Description

For this project, I implemented a new abstarction in the form of simple `protection` within the xv6 file system. In particular, for this project I focused on a small subset of proection for file permissions, new system calls and corresponding user commands for manipulating these protections, as well as protection checking within the `exec()` system call. Also, the `ls` command was updated to display information regarding the new file system permissions.

Deliverables

The following features were added to xv6:

- Modifications to the existing `inode`, `dinode`, and `stat` structures to include the new file permissions based on user, group, and other.
- Two new unions, `mode_t` and `stat_mode_t`, which are used to represent the read-write-execute file permisssions.
- new *persistant* file permissions between xv6 sessions.
- New system calls, `chmod()`, `chown()`, `chgrp()`, which modifies the permissions of a file.
- Corresponding user commands, `chmod`, `chown`, and `chgrp`, that take a valid integer range for the command and a file pathname for which permissions will be modified.
- A modification to the existing `ls` command which displays the new file permission information in the form:
  `mode filename uid gid inode size`
- The modified output of `ls` also includes particular values within the mode column. They represent the file types, setuid permission, and user, group, and other read-write-execute permissions.
- Modifications to the existing `exec` command to include permission checking and `setuid` functionality to alter the UID of a process if permitted.

Implementation

inode/dinode/stat modifications

The following files were modified to add the inode/dinode/stat modifications.

- `fs.h`. A new `mode_t` union which contains a bit–map of read-write-execute bits for other, group, and user permissions and a setuid bit (Lines 33 – 53). The `dinode` struct was modified to include new `ushort uid`, `ushort gid`, and `mode_t mode` fields (Lines 63 – 65).
- `file.h`. Three new fields, `ushort uid`, `ushort gid`, and `union mode_t mode`, were added to the existing `inode` struct (Lines 26 – 28).
- `stat.h`. The same mode union was copied and renamed as a `stat_mode_t` to maintain xv6 conventions by avoiding certain compiler directives (Lines 7 – 27). Also the existing `stat` structure was modified to include three new fields, `ushort uid`, `ushort gid`, and `stat_mode_t mode` (Lines 37 – 39).
- `param.h`. A new default mode parameter, `DEFAULTMODE` was defined to be 0755 which does npot have `setuid` set, allows full permissions for users, and read-execute permissions for groups and others (Line 23).
- `fs.c`. In `iupdate()`, the new `uid`, `gid`, and `mode` fields are copied from the in–memory inode to the disk inode (Lines 212 – 214). The `iget()` function was updated to set the `uid`, `gid`, and `mode` fields of the new inode (Lines 236 – 239, 258 – 260). The `ilock()` function was also modified to copy the new fields from the disk inode to the in–memory inode (Lines 304 – 306). For copying information from the inode to a `stat`, `stati()` was modified to copy to the new `stat` fields (Lines 449 – 451).
- `mkfs.c`. The `ialloc()` function was modified set the disk inode `uid`, `gid`, `mode` fields to the default values defined in `param.h` (Lines 234 – 236).

chmod, chown, chgrp System Calls

The following files were modified to include the new `chmod()`, `chown()`, and `chgrp()` system calls:

- `sysfile.c`. System call handlers, `sys_chmod()`, `sys_chown()`, and `sys_chgrp`, were added (Lines 444 – 474). The kernel–side implementations get an integer argument and a string argument off of the stack. If either fails then the system call also fails and returns –1. Otherwise the correct user–side function defined in `fs.c`, which handles additional error checking, is called.
- `fs.c`. The user–side implememtaion happens here (Lines 678 – 738). The user–side implementation uses the `begin_op()` and `end_op()` functions to define transactional semantics for changing information within an inode. The user–side function finds the inode for a given path name using the `namei()` function. If it fails or the `uid`, `gid`, or `mode` fields for the corresponding functions are not within their bounds, then the function fails and return -1. Otherwise, The inode is locked using `ilock()`; the field of the inode is changed; the inode is updated using `iupdate()`. For `chmod()`, the mode parameter is stored in the mode field's `asInt` (Line 689).

chmod, chown, chgrp User Commands

The following files were modified to add the new `chmod`, `chown`, and `chgrp` user commmands:

- `chmod.c`. The `chmod` user command is implemented here. The command error checks, interprets the mode input string as an octal using the newly defined `atog()` function in `ulib.c`, and invokes the `chmod()` system–call.
- `chown.c`. The `chown` user command is implemented here. The command error checks, interprets the UID input string as an integer using the defined `atoi()` function in `ulib.c`, and invokes the `chown()` system–call.
- `chgrp.c`. The `chgrp` user command is implemented here. The command error checks, interprets the UID input string as an integer using the defined `atoi()` function in `ulib.c`, and invokes the `chgrp()` system–call.
- `ulib.c`. The `atog()` function was defined to take an input string and interpret it as some base representation (Lines 102 – 115).

ls Command

Modifications were made to the existing `ls` command within `ls.c`. An pre-existing file, `print_mode.c`, was included which handles the printing of the new `mode fields` in a specified format (Line 6). The `print_mode()` function is then used at the beginning of each printed line for either a file or directory (Lines 61, 87). A new print header function `printheader()` was defined (Lines 8 – 12) to print the new heading and wa then used at teh beginning of the `ls()` routine (Line 54). Also, the print statements for files and directories were modified to print the new `uid` and `gid` fields (Lines 62, 88).

exec Command

The existing `exec()` function in `exec.c` was modified to check execute permissions for a given file. A `struct stat` was declared and then updated using the `stati()` function (Line 36). The permissions are checked in the order user, group, and finally other checking that the process user ID (or group ID) matches the file UID (or GID) and the execute bit is set. If none are true then `exec` fails before reading the `ELF` header (Lines 37 – 44). Lastly, `exec()` checks the `setuid` bit and changes the process UID if it is set (Lines 118 – 119).

Testing

p5−test Commmand Testing

For this test, I used the supplied `p5−test` command which provides automated testing of all the new and modified system calls. This test will be divided into multiple sub–tests. For the first sub–test, I used the first option of testing the Proc UID which tests the s̈etuid() system call added in project 2. I expected the test to show that the `setuid()` system call wass successful. The output for the test can be seen below: From the output above, I saw that `setuid()` was successful.

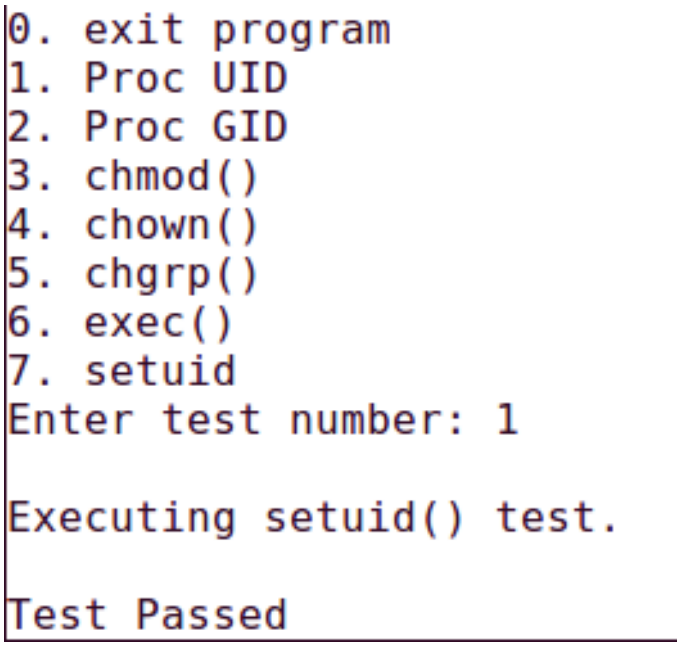

Figure 1: p5–test proc UID test

Thus, this sub–test `PASSES`.

Similarly, for the second sub–test, I tested the existing (as of project 2) system call, `setgid()` using `p5−test`. I again expected the output to indicate that `setgid()` was successful. The output for the test is below:

As shown above, the output showed that `stegid()` was successful.

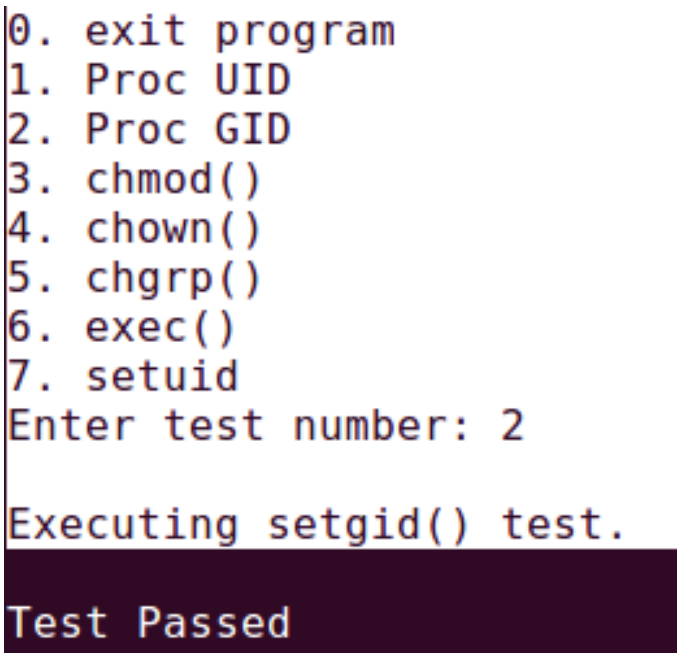

Figure 2: p5–test proc GID test

Therefore, this sub–test `PASSES`.

For the next sub–test, I used the `p5−test` command this time selecting the test chmod option to test the newly added system–call, `chmod()`. I expected the test to to indicate that `chmod()` was successful. Here is the output:
The output above shows that `chmod()` was successful.

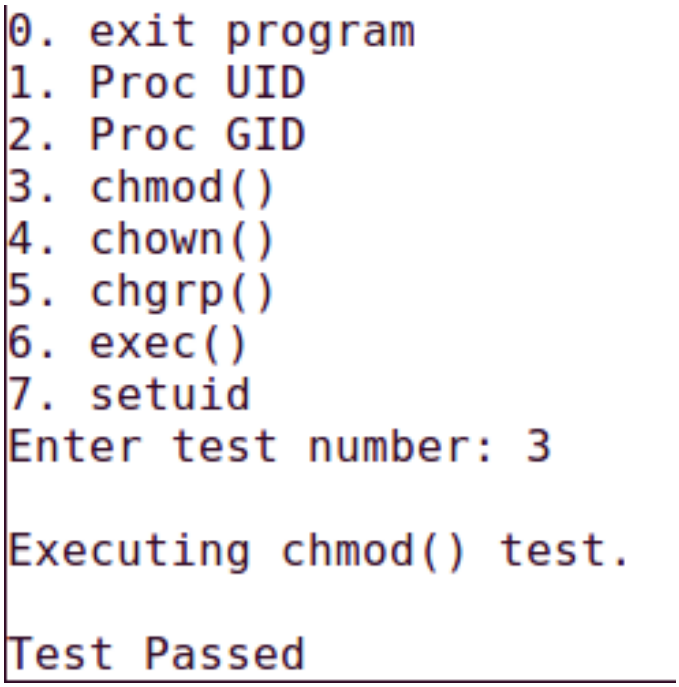

Figure 3: p5–test chmod test

Thus this sub–test `PASSES`.

For the fourth sub–test, I tested the newly created `chown()` system–call again using the `p5−test` command. I expect the automated test to indicate that `chown()` was successful. Here is the output from the text:

The output of the test indicates that `chown()` was successful.

```
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

Therefore, this sub–test `PASSES`.

For the fifth sub–test, I used `p5−test` command to the new system–call `chgrp()`. I expected the test to again indicate that `chgrp()` was usccessful. The output can be seen below:
From the output above, `p5−test` indicated that `chgrp()` was a success.

```
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

Thus, this sub–test `PASSES`.

For the sixth sub–test, I continued to use the supplied test, `p5−test` this time testing the newly added permission checks within the `exec()` system call. `p5−test` tests four cases for `exec()`, three valid options and one invalid option. I expect the three valid options to indicate success and I ecpect the invalid option to fail gracefully. The output for the test follows:
The above output shows three cases with successful execution and the fourth cases failed as expected.

```
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

Thus, this sub–test also `PASSES`.

For the seventh and final sub–test, I used `p5−test` to test the new `setuid` functionality of the `exec()` system–call. The test checks that with correct permissions, a file with the `setuid` bit set, the permission of the process is changed to execute the given file. I exepected the the test to fail when expected and otherwise succeed. Here is the output for the test:
As seen in the output above, all automated tests were successful.

```
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

Thus this sub–test `PASSES`.
Since all sub–tests passed, this test `PASSES`.

chmod Command Testing

This test will be broken into three sub–tests. For the first sub–test, I will use the `ls` command to show that the mode changes when given valid parameters for the `chmod` command. I expected the mode string of `echo` to change to `---x--x--x`. Here is the output for the test:

In the output above, I saw that the mode of `echo` changed after the user command `chmod 0111 echo`.

```
$ chmod 0111 echo
$ 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       3682
-rwxr-xr-x      cat             0       0       4       14548
---x--x--x      echo            0       0       5       13764
-rwxr-xr-x      forktest        0       0       6       9472
-rwxr-xr-x      grep            0       0       7       16184
-rwxr-xr-x      halt            0       0       8       13496
-rwxr-xr-x      init            0       0       9       14320
-rwxr-xr-x      kill            0       0       10      13840
-rwxr-xr-x      ln              0       0       11      13732
-rwxr-xr-x      ls              0       0       12      17788
-rwxr-xr-x      mkdir           0       0       13      13884
-rwxr-xr-x      rm              0       0       14      13872
-rwxr-xr-x      sh              0       0       15      27572
-rwxr-xr-x      stressfs        0       0       16      14460
-rwxr-xr-x      usertests       0       0       17      59648
-rwxr-xr-x      wc              0       0       18      15060
-rwxr-xr-x      zombie          0       0       19      13540
-rwxr-xr-x      date            0       0       20      15028
-rwxr-xr-x      time            0       0       21      15696
-rwxr-xr-x      ps              0       0       22      17220
-rwxr-xr-x      chgrp           0       0       23      13804
-rwxr-xr-x      chmod           0       0       24      13804
-rwxr-xr-x      chown           0       0       25      13804
-rwxr-xr-x      p5-test         0       0       26      28396
-rwxr-xr-x      testsetuid      0       0       27      13660
c---------      console         0       0       28      0
```

Figure 8: Valid chmod test

Thus this sub–test `PASSES`.

For the second sub–test, I tested that when passed an invalid mode parameter, `chmod` failed and did not change the mode (as verified by `ls`). The output can be seen below:
From the output, I saw that `chmod` failed and the mode string of `echo` was unchnaged.

```
$ chmod 2223 echo
Error: chmod failed.
$ 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       3682
-rwxr-xr-x      cat             0       0       4       14548
-rwxr-xr-x      echo            0       0       5       13764
-rwxr-xr-x      forktest        0       0       6       9472
-rwxr-xr-x      grep            0       0       7       16184
-rwxr-xr-x      halt            0       0       8       13496
-rwxr-xr-x      init            0       0       9       14320
-rwxr-xr-x      kill            0       0       10      13840
-rwxr-xr-x      ln              0       0       11      13732
-rwxr-xr-x      ls              0       0       12      17788
-rwxr-xr-x      mkdir           0       0       13      13884
-rwxr-xr-x      rm              0       0       14      13872
-rwxr-xr-x      sh              0       0       15      27572
-rwxr-xr-x      stressfs        0       0       16      14460
-rwxr-xr-x      usertests       0       0       17      59648
-rwxr-xr-x      wc              0       0       18      15060
-rwxr-xr-x      zombie          0       0       19      13540
-rwxr-xr-x      date            0       0       20      15028
-rwxr-xr-x      time            0       0       21      15696
-rwxr-xr-x      ps              0       0       22      17220
-rwxr-xr-x      chgrp           0       0       23      13804
-rwxr-xr-x      chmod           0       0       24      13804
-rwxr-xr-x      chown           0       0       25      13804
-rwxr-xr-x      p5-test         0       0       26      28396
-rwxr-xr-x      testsetuid      0       0       27      13660
c---------      console         0       0       28      0
```

Figure 9: chmod invalid mode test

This sub–test `PASSES`.

For the final sub–test, I verified that `chmod` failed for invalid filename parameters and that no modes were change (again verified by `ls`). The output can be seen below:
In the output above, I saw that `chmod` failed and no mode values were changed.

```
$ chmod 1111 uh
Error: chmod failed.
```

Figure 10: Invalid filename chmod test

This sub–test also `PASSES`.
Since all sub–tests passed, this test `PASSES`.

chown Command Testing

This test will be broken into three sub–tests. For the first sub–test, I will use the `ls` command to show that the UID changes when given valid parameters to the `chown` command. I expected the UID of `echo` to change to 42. Here is the output for the test:

In the output above, I saw that the UID of `echo` changed after the user command `chown 42 echo`.

```
$ chown 42 echo
$ 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       3682
-rwxr-xr-x      cat             0       0       4       14548
-rwxr-xr-x      echo            42      0       5       13764
-rwxr-xr-x      forktest        0       0       6       9472
-rwxr-xr-x      grep            0       0       7       16184
-rwxr-xr-x      halt            0       0       8       13496
-rwxr-xr-x      init            0       0       9       14320
-rwxr-xr-x      kill            0       0       10      13840
-rwxr-xr-x      ln              0       0       11      13732
-rwxr-xr-x      ls              0       0       12      17788
-rwxr-xr-x      mkdir           0       0       13      13884
-rwxr-xr-x      rm              0       0       14      13872
-rwxr-xr-x      sh              0       0       15      27572
-rwxr-xr-x      stressfs        0       0       16      14460
-rwxr-xr-x      usertests       0       0       17      59648
-rwxr-xr-x      wc              0       0       18      15060
-rwxr-xr-x      zombie          0       0       19      13540
-rwxr-xr-x      date            0       0       20      15028
-rwxr-xr-x      time            0       0       21      15696
-rwxr-xr-x      ps              0       0       22      17220
-rwxr-xr-x      chgrp           0       0       23      13804
-rwxr-xr-x      chmod           0       0       24      13804
-rwxr-xr-x      chown           0       0       25      13804
-rwxr-xr-x      p5-test         0       0       26      28396
-rwxr-xr-x      testsetuid      0       0       27      13660
c---------      console         0       0       28      0
```

Figure 11: Valid chown test

Thus this sub–test `PASSES`.

For the second sub–test, I tested that when passed an invalid UID parameter, `chown` failed and did not change the UID (as verified by `ls`). The output can be seen below:
From the output, I saw that `chown` failed and the UID of `echo` was unchnaged.

```
$ chown 333333333 echo
Error: chown failed.
$ 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       3682
-rwxr-xr-x      cat             0       0       4       14548
-rwxr-xr-x      echo            42      0       5       13764
-rwxr-xr-x      forktest        0       0       6       9472
-rwxr-xr-x      grep            0       0       7       16184
-rwxr-xr-x      halt            0       0       8       13496
-rwxr-xr-x      init            0       0       9       14320
-rwxr-xr-x      kill            0       0       10      13840
-rwxr-xr-x      ln              0       0       11      13732
-rwxr-xr-x      ls              0       0       12      17788
-rwxr-xr-x      mkdir           0       0       13      13884
-rwxr-xr-x      rm              0       0       14      13872
-rwxr-xr-x      sh              0       0       15      27572
-rwxr-xr-x      stressfs        0       0       16      14460
-rwxr-xr-x      usertests       0       0       17      59648
-rwxr-xr-x      wc              0       0       18      15060
-rwxr-xr-x      zombie          0       0       19      13540
-rwxr-xr-x      date            0       0       20      15028
-rwxr-xr-x      time            0       0       21      15696
-rwxr-xr-x      ps              0       0       22      17220
-rwxr-xr-x      chgrp           0       0       23      13804
-rwxr-xr-x      chmod           0       0       24      13804
-rwxr-xr-x      chown           0       0       25      13804
-rwxr-xr-x      p5-test         0       0       26      28396
-rwxr-xr-x      testsetuid      0       0       27      13660
c---------      console         0       0       28      0
```

Figure 12: Invalid UID chown test

This sub–test `PASSES`.

For the final sub–test, I verified that `chown` failed for invalid filename parameter and that no UIDs were change (again verified by `ls`). The output can be seen below:
In the output above, I saw that `chown` failed and no UID values were changed.

```
$ chown 42 uh
Error: chown failed.
$ 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       3682
-rwxr-xr-x      cat             0       0       4       14548
-rwxr-xr-x      echo            42      0       5       13764
-rwxr-xr-x      forktest        0       0       6       9472
-rwxr-xr-x      grep            0       0       7       16184
-rwxr-xr-x      halt            0       0       8       13496
-rwxr-xr-x      init            0       0       9       14320
-rwxr-xr-x      kill            0       0       10      13840
-rwxr-xr-x      ln              0       0       11      13732
-rwxr-xr-x      ls              0       0       12      17788
-rwxr-xr-x      mkdir           0       0       13      13884
-rwxr-xr-x      rm              0       0       14      13872
-rwxr-xr-x      sh              0       0       15      27572
-rwxr-xr-x      stressfs        0       0       16      14460
-rwxr-xr-x      usertests       0       0       17      59648
-rwxr-xr-x      wc              0       0       18      15060
-rwxr-xr-x      zombie          0       0       19      13540
-rwxr-xr-x      date            0       0       20      15028
-rwxr-xr-x      time            0       0       21      15696
-rwxr-xr-x      ps              0       0       22      17220
-rwxr-xr-x      chgrp           0       0       23      13804
-rwxr-xr-x      chmod           0       0       24      13804
-rwxr-xr-x      chown           0       0       25      13804
-rwxr-xr-x      p5-test         0       0       26      28396
-rwxr-xr-x      testsetuid      0       0       27      13660
c---------      console         0       0       28      0
```

Figure 13: Invalid filename chown test

This sub–test also `PASSES`.
Since all sub–tests passed, this test `PASSES`.

chgrp Command Testng

This test will be broken into three sub–tests. For the first sub–test, I will use the `ls` command to show that the UID changes when given valid parameters to the `chgrp` command. I expected the GID of `echo` to change to 42. Here is the output for the test:

In the output above, I saw that the GID of `echo` changed to 42 after the user command `chgrp 42 echo`.

```
$ chgrp 42 echo
$ 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       3682
-rwxr-xr-x      cat             0       0       4       14548
-rwxr-xr-x      echo            42      42      5       13764
-rwxr-xr-x      forktest        0       0       6       9472
-rwxr-xr-x      grep            0       0       7       16184
-rwxr-xr-x      halt            0       0       8       13496
-rwxr-xr-x      init            0       0       9       14320
-rwxr-xr-x      kill            0       0       10      13840
-rwxr-xr-x      ln              0       0       11      13732
-rwxr-xr-x      ls              0       0       12      17788
-rwxr-xr-x      mkdir           0       0       13      13884
-rwxr-xr-x      rm              0       0       14      13872
-rwxr-xr-x      sh              0       0       15      27572
-rwxr-xr-x      stressfs        0       0       16      14460
-rwxr-xr-x      usertests       0       0       17      59648
-rwxr-xr-x      wc              0       0       18      15060
-rwxr-xr-x      zombie          0       0       19      13540
-rwxr-xr-x      date            0       0       20      15028
-rwxr-xr-x      time            0       0       21      15696
-rwxr-xr-x      ps              0       0       22      17220
-rwxr-xr-x      chgrp           0       0       23      13804
-rwxr-xr-x      chmod           0       0       24      13804
-rwxr-xr-x      chown           0       0       25      13804
-rwxr-xr-x      p5-test         0       0       26      28396
-rwxr-xr-x      testsetuid      0       0       27      13660
c---------      console         0       0       28      0
```

Figure 14: Valid chgrp test

Thus this sub–test `PASSES`.

For the second sub–test, I tested that when passed an invalid GID parameter, `chgrp` failed and did not change the GID (as verified by `ls`). The output can be seen below:
From the output, I saw that `chgrp` failed and the GID of `echo` was unchnaged.

```
$ chgrp 33333333 echo
Error: chgrp failed.
$ 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       3682
-rwxr-xr-x      cat             0       0       4       14548
-rwxr-xr-x      echo            42      42      5       13764
-rwxr-xr-x      forktest        0       0       6       9472
-rwxr-xr-x      grep            0       0       7       16184
-rwxr-xr-x      halt            0       0       8       13496
-rwxr-xr-x      init            0       0       9       14320
-rwxr-xr-x      kill            0       0       10      13840
-rwxr-xr-x      ln              0       0       11      13732
-rwxr-xr-x      ls              0       0       12      17788
-rwxr-xr-x      mkdir           0       0       13      13884
-rwxr-xr-x      rm              0       0       14      13872
-rwxr-xr-x      sh              0       0       15      27572
-rwxr-xr-x      stressfs        0       0       16      14460
-rwxr-xr-x      usertests       0       0       17      59648
-rwxr-xr-x      wc              0       0       18      15060
-rwxr-xr-x      zombie          0       0       19      13540
-rwxr-xr-x      date            0       0       20      15028
-rwxr-xr-x      time            0       0       21      15696
-rwxr-xr-x      ps              0       0       22      17220
-rwxr-xr-x      chgrp           0       0       23      13804
-rwxr-xr-x      chmod           0       0       24      13804
-rwxr-xr-x      chown           0       0       25      13804
-rwxr-xr-x      p5-test         0       0       26      28396
-rwxr-xr-x      testsetuid      0       0       27      13660
c---------      console         0       0       28      0
```

Figure 15: Invalid GID chgrp test

This sub–test `PASSES`.

For the final sub–test, I verified that `chgrp` failed for an invalid filename parameter and that no GIDs were change (again verified by `ls`). The output can be seen below:
In the output above, I saw that `chgrp` failed and no GID values were changed.

```
$ chgrp 43 uh
Error: chgrp failed.
$ 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       3682
-rwxr-xr-x      cat             0       0       4       14548
-rwxr-xr-x      echo            42      42      5       13764
-rwxr-xr-x      forktest        0       0       6       9472
-rwxr-xr-x      grep            0       0       7       16184
-rwxr-xr-x      halt            0       0       8       13496
-rwxr-xr-x      init            0       0       9       14320
-rwxr-xr-x      kill            0       0       10      13840
-rwxr-xr-x      ln              0       0       11      13732
-rwxr-xr-x      ls              0       0       12      17788
-rwxr-xr-x      mkdir           0       0       13      13884
-rwxr-xr-x      rm              0       0       14      13872
-rwxr-xr-x      sh              0       0       15      27572
-rwxr-xr-x      stressfs        0       0       16      14460
-rwxr-xr-x      usertests       0       0       17      59648
-rwxr-xr-x      wc              0       0       18      15060
-rwxr-xr-x      zombie          0       0       19      13540
-rwxr-xr-x      date            0       0       20      15028
-rwxr-xr-x      time            0       0       21      15696
-rwxr-xr-x      ps              0       0       22      17220
-rwxr-xr-x      chgrp           0       0       23      13804
-rwxr-xr-x      chmod           0       0       24      13804
-rwxr-xr-x      chown           0       0       25      13804
-rwxr-xr-x      p5-test         0       0       26      28396
-rwxr-xr-x      testsetuid      0       0       27      13660
c---------      console         0       0       28      0
```

Figure 16: Invalid chgrp test

This sub–test also `PASSES`.
Since all sub–tests passed, this test `PASSES`.

ls Command Testing

For additional testing of the `ls` command, I tested that the correct information for the mode is printed out by calling `ls` after booting up xv6. I expected two directories, denoted by a d at the start of the mode, once console denoted by a c at the start of the mode, and the rest as files (a starting dash). I expected the rest to be the default mode: `rwxr-xr-x`. The output for the test follows:
In the output above, I saw the expected, correct modes.

```
$ 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       3682
-rwxr-xr-x      cat             0       0       4       14548
-rwxr-xr-x      echo            0       0       5       13764
-rwxr-xr-x      forktest        0       0       6       9472
-rwxr-xr-x      grep            0       0       7       16184
-rwxr-xr-x      halt            0       0       8       13496
-rwxr-xr-x      init            0       0       9       14326
-rwxr-xr-x      kill            0       0       10      13840
-rwxr-xr-x      ln              0       0       11      13732
-rwxr-xr-x      ls              0       0       12      17788
-rwxr-xr-x      mkdir           0       0       13      13884
-rwxr-xr-x      rm              0       0       14      13872
-rwxr-xr-x      sh              0       0       15      27572
-rwxr-xr-x      stressfs        0       0       16      14460
-rwxr-xr-x      usertests       0       0       17      59648
-rwxr-xr-x      wc              0       0       18      15060
-rwxr-xr-x      zombie          0       0       19      13540
-rwxr-xr-x      date            0       0       20      15028
-rwxr-xr-x      time            0       0       21      15696
-rwxr-xr-x      ps              0       0       22      17220
-rwxr-xr-x      chgrp           0       0       23      13804
-rwxr-xr-x      chmod           0       0       24      13804
-rwxr-xr-x      chown           0       0       25      13804
-rwxr-xr-x      p5-test         0       0       26      28396
-rwxr-xr-x      testsetuid      0       0       27      13660
crwxr-xr-x      console         0       0       28      0
```

Figure 17: Modified ls test

Therefore, this test `PASSES`.

Persistance Test

For this test, I tested that changes to the UID, GID, and mode, as shown by the modified `ls` command, presist upon exiting and rebooting (using `make qemu-nox` without a `make clean`. I expected the post–reboot `ls` display to match the `ls` display prior to rebooting. The output can be seen below:
In the output above, The `ls` command's display after rebooting is the same as before the reboot.

```
xv6...
cpu1: starting
cpu0: starting
sb: size 2000 nblocks 1941 ninodes 200 nlog 30 logstart 2 inodes
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       3682
-rwxr-xr-x      cat             0       0       4       14548
---x--x--x      echo            0       0       5       13764
-rwxr-xr-x      forktest        0       0       6       9472
-rwxr-xr-x      grep            0       0       7       16184
-rwxr-xr-x      halt            0       0       8       13496
-rwxr-xr-x      init            0       0       9       14320
-rwxr-xr-x      kill            0       0       10      13840
-rwxr-xr-x      ln              0       0       11      13732
-rwxr-xr-x      ls              0       0       12      17788
-rwxr-xr-x      mkdir           0       0       13      13884
-rwxr-xr-x      rm              0       0       14      13872
-rwxr-xr-x      sh              0       0       15      27572
-rwxr-xr-x      stressfs        0       0       16      14460
-rwxr-xr-x      usertests       0       0       17      59648
-rwxr-xr-x      wc              0       0       18      15060
-rwxr-xr-x      zombie          0       0       19      13540
-rwxr-xr-x      date            0       0       20      15028
-rwxr-xr-x      time            0       0       21      15696
-rwxr-xr-x      ps              0       0       22      17220
-rwxr-xr-x      chgrp           0       0       23      13804
-rwxr-xr-x      chmod           0       0       24      13804
-rwxr-xr-x      chown           0       0       25      13804
-rwxr-xr-x      p5-test         0       0       26      28396
-rwxr-xr-x      testsetuid      0       0       27      13660
c---------      console         0       0       28      0
```

Figure 18: Modification Persistance Test

Thus this test `PASSES`.