr-xr-x console       0     0       28      0
$ chown 55 biscuit
chown: Invalid pathname.
$ ls
mode            name    uid     gid     inode   size
drwxr-xr-x .             0     0       1       512
drwxr-xr-x ..            0     0       1       512
-rwxr-xr-x README        0     0       2       1973
-rwxr-xr-x README-PSU    0     0       3       3761
-rwxr-xr-x cat           0     0       4       14500
-rwxr-xr-x echo          0     0       5       13716
-rwxr-xr-x forktest      0     0       6       9432
-rwxr-xr-x grep          0     0       7       16144
-rwxr-xr-x halt          0     0       8       13448
-rwxr-xr-x init          0     0       9       14272
-rwxr-xr-x kill          0     0       10      13792
-rwxr-xr-x ln            0     0       11      13692
-rwxr-xr-x ls            0     0       12      17564
-rwxr-xr-x mkdir         0     0       13      13844
-rwxr-xr-x rm            0     0       14      13824
-rwxr-xr-x sh            0     0       15      27524
-rwxr-xr-x stressfs      0     0       16      14412
-rwxr-xr-x usertests     0     0       17      59600
-rwxr-xr-x wc            0     0       18      15012
-rwxr-xr-x zombie        0     0       19      13492
-rwxr-xr-x date          0     0       20      15196
-rwxr-xr-x time          44    0       21      14972
-rwxr-xr-x ps            0     0       22      15960
-rwxr-xr-x chgrp         0     0       23      14012
-rwxr-xr-x chmod         0     0       24      14356
-rwxr-xr-x chown         0     0       25      14012
-rwxr-xr-x p5-test       0     0       26      28348
-rwxr-xr-x testsetuid    0     0       27      13612
crwxr-xr-x console       0     0       28      0
$
```

Figure 14: `chown` does not change anything when given an invalid filename.



```
-rwxr-xr-x date         0    0     20     15196
-rwxr-xr-x time         44   0     21     14972
-rwxr-xr-x ps           0    0     22     15960
-rwxr-xr-x chgrp        0    0     23     14012
-rwxr-xr-x chmod        0    0     24     14356
-rwxr-xr-x chown        0    0     25     14012
-rwxr-xr-x p5-test      0    0     26     28348
-rwxr-xr-x testsetuid   0    0     27     13612
crwxr-xr-x console      0    0     28     0
$ chown 8888888 time
chown: Invalid UID.
$ ls
mode            name   uid   gid    inode   size
drwxr-xr-x .            0    0     1      512
drwxr-xr-x ..           0    0     1      512
-rwxr-xr-x README       0    0     2      1973
-rwxr-xr-x README-PSU   0    0     3      3761
-rwxr-xr-x cat          0    0     4      14500
-rwxr-xr-x echo         0    0     5      13716
-rwxr-xr-x forktest     0    0     6      9432
-rwxr-xr-x grep         0    0     7      16144
-rwxr-xr-x halt         0    0     8      13448
-rwxr-xr-x init         0    0     9      14272
-rwxr-xr-x kill         0    0     10     13792
-rwxr-xr-x ln           0    0     11     13692
-rwxr-xr-x ls           0    0     12     17564
-rwxr-xr-x mkdir        0    0     13     13844
-rwxr-xr-x rm           0    0     14     13824
-rwxr-xr-x sh           0    0     15     27524
-rwxr-xr-x stressfs     0    0     16     14412
-rwxr-xr-x usertests    0    0     17     59600
-rwxr-xr-x wc           0    0     18     15012
-rwxr-xr-x zombie       0    0     19     13492
-rwxr-xr-x date         0    0     20     15196
-rwxr-xr-x time         44   0     21     14972
-rwxr-xr-x ps           0    0     22     15960
-rwxr-xr-x chgrp        0    0     23     14012
-rwxr-xr-x chmod        0    0     24     14356
-rwxr-xr-x chown        0    0     25     14012
-rwxr-xr-x p5-test      0    0     26     28348
-rwxr-xr-x testsetuid   0    0     27     13612
crwxr-xr-x console      0    0     28     0
$
```

Figure 15: `chown` does not change the `time` file when given an invalid UID.

3. *Fails and does not change UID when given invalid UID*
   This test is expected to fail, when given an invalid UID. To conduct this test, I entered a number well outside the bounds of a valid UID (Figure 15).
   Thus sub-test `PASSES`.

Result: All sub-tests pass; Test 3 `PASSES`.

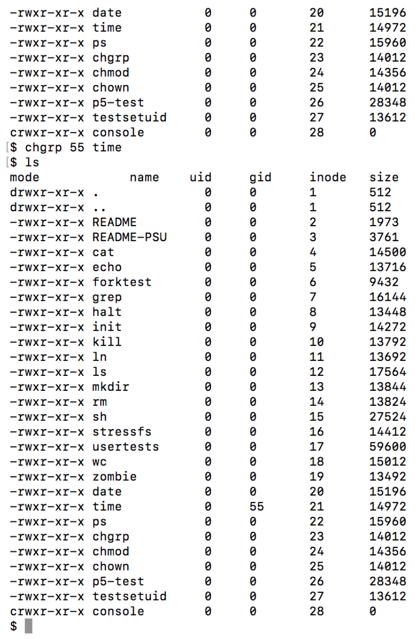
0.4.4 `chgrp` Command

1. *Changes mode when given valid parameters*
   This test is expected to pass, given a valid parameter. To conduct this test, I entered a valid GID for the `time` file (Figure 16).
   This sub-test `PASSES`.

2. *Fails and does not change GID when given invalid filename*
   This test is expected to fail, when given an invalid filename. To conduct this test, I entered a filename for a file that doesn't exist: "biscuit" (Figure 17).
   This sub-test `PASSES`.

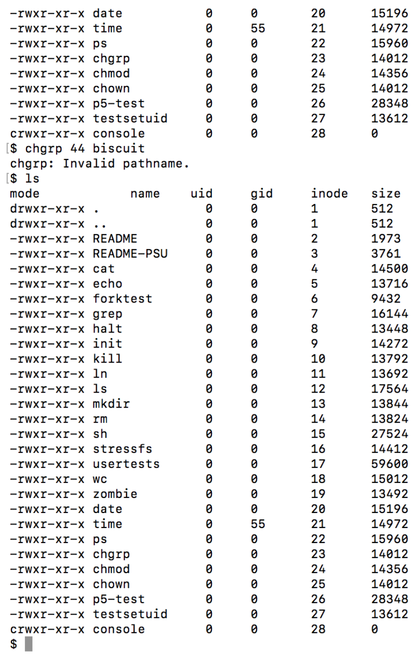
3. *Fails and does not change GID when given invalid GID*
   This test is expected to fail, when given an invalid GID. To conduct this test, I entered a number well outside the bounds of a valid GID for the `time` file (Figure 18).
   Thus sub-test `PASSES`.

Result: All sub-tests pass; Test 4 `PASSES`.



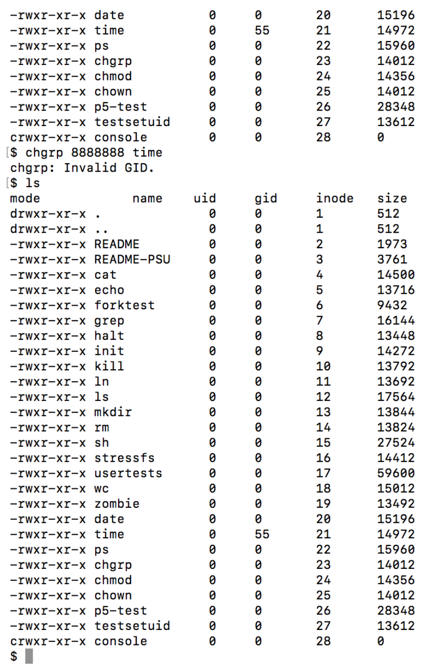
```
-rwxr-xr-x date         0    0      20     15196
-rwxr-xr-x time         0    0      21     14972
-rwxr-xr-x ps           0    0      22     15960
-rwxr-xr-x chgrp        0    0      23     14012
-rwxr-xr-x chmod        0    0      24     14356
-rwxr-xr-x chown        0    0      25     14012
-rwxr-xr-x p5-test      0    0      26     28348
-rwxr-xr-x testsetuid   0    0      27     13612
crwxr-xr-x console      0    0      28     0
$ chgrp 55 time
$ ls
mode            name    uid    gid     inode   size
drwxr-xr-x .            0      0       1       512
drwxr-xr-x ..           0      0       1       512
-rwxr-xr-x README       0      0       2       1973
-rwxr-xr-x README-PSU   0      0       3       3761
-rwxr-xr-x cat          0      0       4       14500
-rwxr-xr-x echo         0      0       5       13716
-rwxr-xr-x forktest     0      0       6       9432
-rwxr-xr-x grep         0      0       7       16144
-rwxr-xr-x halt         0      0       8       13448
-rwxr-xr-x init         0      0       9       14272
-rwxr-xr-x kill         0      0       10      13792
-rwxr-xr-x ln           0      0       11      13692
-rwxr-xr-x ls           0      0       12      17564
-rwxr-xr-x mkdir        0      0       13      13844
-rwxr-xr-x rm           0      0       14      13824
-rwxr-xr-x sh           0      0       15      27524
-rwxr-xr-x stressfs     0      0       16      14412
-rwxr-xr-x usertests    0      0       17      59600
-rwxr-xr-x wc           0      0       18      15012
-rwxr-xr-x zombie       0      0       19      13492
-rwxr-xr-x date         0      0       20      15196
-rwxr-xr-x time         0      55      21      14972
-rwxr-xr-x ps           0      0       22      15960
-rwxr-xr-x chgrp        0      0       23      14012
-rwxr-xr-x chmod        0      0       24      14356
-rwxr-xr-x chown        0      0       25      14012
-rwxr-xr-x p5-test      0      0       26      28348
-rwxr-xr-x testsetuid   0      0       27      13612
crwxr-xr-x console      0      0       28      0
$ 
```

Figure 16: `chgrp` changes the GID for the `time` file.



```
-rwxr-xr-x date         0    0      20     15196
-rwxr-xr-x time         0    55     21     14972
-rwxr-xr-x ps           0    0      22     15960
-rwxr-xr-x chgrp        0    0      23     14012
-rwxr-xr-x chmod        0    0      24     14356
-rwxr-xr-x chown        0    0      25     14012
-rwxr-xr-x p5-test      0    0      26     28348
-rwxr-xr-x testsetuid   0    0      27     13612
crwxr-xr-x console      0    0      28     0
$ chgrp 44 biscuit
chgrp: Invalid pathname.
$ ls
mode            name    uid    gid     inode   size
drwxr-xr-x .            0      0       1       512
drwxr-xr-x ..           0      0       1       512
-rwxr-xr-x README       0      0       2       1973
-rwxr-xr-x README-PSU   0      0       3       3761
-rwxr-xr-x cat          0      0       4       14500
-rwxr-xr-x echo         0      0       5       13716
-rwxr-xr-x forktest     0      0       6       9432
-rwxr-xr-x grep         0      0       7       16144
-rwxr-xr-x halt         0      0       8       13448
-rwxr-xr-x init         0      0       9       14272
-rwxr-xr-x kill         0      0       10      13792
-rwxr-xr-x ln           0      0       11      13692
-rwxr-xr-x ls           0      0       12      17564
-rwxr-xr-x mkdir        0      0       13      13844
-rwxr-xr-x rm           0      0       14      13824
-rwxr-xr-x sh           0      0       15      27524
-rwxr-xr-x stressfs     0      0       16      14412
-rwxr-xr-x usertests    0      0       17      59600
-rwxr-xr-x wc           0      0       18      15012
-rwxr-xr-x zombie       0      0       19      13492
-rwxr-xr-x date         0      0       20      15196
-rwxr-xr-x time         0      55      21      14972
-rwxr-xr-x ps           0      0       22      15960
-rwxr-xr-x chgrp        0      0       23      14012
-rwxr-xr-x chmod        0      0       24      14356
-rwxr-xr-x chown        0      0       25      14012
-rwxr-xr-x p5-test      0      0       26      28348
-rwxr-xr-x testsetuid   0      0       27      13612
crwxr-xr-x console      0      0       28      0
$ 
```

Figure 17: `chgrp` does not change anything when given an invalid filename.



```
-rwxr-xr-x date          0     0      20      15196
-rwxr-xr-x time          0     55     21      14972
-rwxr-xr-x ps            0     0      22      15960
-rwxr-xr-x chgrp         0     0      23      14012
-rwxr-xr-x chmod         0     0      24      14356
-rwxr-xr-x chown         0     0      25      14012
-rwxr-xr-x p5-test       0     0      26      28348
-rwxr-xr-x testsetuid    0     0      27      13612
crwxr-xr-x console       0     0      28      0
$ chgrp 8888888 time
chgrp: Invalid GID.
$ ls
mode            name    uid     gid     inode   size
drwxr-xr-x .             0     0      1       512
drwxr-xr-x ..            0     0      1       512
-rwxr-xr-x README        0     0      2       1973
-rwxr-xr-x README-PSU    0     0      3       3761
-rwxr-xr-x cat           0     0      4       14500
-rwxr-xr-x echo          0     0      5       13716
-rwxr-xr-x forktest      0     0      6       9432
-rwxr-xr-x grep          0     0      7       16144
-rwxr-xr-x halt          0     0      8       13448
-rwxr-xr-x init          0     0      9       14272
-rwxr-xr-x kill          0     0      10      13792
-rwxr-xr-x ln            0     0      11      13692
-rwxr-xr-x ls            0     0      12      17564
-rwxr-xr-x mkdir         0     0      13      13844
-rwxr-xr-x rm            0     0      14      13824
-rwxr-xr-x sh            0     0      15      27524
-rwxr-xr-x stressfs      0     0      16      14412
-rwxr-xr-x usertests     0     0      17      59600
-rwxr-xr-x wc            0     0      18      15012
-rwxr-xr-x zombie        0     0      19      13492
-rwxr-xr-x date          0     0      20      15196
-rwxr-xr-x time          0     55     21      14972
-rwxr-xr-x ps            0     0      22      15960
-rwxr-xr-x chgrp         0     0      23      14012
-rwxr-xr-x chmod         0     0      24      14356
-rwxr-xr-x chown         0     0      25      14012
-rwxr-xr-x p5-test       0     0      26      28348
-rwxr-xr-x testsetuid    0     0      27      13612
crwxr-xr-x console       0     0      28      0
$ 
```

Figure 18: `chgrp` does not change the `time` file when given an invalid GID.

0.4.5 `ls` Command

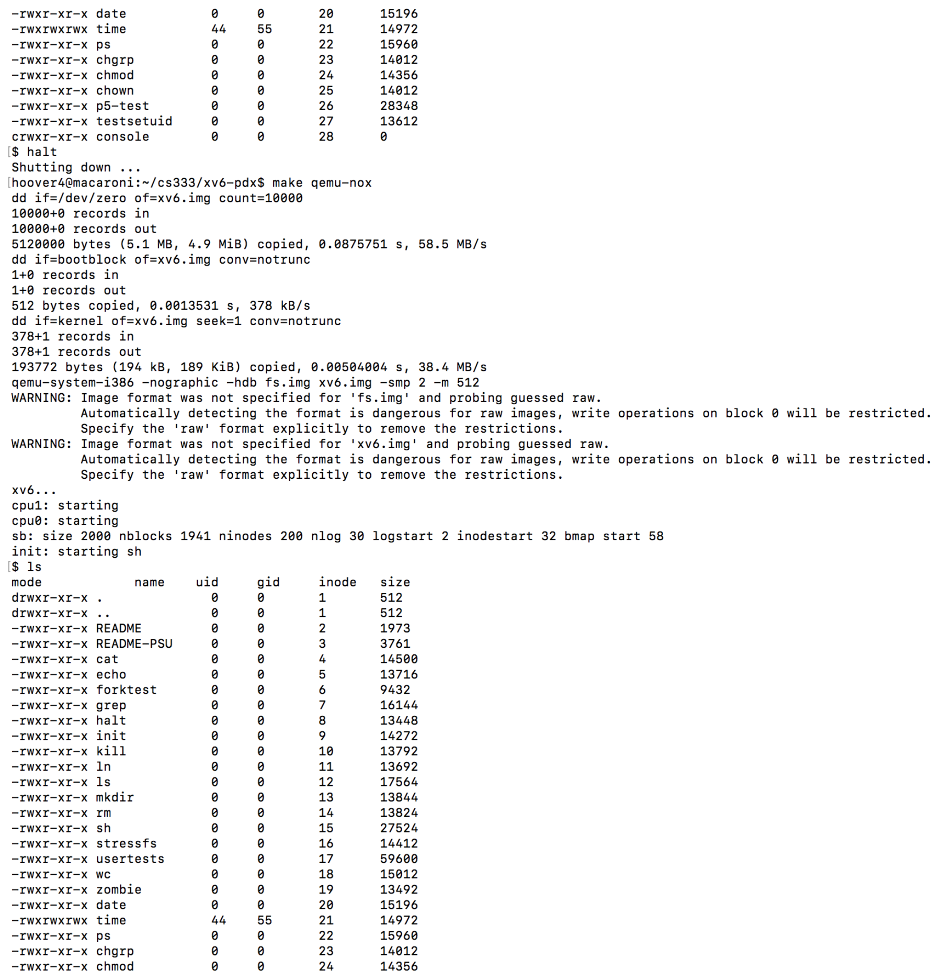
Prints out correct information, including correct permissions for the mode bits
This test is expected to pass, and has been demonstrated numerous time already, as all testing was done using the `ls` command (Figures 8 – 19). The correct `mode` bits are displayed, along with the correct `uid`, `gid`, `inode` number, and `size` in bytes.

Result: Test 5 `PASSES`.

0.4.6 Change Persistence

Changes to UID, GID, and mode persist on exiting and rebooting xv6
This test is expected to pass. To conduct this test, I gave the `time` file `rwx` permissions for user / group / other, a `uid` of 44, and a `gid` of 55; using the `chmod`, `chown`, and `chgrp` commands, respectively. I then entered `halt` into the xv6 command line, and restarted with `make qemu-nox`. The changes to `time` persisted (Figure 19).

Result: Test 6 `PASSES`.



```
-rwxr-xr-x date          0     0      20      15196
-rwxrwxrwx time          44    55     21      14972
-rwxr-xr-x ps            0     0      22      15960
-rwxr-xr-x chgrp         0     0      23      14012
-rwxr-xr-x chmod         0     0      24      14356
-rwxr-xr-x chown         0     0      25      14012
-rwxr-xr-x p5-test       0     0      26      28348
-rwxr-xr-x testsetuid    0     0      27      13612
crwxr-xr-x console       0     0      28      0
[$ halt
Shutting down ...
[hoover4@macaroni:~/cs333/xv6-pdx$ make qemu-nox
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.0875751 s, 58.5 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.0013531 s, 378 kB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
378+1 records in
378+1 records out
193772 bytes (194 kB, 189 KiB) copied, 0.00504004 s, 38.4 MB/s
qemu-system-i386 -nographic -hdb fs.img xv6.img -smp 2 -m 512
WARNING: Image format was not specified for 'fs.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
WARNING: Image format was not specified for 'xv6.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
xv6...
cpu1: starting
cpu0: starting
sb: size 2000 nblocks 1941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
[$ ls
mode            name    uid     gid     inode   size
drwxr-xr-x .             0     0      1       512
drwxr-xr-x ..            0     0      1       512
-rwxr-xr-x README        0     0      2       1973
-rwxr-xr-x README-PSU    0     0      3       3761
-rwxr-xr-x cat           0     0      4       14500
-rwxr-xr-x echo          0     0      5       13716
-rwxr-xr-x forktest      0     0      6       9432
-rwxr-xr-x grep          0     0      7       16144
-rwxr-xr-x halt          0     0      8       13448
-rwxr-xr-x init          0     0      9       14272
-rwxr-xr-x kill          0     0      10      13792
-rwxr-xr-x ln            0     0      11      13692
-rwxr-xr-x ls            0     0      12      17564
-rwxr-xr-x mkdir         0     0      13      13844
-rwxr-xr-x rm            0     0      14      13824
-rwxr-xr-x sh            0     0      15      27524
-rwxr-xr-x stressfs      0     0      16      14412
-rwxr-xr-x usertests     0     0      17      59600
-rwxr-xr-x wc            0     0      18      15012
-rwxr-xr-x zombie        0     0      19      13492
-rwxr-xr-x date          0     0      20      15196
-rwxrwxrwx time          44    55     21      14972
-rwxr-xr-x ps            0     0      22      15960
-rwxr-xr-x chgrp         0     0      23      14012
-rwxr-xr-x chmod         0     0      24      14356
```

Figure 19: Changes to `time` file are shown to persist a shut down and restart.