Project Five Report
Introduction to Operating Systems
Spring 2017

Andy Keene

Description

For this assignment I learned about the file system abstraction in xv6; its underlying meta information; transactional operations in a log based file system; and implemented a new simple file protection feature. This feature is conditionally compilable in its entirety and thus can be turned on or off.

Deliverables

The following features were added to xv6:

- Each file now contains the associated meta information for its owner (`uid`), group (`gid`), and permissions (`mode`); the mode represents what read, write, and execute permissions the owner, group members, and others have along with a `setuid` bit which indicates whether the process executing the file must inherit the files UID.

  Thus, the `inode`, `dinode`, and `stat` structures were modified to contain this information while the kernel side functions `ilock()`, `iupdate()`, `stati()` and `create()` were modified to copy, return, and set this information as appropriate.

- The program `mkfs` was modified to set all files owner, group, and mode to the default values upon creation of the xv6 file system. The default values are 0 for the owner and group and 0755 for the mode.

- New system calls to support file ownership and permissions:

  – chmod sets the given `mode`, or permission, bits for the target specified by `pathname`.

  ```
  int
  chmod(char *pathname, int mode);
  ```

  – chown sets the UID for the file target specified by `pathname` to the given `owner`.

  ```
  int
  chown(char *pathname, int owner);
  ```

  – chgrp sets the GID for the file target specified by `pathname` to the given `group`.

  ```
  int
  chgrp(char *pathname, int group);
  ```

- Existing system calls were modified to both support and enforce file ownership and permissions. The `exec()` system call will first no longer read the file unless the calling process has execute permission for the file. Secondly, when the new image is omitted if the files setuid mode bit is set, the executing process will inherit the files UID.

- New user commands were added to give the user the ability to update a files UID, GID, and mode:

  – `chmod` sets the given mode, or permission, bits for file mode; where the file is specified by the `target` pathname.
  Usage: `chmod MODE TARGET`

  – `chown` sets the given owner as the file UID for the specified target; where target is a file path.
  Usage: `chmod OWNER TARGET`

  – `chgrp` sets the given group as the file GID for the specified target; where target is a file path.
  Usage: `chmod GROUP TARGET`

- The existing user command `ls` was modified to now display, in total the:

  `mode`. The first column 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`. 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.

  The new output displays as:

```
$ 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  cat         0       0       3       14884
-rwxr-xr-x  echo        0       0       4       13849
-rwxr-xr-x  forktest    0       0       5       9361
-rwxr-xr-x  grep        0       0       6       16812
-rwxr-xr-x  init        0       0       7       14750
-rwxr-xr-x  kill        0       0       8       13981
-rwxr-xr-x  ln          0       0       9       13879
-rwxr-xr-x  ls          0       0       10      19010
-rwxr-xr-x  mkdir       0       0       11      14010
-rwxr-xr-x  rm          0       0       12      13991
-rwxr-xr-x  sh          0       0       13      26855
-rwxr-xr-x  stressfs    0       0       14      14969
-rwxr-xr-x  usertests   0       0       15      69424
-rwxr-xr-x  wc          0       0       16      15470
-rwxr-xr-x  zombie      0       0       17      13615
-rwxr-xr-x  halt        0       0       18      13441
-rwxr-xr-x  MMchown     0       0       19      14464
-rwxr-xr-x  MMchgrp     0       0       20      14464
-rwxr-xr-x  MMchmod     0       0       21      14780
crwxr-xr-x  console     0       0       22      0
$ ls halt
mode            name    uid     gid     inode   size
-rwxr-xr-x  halt        0       0       18      13441
$
```

- Conditional compilation. Project 5 uses the `CS333_P5` flag, which may be turned on or turned off to enable the features outlined in this report.

Implementation

UID, GID, and Mode File Meta-information

To support the UID, GID, and Mode meta-information for files the `inode`, `dinode`, and `stat` structures were updated with new fields. A new union definition, `mode_t`, to define the mode to be accessible as either bits or an int was added in fs.h (lines 32-49) as:

```
union mode_t {
  struct {
    uint o_x : 1;
    uint o_w : 1;
    uint o_r : 1;   // other
    uint g_x : 1;
    uint g_w : 1;
    uint g_r : 1;   // group
    uint u_x : 1;
    uint u_w : 1;
    uint u_r : 1;   // user
    uint setuid : 1;
    uint     : 22   // pad
  } flags;
  uint asInt;
};
```

The mode defines what read, write, and execute abilities that the owner, group, and others have for this file while the `setuid` bit indicates whether the executing process must inherit the files UID. The definitions for `inode` and `dinode` share this definition, while due to namespace issues, an identical copy to the definition above but named as `stat_mode_t` was added to stat.h (lines 5-22). Additionally, the fields `ushort uid`, `ushort gid`, and `union mode_t mode` were added to the `dinode` and `inode` structure definitions (fs.h lines 57-61, file.h lines 23-27). Due to the delicate balance a `dinode`s size plays in the construction of the file system the number of `dinodes`, NDIRECT, was modified to be 10 (fs.h lines 23-25) to accommodate the additional space each individual `dinode` now occupies. For example, the new `dinode` structure in total is now defined as (with conditional compilation tags removed):

```
struct dinode {
    short type;             // File type
    short major;            // Major device number (T_DEV only)
    short minor;            // Minor device number (T_DEV only)
    short nlink;            // Number of links to inode in file system
    ushort uid;             // owner ID
    ushort gid;             // group ID
    union mode_t mode;      // protection/mode bits
    uint size;              // Size of file (bytes)
    uint addrs[NDIRECT+1];    // Data block addresses
  };
```

Similarly, the the fields `ushort uid`, `ushort gid`, and `union stat_mode_t mode` were added to the `stat` structure (stat.h lines 30-34).

To support modifications to kernel side functions which must now set or update these fields in the respective structures, default values for the ownership (UID), groups (GID), and mode of a file were added to param.h (lines 14-18). These values for `DEFAULT_UID`, `DEFAULT_GID`, `DEFAULT_MODE` were defined as 0, 0, and 0755 respectively. The default mode, 0755, defines permissions to be: the user can read, write and execute; the group and others can only read and execute; and the setuid bit is *not* set. The inital process now has its UID and GID set using `DEFAULT_UID` and `DEFAULT_GID` values rather than the old definitions which implied the use was exclusively for the initial process (proc.c 496-497).

The following kernel side functions were modified to support the copying of this information between file abstraction structures, and initializing these fields on creation:

- `ilock`. `ilock()` is responsible for setting a lock on the inode reference and updating the inodes information with its corresponding on-disk dinode. As such, `ilock()` now copies the `uid`, `gid`, and `mode` fields from the dinode *to* the inode (fs.c lines 294-299).
- `iupdate`. `iupdate()` is responsible for refreshing the corresponding dinode with changes to the given inode. As such, `ilock()` now copies the `uid`, `gid`, and `mode` fields from the inode *to* the dinode (fs.c lines 211-216).
- `stati`. `stati()` is invoked deep within the `fstat` system call and is responsible for copying inode information to the stat structure. As such, `stati()` now copies the `uid`, `gid`, and `mode` fields from the inode *to* the stat structure (fs.c lines 211-216).
- `create`. `create()` is invoked by system calls responsible for creating a file or directory - the inode structure created now has its `uid` field set to be that of the calling process, the `gid` field to be that of the calling process, and the `mode` field to be the default of 0755 (sysfile.c lines 265-273). It should be noted that the rationale for setting the UID and GID to the calling process is that the calling process is the files owner - and thus should be linked by default in this manor. This establishes basic permission restrictions - i.e. only the *user* or rather owner, may edit the file by default.

mkfs

The program `mkfs` is responsible for building the xv6 file system on boot if it is not already built. When a file is created the `ialloc()` function call within `mkfs` is invoked and sets properties such as `type`, `nlink` (number of links), and `size` for the `dinode` before it is written to disk; here default values for the `uid`, `gid`, and `mode` are now set for the `dinode` (mkfs.c lines 233-338) before writing to disk. Additionally, a project 5 conditional compilation flag (`CS333_P5`) was added to the mkfs make target (Makefile line 133) since this does not use the CFLAGS variable responsible for flagging conditional compilation for the rest of the files. The original target can still be set by swapping line 133 with the original target which is commented out on line 134 of the Makefile.

New System Calls

Using the process outlined in project one, the following system calls were implemented by adding: a user-side header in `user.h`; creating a system call number in `syscall.h`; updating the system call jump table, the system call name table, and adding a kernel side header in `syscall.c`; a user-side stub in `usys.S`; and implementation in `sysfile.c` since the system call is responsible for handling file information.

Each of the following system calls is responsible for updating a files meta data on disk. Thus, all follow a transactional procedure. The transaction procedure begins by calling `begin_op()` to instantiate a transaction with the file system. The procedure then attempts to aquire the inode for the given *pathname* via a call to the kernel side `namei()` where if the inode is not found, the transaction is ended with a call to `end_op()` and -1 is returned. Upon successfully retrieving the inode for the *pathname* it is locked for the purpose of atomicity using `ilock()`. The appropriate inode meta information is then updated before `iupdate()` is called to sync the changes to the inode with the disk. Lastly, the transaction is ended with a call to `end_op()` and 0 is returned to signal a successful update of the files meta information.

Each new system call's description and it's corresponding file changes are as follows:

- The `chmod()` system call sets the mode of the given file *pathname*, to the given *mode*. The given *mode* is retrieved from the stack as an int using `argint()` while the targets *pathname* is retrieved using `argptr()`; if `chmod()` fails to retrieve either of the arguments, or if the successfully retrieved *mode* is greater than the octal representation 0755 or less than 0, then -1 is returned to signal an error. Upon successfully retrieving the arguments from the stack and validating the `mode`s range, `chmod()` follows the transactional procedure outlined above where the file meta information updated is the inodes `mode`.

  The files modified to support this system call are as follows:

  - user.h (line 42)
  - usys.S (line 41)
  - syscall.h (line 34)
  - syscall.c (lines 111, 148, 188)
  - sysfile.c (lines 456-486)

  The function prototype is:

  ```
  int
  chmod(char *pathname, int mode);
  ```

- The `chown()` system call sets the UID of the given file *pathname*, to the given *owner*. The given *owner* is retrieved from the stack as an int using `argint()` while the targets *pathname* is retrieved using `argptr()`; if `chown()` fails to retrieve either of the arguments, or if the successfully retrieved *owner* is greater than the maximum UID (32767) or less than 0, then -1 is returned to signal an error. Upon successfully retrieving the arguments from the stack and validating the `owner`s range, `chown()` follows the transactional procedure outlined above where the file meta information updated is the inodes `uid`.

  The files modified to support this system call are as follows:

  - user.h (line 43)
  - usys.S (line 42)
  - syscall.h (line 35)
  - syscall.c (lines 112, 149, 189)
  - sysfile.c (lines 488-519)

  The function prototype is:

  ```
  int
  chown(char *pathname, int owner);
  ```

- The `grp()` system call sets the UID of the given file *pathname*, to the given *group*. The given *group* is retrieved from the stack as an int using `argint()` while the targets *pathname* is retrieved using `argptr()`; if `chgrp()` fails to retrieve either of the arguments, or if the successfully retrieved *group* is greater than the maximum GID (32767) or less than 0, then -1 is returned to signal an error. Upon successfully retrieving the arguments from the stack and validating the `group`s range, `chgrp()` follows the transactional procedure outlined above where the file meta information updated is the inodes `gid`.

  The files modified to support this system call are as follows:

  - user.h (line 44)
  - usys.S (line 43)
  - syscall.h (line 36)
  - syscall.c (lines 113, 150, 190)
  - sysfile.c (lines 521-552)

  The function prototype is:

```
int
chgrp(char *pathname, int group);
```

Modified system calls - exec()

The system call `exec()` was modified to enforce the permissions set by the mode of a file when executing a file. To support this the headers for fs.h and file.h (in order) were included to give `exec()` the ability to see and use the inode structures relevant fields (lines 12-13). `exec()` now uses the permissions associated with the group the process belongs to (exec.c lines 46-59); that is in-order, if the calling process and the file have the same UID, the modes *user* permissions are used; if the calling process and the file have the same GID, the modes *group* permissions are used; if the calling process and the file have neither the same UID nor GID then the modes *other* permissions are used.

If the permission group to which the calling process belongs *does not* have executable permissions (i.e. the `x` bit is not set) then the file is not read and `exec()` jumps to the `bad` routine which cleans up the opened resources and returns -1. However, if the process has executable permissions, as outlined, then the `setuid` of the file is checked (lines 63-64) where if this mode bit is set then the file UID is saved and later used to set the calling process UID immediately before execution of the file begins (lines 129-130). Note that `setuid` is initialized as -1 since this is an illegal UID and thus indicates, if unchanged, that the process UID *should not* be set before execution.

New User Commands

All user commands wrapped the operating code in conditional compilation tags, so that if the project 5 flag is not set but the user programs are not removed from the Makefile, their execution will do nothing.

- The `chmod` user command (chmod.c) first verifies it is given both a *mode* and filepath *target* argument, and that the *mode* is exactly four digits (lines 12-20). If these conditions are not met a usage message is printed to standard out and the program exits. The string representing the *mode* is then parsed using a routing similar to the user library function `atoi()` (lines 25-33), where all octal characters are converted to an integer and if the character is not valid a usage message is printed to the screen and the program exits. Consequently, an integer representing the octal character string is created *iff* all characters in the string are valid octal numbers (0-7). Note that although this does not necessarily ensure a correct mode the system call `chmod()` validates the mode independently. The integer representing the octal given *mode* and the *target* are then passed to the `chmod()` system call; where if this call returns an error a message is printed to standard out (35-36). `chmod` was added to the Makefile user programs (line 163).

  Usage: `chmod MODE TARGET`

- The `chown` user command (chown.c) validates it is given both an *owner* and filepath *target* (lines 11-14). It then passes the given *owner*, converted to an integer using the user side function `atoi()`, and the *target* argument to the `chown()` system call (lines 18-19). If the call fails, a message is printed to standard out before the program exits (lines 18-19). `chown` was added to the Makefile user programs (line 164).

Usage: `chown OWNER TARGET`

- The `chgrp` user command (chgrp.c) validates it is given both a *group* and filepath *target* target(lines 11-14). It then passes the given *group*, converted to an integer using the user side function `atoi()`, and the *target* argument to the `chgrp()` system call (lines 18-19). If the call fails, a message is printed to standard out before the program exits (lines 19-21). `chgrp` was added to the Makefile user programs (line 165).

  Usage: `chgrp GROUP TARGET`

ls Modification

Due to the modifications in the stat.h and the `stati()` system call described in the *UID, GID, and Mode File Meta-information* section, the `fstat()` system call returns a stat structure that contains the files `uid`, `gid`, and `mode`. As such, the only modifications necessary to update the display information in `ls` was the printing routine. The print_mode.c routine that was provided is now included in ls.c (line 6); this routine is responsible for displaying the mode in the form of `-rwx-r-xr-x` where: 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'; and the right nine spots indicate the user, group and other permissions (i.e. `-uuugggooo`). Line 49 is now responsible for printing a header to identify the respective columns, where lines 55-56 and lines 80-81 now print the new information for files and directories. As discussed in the *Deliverables* section, the information for each file and directory that is printed, in total includes the mode, name, uid, gid, inode number, and size. The new output displays as:

```
$ 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  cat           0     0       3       14884
-rwxr-xr-x  echo          0     0       4       13849
-rwxr-xr-x  forktest      0     0       5       9361
-rwxr-xr-x  grep          0     0       6       16812
-rwxr-xr-x  init          0     0       7       14750
-rwxr-xr-x  kill          0     0       8       13981
-rwxr-xr-x  ln            0     0       9       13879
-rwxr-xr-x  ls            0     0       10      19010
-rwxr-xr-x  mkdir         0     0       11      14010
-rwxr-xr-x  rm            0     0       12      13991
-rwxr-xr-x  sh            0     0       13      26855
-rwxr-xr-x  stressfs      0     0       14      14969
-rwxr-xr-x  usertests     0     0       15      69424
-rwxr-xr-x  wc            0     0       16      15470
-rwxr-xr-x  zombie        0     0       17      13615
-rwxr-xr-x  halt          0     0       18      13441
-rwxr-xr-x  MMchown       0     0       19      14464
-rwxr-xr-x  MMchgrp       0     0       20      14464
-rwxr-xr-x  MMchmod       0     0       21      14780
crwxr-xr-x  console       0     0       22      0
$ ls halt
mode            name    uid     gid     inode   size
-rwxr-xr-x  halt          0     0       18      13441
```

Testing

System calls

All system calls - `setuid()`, `setgid()`, `chmod()`, `chgrp()`, `chown()`, `exec()` - were tested using the provided automated test suite `p5-test`. The following subsections, one for each system call, will describe: what is to be tested; how the `p5-test` harness runs the test; what the expected results are; what the results are; and if the results demonstrate the correct behavior for the system call. The main loop (lines 344-355) prints a menu with options 0-7 invoking the corresponding test function by calling `doTest()` which runs the given function name with the given argument; this is either the empty string or the name of the file `setsetuid` which will simply print the UID of its process. Unless noted otherwise, a secondary helper function `check()` is used to run all system calls with arbitrary arguments and both prints an error message and aborts when the command invoked (i.e. `exec()`) returns an error code. Thus anytime `check()`, or a system call, is called in subsequent tests, we know the program will inform us and abort when an error code is returned. Since some of the tests within `p5-test` depend on default permissions for the file each test will only be run immediately after a make and boot of xv6.

0.0.1 setuid()

This test will demonstrate the correct behavior of the `setuid()` system call. To do this, we will run `p5-test` immediately after the make and boot of xv6 and press 1 to trigger the function `doUidTest()` with a null argument (described previously in *0.0 System Calls*).

`doUidTest()` will first get and save the UID of its process. Next it will increment this value by 1 and set this as its UID through the `setuid()` system call where if the return code signals an error occurred the test quite will print a report and abort. Note that since no UIDs will have been changed previous to the test running and all processes inherit their UID from their parent with the exception of `initproc()` whose UID is set to the default value 0, we know the increment by 1 will be a valid UID. `doUidTest()` will then verify this UID was set correctly by calling `getuid()` and comparing the returned value against what it set the UID to be; here if a match is not found `p5-test` will error and abort. Lastly, for the list of invalid UIDs - 32767+5, -41, $\tilde{0}$ - the test will validate an error code is returned by `setuid()` if any are used as an argument - any successful return code will cause `p5-test` to error and abort. Since `doUidTest()` errors and aborts at any incorrect behavior and demonstrates both valid and invalid UIDs are set or handled correctly, we expect if `setuid()` is correct that `doUidTest()` will complete and prints that the test was successful.

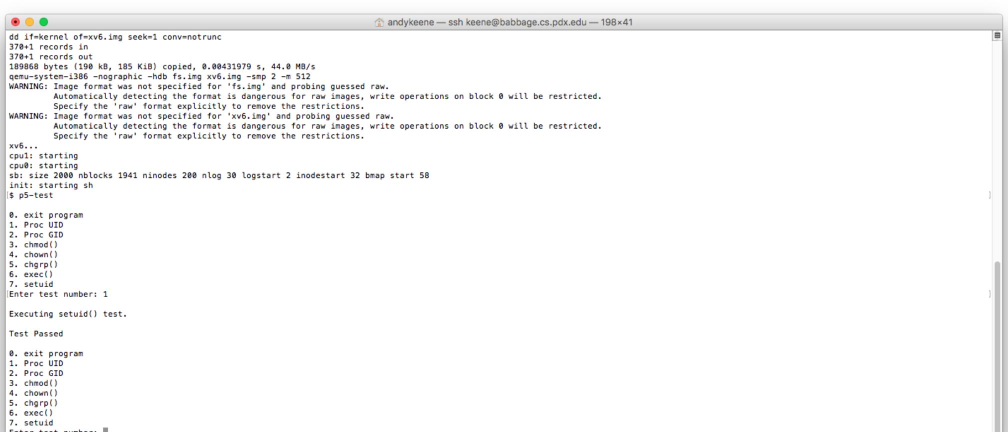


Figure 1: p5-test option 1: doUidTest()

Here we see that upon booting xv6 that `p5-test` was immediately run with option 1 invoking the `doUidTest()`. Since the completion of the test, by virtue of the code outlined above, demonstrates the correct behavior of `setuid()` we can conclude that this sub-test **PASSES**.

0.0.2 setgid()

This test will demonstrate the correct behavior of the `setgid()` system call. To do this, we will run `p5-test` immediately after the make and boot of xv6 and press 2 to trigger the function `doGidTest()` with a null argument (described previously in *0.0 System Calls*).

`doGidTest()` will first get and save the GID of its process. Next it will increment this value by 1 and set this as its GID through the `setgid()` system call where if the return code signals an error occurred the test quite will print a report and abort. Note that since no GIDs will have been changed previous to the test running and all processes inherit their GID from their parent with the exception of `initproc()` whose GID is set to the default value 0, we know the increment by 1 will yield a valid GID. `doGidTest()` will then verify this GID was set correctly by calling `getgid()` and comparing the returned value against what it set the GID to be; here if a match is not found, another `p5-test` will error and abort. Lastly, for the list of invalid GIDs - 32767+5, -41, $\tilde{0}$ - the test will validate an error code is returned by `setgid()` if any are used as an argument - any successful return code will cause `p5-test` to error and abort. Since `doGidTest()` errors and aborts at any incorrect behavior and demonstrates both valid and invalid UIDs are set or handled correctly, we expect if `setgid()` is correct that `doGidTest()` will complete and prints that the test was successful.

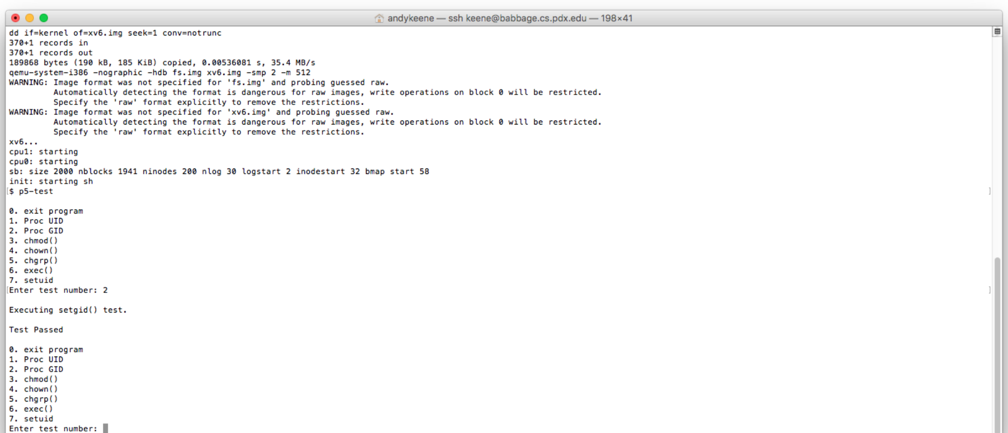


Figure 2: p5-test option 2: doGidTest()

Here we see that upon booting xv6 that `p5-test` was immediately run with option 2 invoking the `doGidTest()`. Since the completion of the test, by virtue of the code outlined above, demonstrates the correct behavior of `setgid()` we can conclude that this sub-test **PASSES**.

0.0.3 chmod()

This test will demonstrate the correct behavior of `chmod()`. To do this, we will run `p5-test` immediately after the boot of xv6 and press 3 to trigger the function `doChmodTest()` with an argument of `testsetuid` (described previously in *0.0 System Calls*).

`doChmodTest()` (lines 164-197) gets file meta information for testsetuid by calling `(stat()` and saves its current mode. Next, for each of the permissions 1544, 1454, 1445, 1666, `doChmodTest()` sets permissions for the file testsetuid and refreshes the file information by calling `stat()` again on the stat structure it holds. If the mode from the file meta information after the refresh matches the original permissions it saved, or if the refreshed mode does not match the mode given to the last invocation of `chmod()` (addition to lines 195-199) then the function will print an error message and exit. After all the permissions listed have been tested `doChmodTest()` will print that the test passed and return the the main menu.

Since the correct behavior of `chmod()` is to set a *valid* mode for the given file if it exists, and we know that the file exists, that each mode is valid, and that the file will have the default permissions of 0755 which isn't listed in the tested permissions - we expect for the test to run without any error messages and to print that the test passed before it exits after completion.

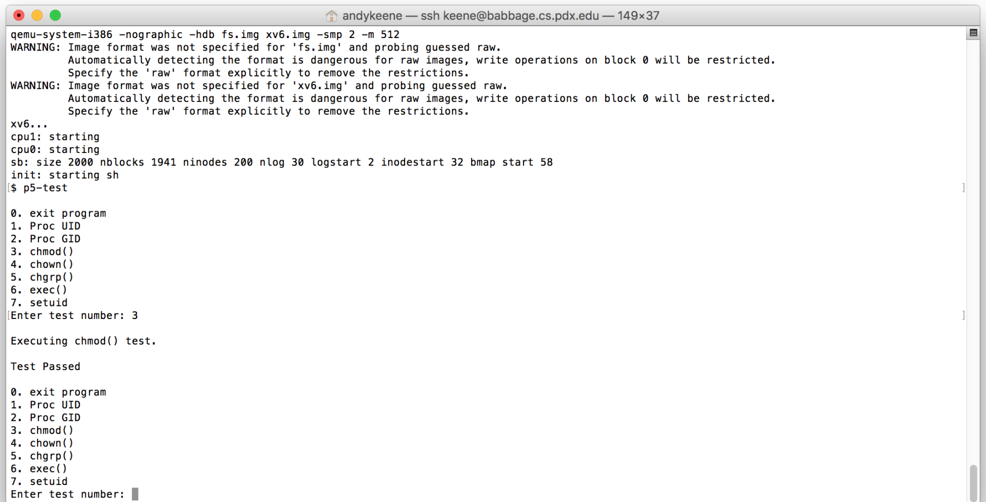


Figure 3: p5-test option 3: doChmodTest()

Here we see that upon boot of xv6 that the program `p5-test` was immediately run, and that by pressing 3 we invoked the specific test for `doChmodTest()`. Next we see that the test ran without errors and that control was returned to the main menu as expected. Thus, since our expectations for this sub-test were met, this sub-test **PASSES**.

0.0.4 chown()

This test will demonstrate the correct behavior of the `chown()` system call. To do this, we will run `p5-test` immediately after the make and boot of xv6 and press 4 to trigger the function `doChownTest()` with an argument of `testsetuid` (described previously in *0.0 System Calls*).

`doChownTest()` will notify us that is is executing then get the meta information for the file testsetuid by calling `stat`. Next it will save the original UID, uid1, of testsetuid before calling `chown()` with an owner argument of uid1+1. Since during the file system creation all files are given a UID of $DEFAULT\_UID$ (demonstrated in test mkfs) which we know by virtue of code to be defined as 0 we can conclude that uid1+1 will in fact be within the valid bounds for a UID and consequently, the system call should succeed. `doChownTest()` then checks the return code of `chown()` where an error message will be printed if the return code corresponds to an error. The file meta information for testtestuid will then be refreshed with another call to stat. Using the stat structure, `doChownTest()` will then ensure that the refreshed UID of the file matches the original UID +1, or uid1+1 (this addition to the test occurs on line 226). Before returning control to the main loop `doChownTest()` will reset the original file UID and print a message of success.

We expect that after making and booting xv6 that when `p5-test` is run and option 4 is selected that we will see only a message that `doChownTest()` is running and a message that the test passed - by virtue of the code described for this test, will will know `chown()` correctly updated the UID.

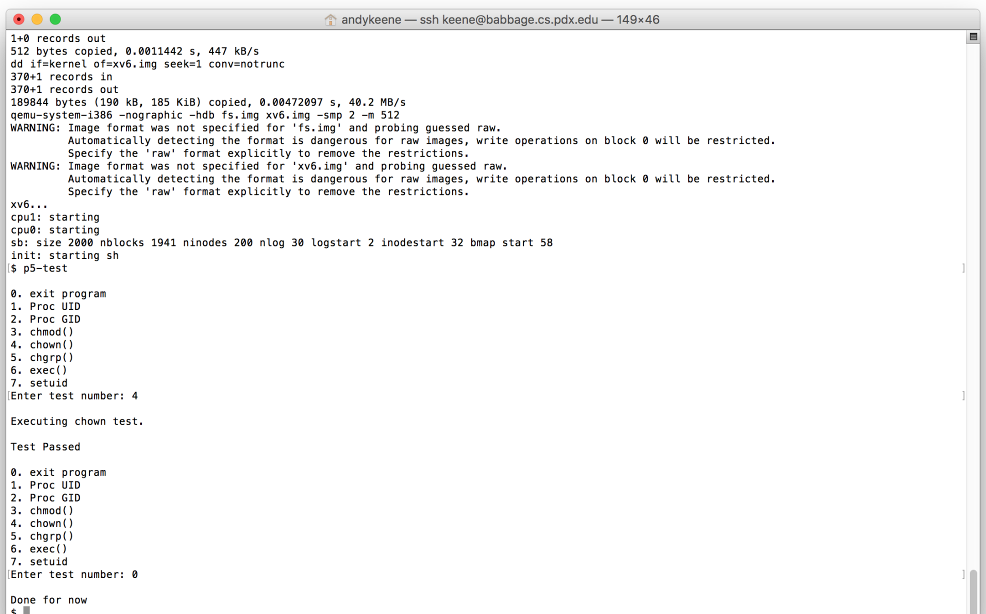


Figure 4: p5-test option 4: doChowntest()

Here we see that upon boot of xv6 that the program `p5-test` was immediately run, and that by pressing 4 we invoked the specific test for `doChownTest()`. Next we see that the test ran without errors and that control was returned to the main menu as expected. Thus, since our expectations for this sub-test were met, this sub-test **PASSES**.

0.0.5 chgrp()

This test will demonstrate the correct behavior of the `chgrp()` system call. To do this, we will run `p5-test` immediately after the make and boot of xv6 and press 5 to trigger the function `doChgrpTest()` with an argument of `testsetuid` (described previously in *0.0 System Calls*).

`doChgrpTest()` will notify us that is is executing then get the meta information for the file testsetuid by calling `stat`. Next it will save the original GID, gid1, of testsetuid before calling `chgrp()` with an owner argument of gid1+1. Since during the file system creation all files are given a GID of $DEFAULT\_GID$ (demonstrated in test mkfs) which we know by virtue of code to be defined as 0 we can conclude that gid1+1 will in fact be within the valid bounds for a GID and consequently, the system call should succeed. `doChgrpTest()` then checks the return code of `chgrp()` where an error message will be printed if the return code corresponds to an error. The file meta information for testtestuid will then be refreshed with another call to stat. Using the stat structure, `doChgrpTest()` will then ensure that the refreshed GID of the file matches the original GID +1, or gid1+1 (this addition to the test occurs on line 256). Before returning control to the main loop `doChgrpTest()` will reset the original file GID and print a message of success.

We expect that after making and booting xv6 that when `p5-test` is run and option 4 is selected that we will see only a message that `doChgrpTest()` is running and a message that the test passed - by virtue of the code described for this test, will will know `chgrp()` correctly updated the GID.

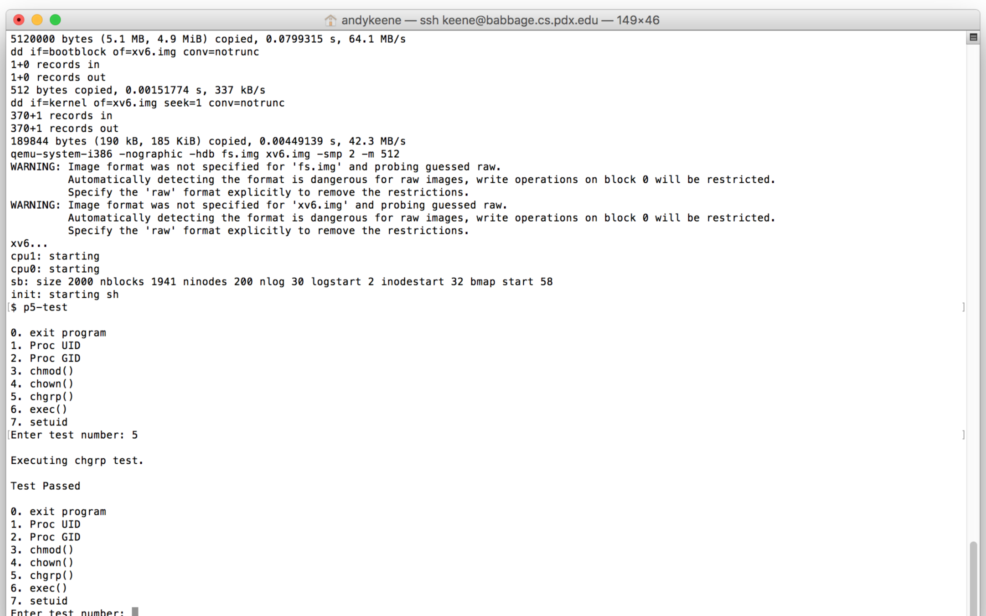


Figure 5: p5-test option 5: doChowntest()

Here we see that upon boot of xv6 that the program `p5-test` was immediately run, and that by pressing 5 we invoked the specific test for `doChgrpTest()`. Next we see that the test ran without errors and that control was returned to the main menu as expected. Thus, since our expectations for this sub-test were met, this sub-test **PASSES**.

0.0.6 exec()

This test will demonstrate the correct behavior of the `exec()` system call. To do this, we will run `p5-test` immediately after the make and boot of xv6 and press 6 to trigger the function `doExecTest()` with an argument of `testsetuid` (described previously in *0.0 System Calls*).

`doExecTest()` will print that is running then will iteratively: set the UID and GID of the process; set the UID, GID, and mode of the file; then it will fork a child which will inherit its parents UID and attempt to execute, via `exec()`, the file testsetuid. If the fork or `exec()` fail an error message will be printed. If the execution of testsetuid succeeds, the program will run and print *"***** In testsetuid: my uid is x"*.

Our expectations for `doExecTest()` behavior for each of the following permission/UID/GID combos, done in-order, follow as:

- Only the file UID and process UID will be set to match with only executable permissions set for the owner - thus we expect the child process should be able to execute the testsetuid program.
- Only file GID and process GID will be set to match with only executable permissions set for the group - thus we expect the child process should be able to execute the testsetuid program.
- Neither the UID or GID of the file and process will be set to match, but executable permissions will be set for other - thus we expect the child process should be able to execute the testsetuid program.
- Both the UID and GID for the field and process will be set to match but no executable permissions will be set - thus we expect the child process should *not* be able to execute the testsetuid program.

Further, expectations for the successful execution of testsetuid will be printed by the child process to the screen prior to attempting `exec()`.

Figure 6: p5-test option 6: doExectest()

Here we see that upon a make and boot of xv6 that we ran option 6 of `p5-test` and that `doExecTest()` successfully completed. Further, we see that from testsetuids execution and printing that the outlined expectations above (and reiterated within the test) are all met - the first three combinations of UID, GID, and permissions for `exec()` succeed while the last fails. Thus, since each of our expectations for this test were met, this sub-test **PASSES**.

0.0.7 setuid flag

This test will demonstrate the correct behavior of the `exec()` system call with the `setuid` mode flag set. To do this, we will run `p5-test` immediately after the make and boot of xv6 and press 7 to trigger the function `doSetuidTest()` with an argument of `testsetuid` (described previously in *0.0 System Calls*).

`doSetuidTest` will follow the same setup procedure as test *0.0.6 exec()* but in additional will now print the UID and GID for both the process and the file, and print the files permissions (as base-10). Given the explicit list of permissions to be set as 1544, 1454, 1445, and 1666 and that the execution of `testsetuid` is to print its UID, our modified expectations are as follows:

- Only the file UID and process UID will be set to match with only executable permissions set for the owner - thus we expect the child process should be able to execute the testsetuid program and since the setuid flag is set that testsetuid should print a UID that matches the file.
- Only file GID and process GID will be set to match with only executable permissions set for the group - thus we expect the child process should be able to execute the testsetuid program and since the setuid flag is set that testsetuid should print a UID that matches the file.
- Neither the UID or GID of the file and process will be set to match, but executable permissions will be set for other - thus we expect the child process should be able to execute the testsetuid program and since the setuid flag is set that testsetuid should print a UID that matches the file.
- Both the UID and GID for the field and process will be set to match but no executable permissions will be set - thus we expect the child process should *not* be able to execute the testsetuid program and no UID should be printed.

Figure 7: p5-test option 7: doSetuidTest()

Here we see that: we ran `p5-test` with option 7; that the first three UID/GID/permission combinations where the process had executable permissions for the file testsetuid and the setuid flag was set, the program executed correctly and printed a UID that matched the file; that on the fourth execution (where no executable permissions were given) failed as expected; and that the test suite deemed `doSetuidTest()` as passing. Thus since all of our expectations for each stage of the test were met we can conclude that this sub-test **PASSES**.

Since all sub-tests passed for the test *System Calls*, we can conclude that the test *System Calls* **PASSES**.

Persistence

This test will demonstrate that any changes made to the file system persist after the xv6 kernel is shut down. To do this we will make and boot xv6, then display the filesystem contents with `ls` which we expect to show the default files with the default UID of 0, GID of 0, and mode of 0755 (or -rwxr-xr-x). Next we will modify the UID, GID and mode of the default file README with values of 111, 222, and 0000 respectively. Next we will execute `ls` again to show the expected effect of these changes on the current directory. We will then shut down xv6 with `halt` and restart with `make qemu-nox`; since the file system already exists, it will not trigger the `mkfs` target in the Makefile and instead simply boot the previously compiled version of xv6. Upon boot, since the file system was not rebuilt, we expect `ls` to reflect that our previous changes persist for the README file.

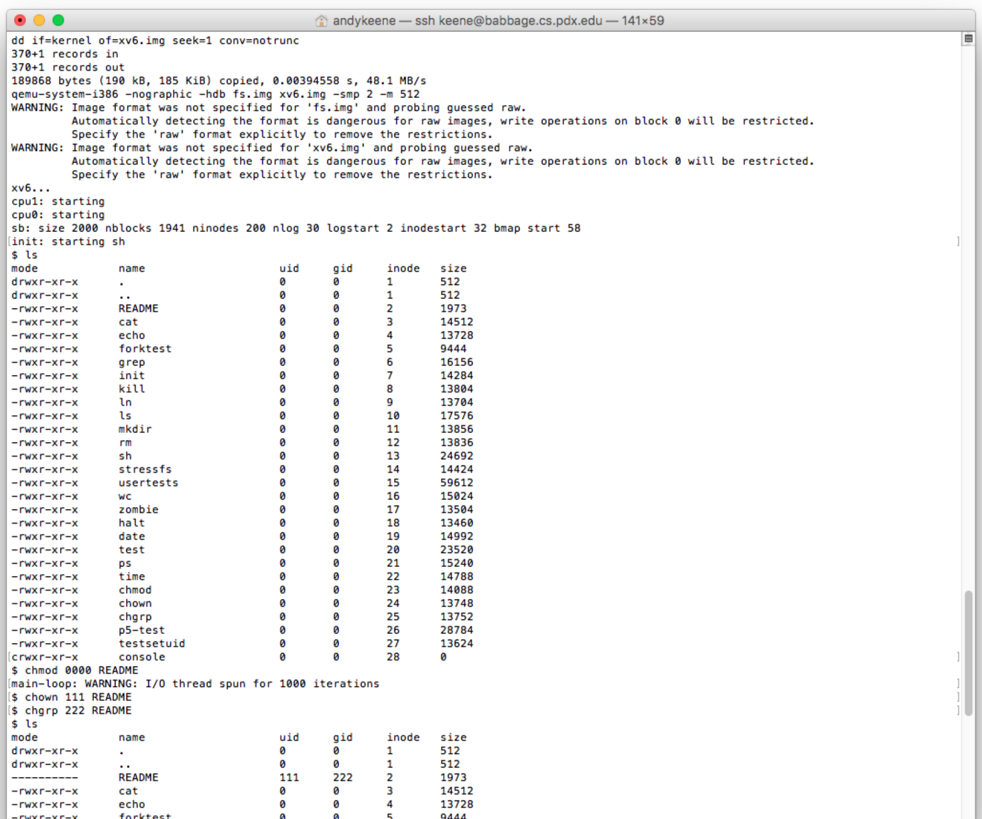
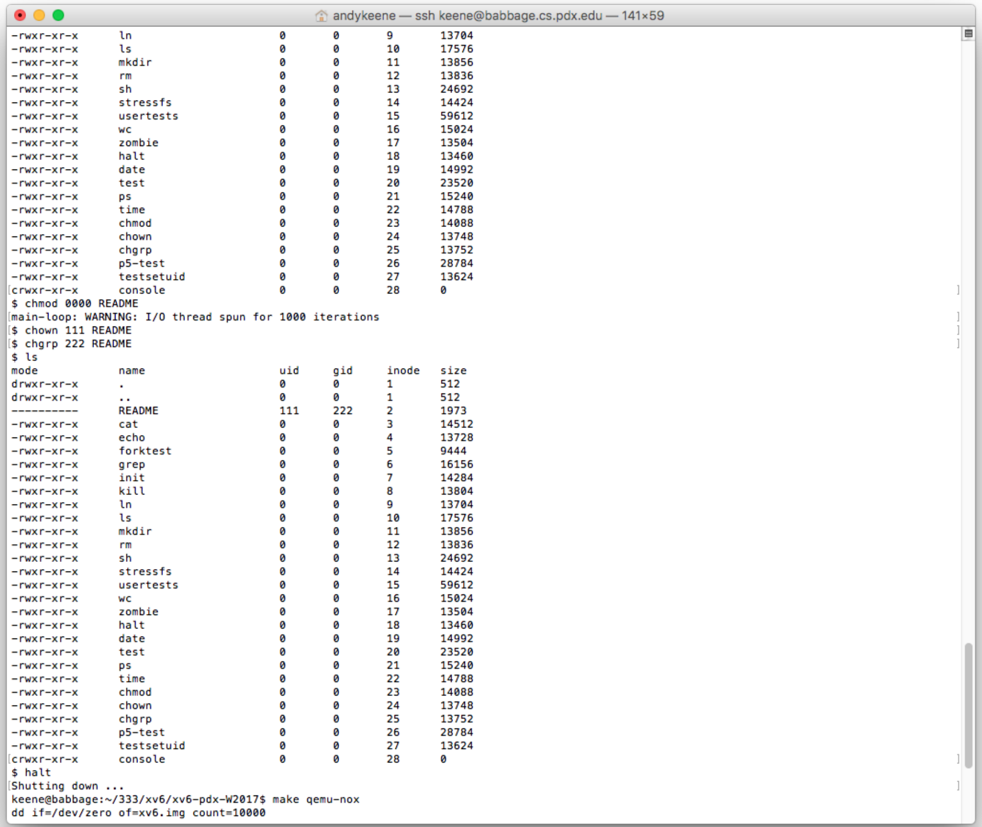
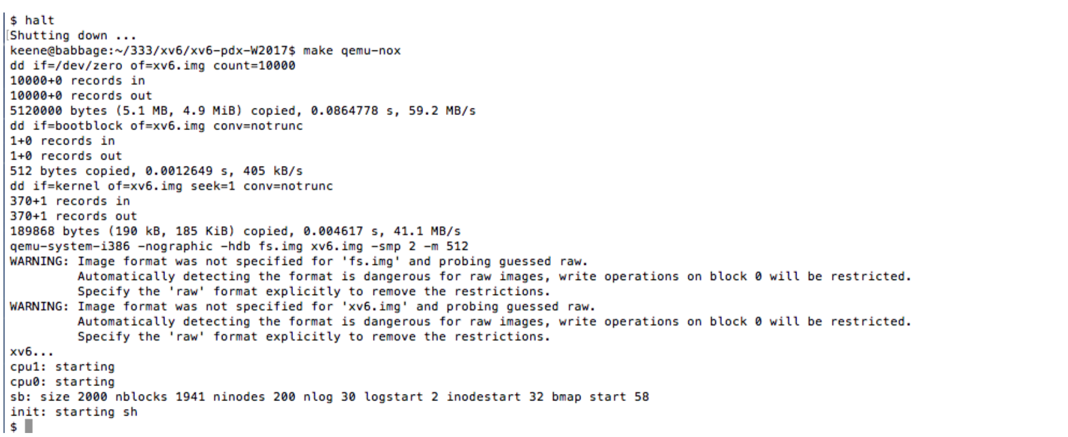


Figure 8: First boot of xv6 with file contents



```
-rwxr-xr-x      ln                  0      0      9      13704
-rwxr-xr-x      ls                  0      0      10     17576
-rwxr-xr-x      mkdir               0      0      11     13856
-rwxr-xr-x      rm                  0      0      12     13836
-rwxr-xr-x      sh                  0      0      13     24692
-rwxr-xr-x      stressfs            0      0      14     14424
-rwxr-xr-x      usertests           0      0      15     59612
-rwxr-xr-x      wc                  0      0      16     15024
-rwxr-xr-x      zombie              0      0      17     13504
-rwxr-xr-x      halt                0      0      18     13460
-rwxr-xr-x      date                0      0      19     14992
-rwxr-xr-x      test                0      0      20     23520
-rwxr-xr-x      ps                  0      0      21     15240
-rwxr-xr-x      time                0      0      22     14788
-rwxr-xr-x      chmod               0      0      23     14088
-rwxr-xr-x      chown               0      0      24     13748
-rwxr-xr-x      chgrp               0      0      25     13752
-rwxr-xr-x      p5-test             0      0      26     28784
-rwxr-xr-x      testsetuid          0      0      27     13624
crwxr-xr-x      console             0      0      28     0
$ chmod 0000 README
main-loop: WARNING: I/O thread spun for 1000 iterations
$ chown 111 README
$ chgrp 222 README
$ ls
mode            name                uid    gid    inode  size
drwxr-xr-x      .                   0      0      1      512
drwxr-xr-x      ..                  0      0      1      512
----------      README              111    222    2      1973
-rwxr-xr-x      cat                 0      0      3      14512
-rwxr-xr-x      echo                0      0      4      13728
-rwxr-xr-x      forktest            0      0      5      9444
-rwxr-xr-x      grep                0      0      6      16156
-rwxr-xr-x      init                0      0      7      14284
-rwxr-xr-x      kill                0      0      8      13804
-rwxr-xr-x      ln                  0      0      9      13704
-rwxr-xr-x      ls                  0      0      10     17576
-rwxr-xr-x      mkdir               0      0      11     13856
-rwxr-xr-x      rm                  0      0      12     13836
-rwxr-xr-x      sh                  0      0      13     24692
-rwxr-xr-x      stressfs            0      0      14     14424
-rwxr-xr-x      usertests           0      0      15     59612
-rwxr-xr-x      wc                  0      0      16     15024
-rwxr-xr-x      zombie              0      0      17     13504
-rwxr-xr-x      halt                0      0      18     13460
-rwxr-xr-x      date                0      0      19     14992
-rwxr-xr-x      test                0      0      20     23520
-rwxr-xr-x      ps                  0      0      21     15240
-rwxr-xr-x      time                0      0      22     14788
-rwxr-xr-x      chmod               0      0      23     14088
-rwxr-xr-x      chown               0      0      24     13748
-rwxr-xr-x      chgrp               0      0      25     13752
-rwxr-xr-x      p5-test             0      0      26     28784
-rwxr-xr-x      testsetuid          0      0      27     13624
crwxr-xr-x      console             0      0      28     0
$ halt
Shutting down ...
keene@babbage:~/333/xv6/xv6-pdx-W2017$ make qemu-nox
dd if=/dev/zero of=xv6.img count=10000
```

Figure 9: Effect of changing REAME UID, GID and mode



```
$ halt
Shutting down ...
keene@babbage:~/333/xv6/xv6-pdx-W2017$ 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, 0.0864778 s, 59.2 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.0012649 s, 405 kB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
370+1 records in
370+1 records out
189868 bytes (190 kB, 185 KiB) copied, 0.004617 s, 41.1 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
$
```

Figure 10: Persistence after rebooting xv6

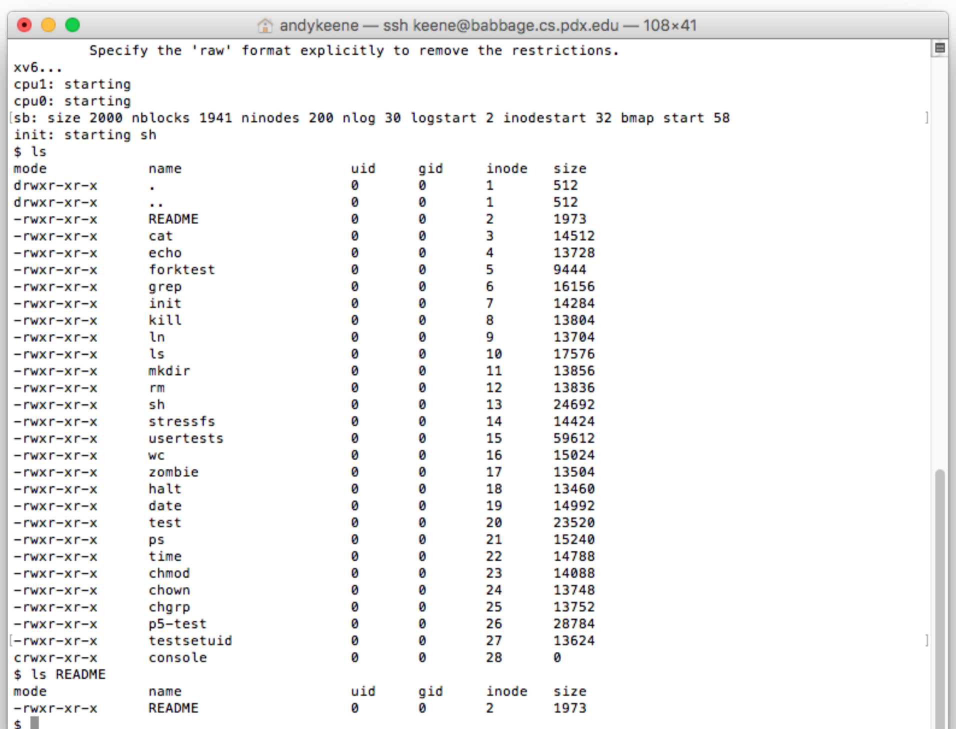
Here we see in the three figures above that all files existed with the default permissions on the first boot of xv6; that `ls` shows the file meta information for README change as expected; and that after shutting down with `halt` and restarting the system with `make qemu-nox` that the changes we made persist. Thus, since our expectations for this sub-test were met, this sub-test **PASSES**.

User Commands

0.0.0 ls

This test will demonstrate that the user command `ls` correctly prints the new associated information with file metadata - namely the inode number, GID, UID, and mode bits. To do this we will make and boot xv6 then immediately list the contents of the current directory with `ls`. Here we expect all default files to be listed in ascending inode numbering. We also expect that since all GID, UID, and mode values are set to the defaults of 0, 0, and 0755 respectively by the `mkfs` program that each file will be listed with these values. It should be noted that the mode 0755 corresponds to listing of permissions as `-rwxr-xr-x`; however, the leading `-` may differ since this is the indicator of whether the file is a directory (i.e. `d`), a device like the console (i.e. `c`), or simply a file (i.e. `-`). We also expect, by virtue of the initial files not changing, that we will only see one directory, which is the root, under the names `.` and `..` - these indicate the current and parent directory. The only device, `c` that we expect to see is the console program responsible for handling interrupts.

We will then use `ls` to list the same meta information for a single file within the directory, README - here we expect that *only* the meta information for README will be listed with matching values of the initial `ls` output.



```
         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      cat                     0       0       3       14512
-rwxr-xr-x      echo                    0       0       4       13728
-rwxr-xr-x      forktest                0       0       5       9444
-rwxr-xr-x      grep                    0       0       6       16156
-rwxr-xr-x      init                    0       0       7       14284
-rwxr-xr-x      kill                    0       0       8       13804
-rwxr-xr-x      ln                      0       0       9       13704
-rwxr-xr-x      ls                      0       0       10      17576
-rwxr-xr-x      mkdir                   0       0       11      13856
-rwxr-xr-x      rm                      0       0       12      13836
-rwxr-xr-x      sh                      0       0       13      24692
-rwxr-xr-x      stressfs                0       0       14      14424
-rwxr-xr-x      usertests               0       0       15      59612
-rwxr-xr-x      wc                      0       0       16      15024
-rwxr-xr-x      zombie                  0       0       17      13504
-rwxr-xr-x      halt                    0       0       18      13460
-rwxr-xr-x      date                    0       0       19      14992
-rwxr-xr-x      test                    0       0       20      23520
-rwxr-xr-x      ps                      0       0       21      15240
-rwxr-xr-x      time                    0       0       22      14788
-rwxr-xr-x      chmod                   0       0       23      14088
-rwxr-xr-x      chown                   0       0       24      13748
-rwxr-xr-x      chgrp                   0       0       25      13752
-rwxr-xr-x      p5-test                 0       0       26      28784
-rwxr-xr-x      testsetuid              0       0       27      13624
crwxr-xr-x      console                 0       0       28      0
$ ls README
mode            name                    uid     gid     inode   size
-rwxr-xr-x      README                  0       0       2       1973
$
```

Figure 11: ls

In the figure, *ls*, above we see: that all files have the default permissions `-rwxr-xr-x` with the exception of `.` and `..` which contain a directory denotation and the console file which contains a `c` denotation; we also see that all inodes are listed in ascending order; that all files have the default UID and GID of 0; and that individual listing by `ls` of README matches the initial output values and formatting. Thus, since all of our expectations were met we can conclude that this sub-test **PASSES**.

0.0.1 Valid Parameters for chmod, chown and chrp

This test will demonstrate that the user commands `chmod`, `chown`, and `chgrp` change the mode, UID, and GID, respectively, of a file when passed in valid parameters (other than the defaults). To do this we will first boot xv6 and execute `ls` to show the current file directory - note `ls` was demonstrated as correct previously in the test *ls*. Here we expect the files listed include the file *cat* which is to be used for our testing, and that all files will have the default mode 0755, the default GID of 0, and the default UID of 0 (by virtue of definitions in param.h). We will then:

- Invoke the command `chmod 0777 cat` then execute `ls` whose output we now expect to display the permissions `-rwxrwxrwx` for cat.
- Invoke the command `chown 19 cat` then execute `ls` whose output we now expect to differ from the last only in the display of the UID `19` for cat.
- Invoke the command `chgrp 88 cat` then execute `ls` whose output we now expect to differ from the last only in the display of the GID `88` for cat.

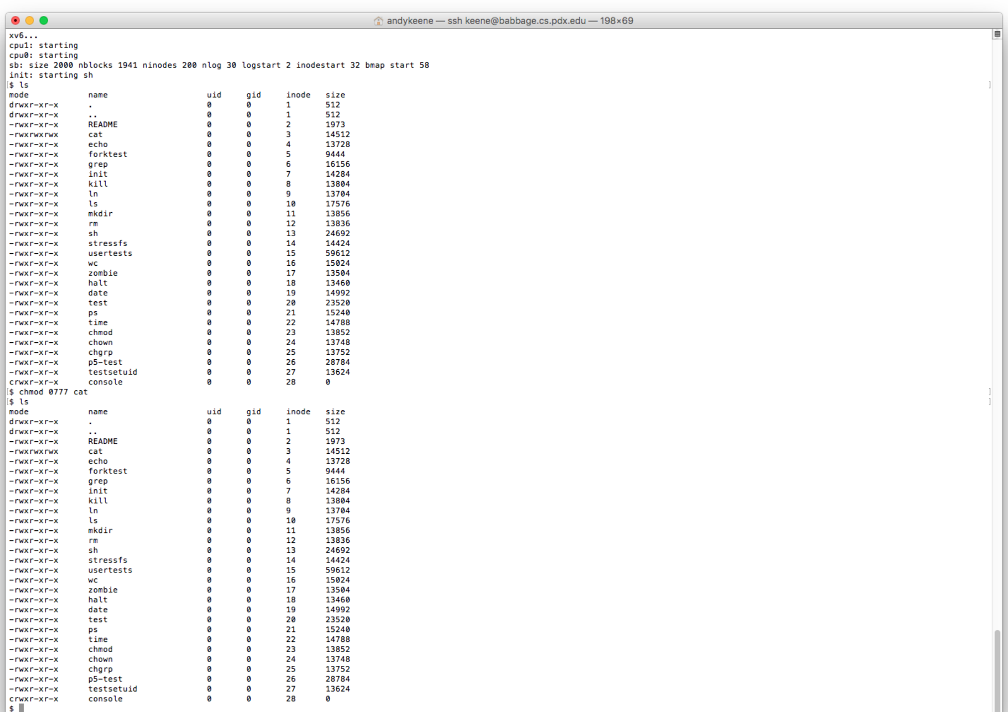
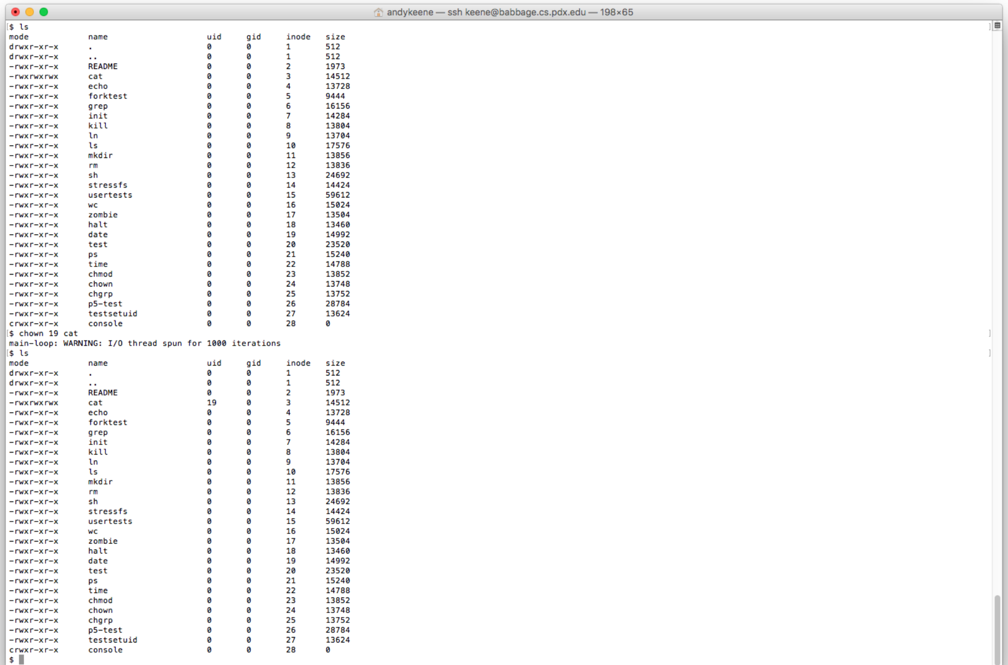
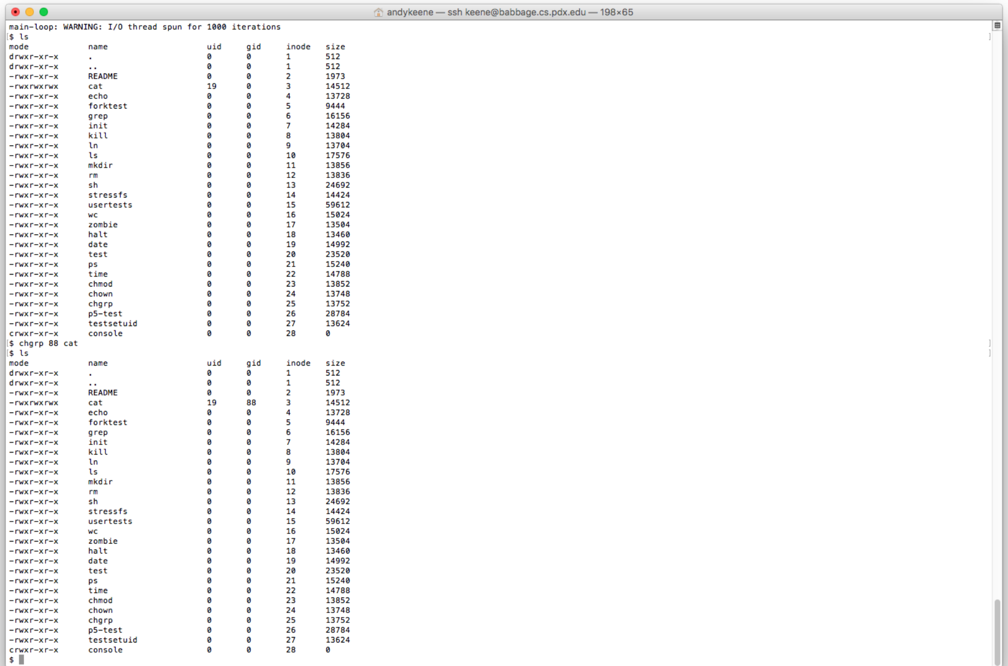


Figure 12: chmod 0755 cat




```
$ 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
-rwxrwxrwx    cat                0       0       3       14512
-rwxr-xr-x    echo               0       0       4       13728
-rwxr-xr-x    forktest           0       0       5       9444
-rwxr-xr-x    grep               0       0       6       16156
-rwxr-xr-x    init               0       0       7       14284
-rwxr-xr-x    kill               0       0       8       13804
-rwxr-xr-x    ln                 0       0       9       13704
-rwxr-xr-x    ls                 0       0       10      17576
-rwxr-xr-x    mkdir              0       0       11      13856
-rwxr-xr-x    rm                 0       0       12      13836
-rwxr-xr-x    sh                 0       0       13      24692
-rwxr-xr-x    stressfs           0       0       14      14424
-rwxr-xr-x    usertests          0       0       15      59612
-rwxr-xr-x    wc                 0       0       16      15024
-rwxr-xr-x    zombie             0       0       17      13504
-rwxr-xr-x    halt               0       0       18      13460
-rwxr-xr-x    date               0       0       19      14992
-rwxr-xr-x    test               0       0       20      23520
-rwxr-xr-x    ps                 0       0       21      15240
-rwxr-xr-x    time               0       0       22      14788
-rwxr-xr-x    chmod              0       0       23      13852
-rwxr-xr-x    chown              0       0       24      13748
-rwxr-xr-x    chgrp              0       0       25      13752
-rwxr-xr-x    p5-test            0       0       26      28784
-rwxr-xr-x    testsetuid         0       0       27      13624
crwxr-xr-x    console            0       0       28      0
$ chown 19 cat
main-loop: WARNING: I/O thread spun for 1000 iterations
$ 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
-rwxrwxrwx    cat                19      0       3       14512
-rwxr-xr-x    echo               0       0       4       13728
-rwxr-xr-x    forktest           0       0       5       9444
-rwxr-xr-x    grep               0       0       6       16156
-rwxr-xr-x    init               0       0       7       14284
-rwxr-xr-x    kill               0       0       8       13804
-rwxr-xr-x    ln                 0       0       9       13704
-rwxr-xr-x    ls                 0       0       10      17576
-rwxr-xr-x    mkdir              0       0       11      13856
-rwxr-xr-x    rm                 0       0       12      13836
-rwxr-xr-x    sh                 0       0       13      24692
-rwxr-xr-x    stressfs           0       0       14      14424
-rwxr-xr-x    usertests          0       0       15      59612
-rwxr-xr-x    wc                 0       0       16      15024
-rwxr-xr-x    zombie             0       0       17      13504
-rwxr-xr-x    halt               0       0       18      13460
-rwxr-xr-x    date               0       0       19      14992
-rwxr-xr-x    test               0       0       20      23520
-rwxr-xr-x    ps                 0       0       21      15240
-rwxr-xr-x    time               0       0       22      14788
-rwxr-xr-x    chmod              0       0       23      13852
-rwxr-xr-x    chown              0       0       24      13748
-rwxr-xr-x    chgrp              0       0       25      13752
-rwxr-xr-x    p5-test            0       0       26      28784
-rwxr-xr-x    testsetuid         0       0       27      13624
crwxr-xr-x    console            0       0       28      0
$
```

Figure 13: chown 19 cat



andykeene — ssh keene@babbage.cs.pdx.edu — 198×65

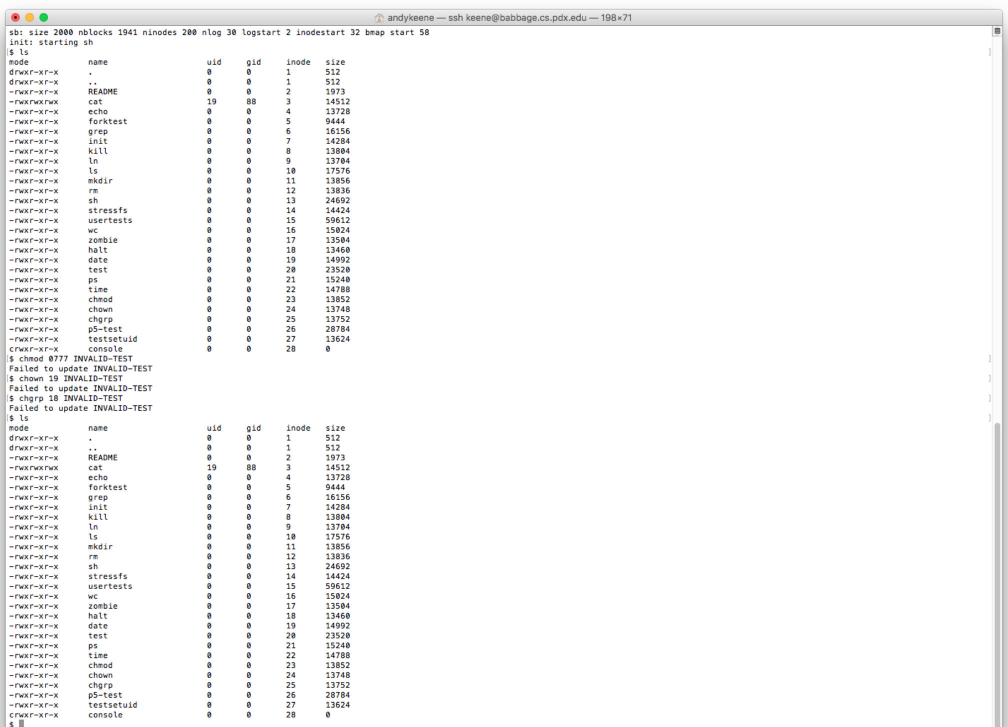
```
main-loop: WARNING: I/O thread spun for 1000 iterations
$ ls
mode            name                uid     gid     inode   size
drwxr-xr-x      .                   0       0       1       512
drwxr-xr-x      ..                  0       0       1       512
-rwxr-xr-x      README              0       0       2       1973
-rwxrwxrwx      cat                 19      0       3       14512
-rwxr-xr-x      echo                0       0       4       13728
-rwxr-xr-x      forktest            0       0       5       9444
-rwxr-xr-x      grep                0       0       6       16156
-rwxr-xr-x      init                0       0       7       14284
-rwxr-xr-x      kill                0       0       8       13804
-rwxr-xr-x      ln                  0       0       9       13704
-rwxr-xr-x      ls                  0       0       10      17576
-rwxr-xr-x      mkdir               0       0       11      13856
-rwxr-xr-x      rm                  0       0       12      13836
-rwxr-xr-x      sh                  0       0       13      24692
-rwxr-xr-x      stressfs            0       0       14      14424
-rwxr-xr-x      usertests           0       0       15      59612
-rwxr-xr-x      wc                  0       0       16      15024
-rwxr-xr-x      zombie              0       0       17      13504
-rwxr-xr-x      halt                0       0       18      13460
-rwxr-xr-x      date                0       0       19      14992
-rwxr-xr-x      test                0       0       20      23520
-rwxr-xr-x      ps                  0       0       21      15240
-rwxr-xr-x      time                0       0       22      14788
-rwxr-xr-x      chmod               0       0       23      13852
-rwxr-xr-x      chown               0       0       24      13748
-rwxr-xr-x      chgrp               0       0       25      13752
-rwxr-xr-x      p5-test             0       0       26      28784
-rwxr-xr-x      testsetuid          0       0       27      13624
crwxr-xr-x      console             0       0       28      0
$ chgrp 88 cat
$ 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
-rwxrwxrwx      cat                 19      88      3       14512
-rwxr-xr-x      echo                0       0       4       13728
-rwxr-xr-x      forktest            0       0       5       9444
-rwxr-xr-x      grep                0       0       6       16156
-rwxr-xr-x      init                0       0       7       14284
-rwxr-xr-x      kill                0       0       8       13804
-rwxr-xr-x      ln                  0       0       9       13704
-rwxr-xr-x      ls                  0       0       10      17576
-rwxr-xr-x      mkdir               0       0       11      13856
-rwxr-xr-x      rm                  0       0       12      13836
-rwxr-xr-x      sh                  0       0       13      24692
-rwxr-xr-x      stressfs            0       0       14      14424
-rwxr-xr-x      usertests           0       0       15      59612
-rwxr-xr-x      wc                  0       0       16      15024
-rwxr-xr-x      zombie              0       0       17      13504
-rwxr-xr-x      halt                0       0       18      13460
-rwxr-xr-x      date                0       0       19      14992
-rwxr-xr-x      test                0       0       20      23520
-rwxr-xr-x      ps                  0       0       21      15240
-rwxr-xr-x      time                0       0       22      14788
-rwxr-xr-x      chmod               0       0       23      13852
-rwxr-xr-x      chown               0       0       24      13748
-rwxr-xr-x      chgrp               0       0       25      13752
-rwxr-xr-x      p5-test             0       0       26      28784
-rwxr-xr-x      testsetuid          0       0       27      13624
crwxr-xr-x      console             0       0       28      0
$
```

Figure 14: chgrp 88 cat

We see that immediately after boot that the file cat does in fact exist in the current directory with the default mode of 0755, the default UID of 0, and the default GID of 0. We also see that after execution of `chmod 0777 cat` that the file then has the correct corresponding permissions `-rwxrwxrwx`; that after the execution of `chown 19 cat` the file has the correct UID of 19; and that after the execution of `chgrp 88 cat` that the file has the correct GID of 88. Thus all of our expectations were met. Since all of our expectations were met, this sub-test **PASSES**.

0.0.2 Invalid filename parameter for chmod, chown and chrp

This test will demonstrate that the user commands `chmod`, `chown`, and `chgrp` do not modify any file when given an invalid filename. To do this we will first boot xv6 and execute `ls` to show the current file directory - note `ls` was demonstrated as correct previously in the test *ls*. We will then choose a filename that does not exist in the directory, such as TEST-INVALID, and execute the user commands `chmod`, `chown`, and `chgrp` with the valid numeric parameters 0777, 19, and 88, but with the invalid filename TEST-INVALID. The behavior for these user commands with valid parameters was demonstrated correct in the previous test *Valid Parameters for chmod, chown and chrp* which will demonstrate that any failure here is due strictly to an invalid filename. Since, by the construction of these user commands an error message will be printed upon an unsuccessful return code of the system call, we expect after the execution of each command `chmod`, `chown`, and `chgrp` to print an error message. We will then execute `ls` again, and expect the output to be identical to the the first.




```
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
-rwxrwxrwx      cat                     19      88      3       14512
-rwxr-xr-x      echo                    0       0       4       13728
-rwxr-xr-x      forktest                0       0       5       9444
-rwxr-xr-x      grep                    0       0       6       16156
-rwxr-xr-x      init                    0       0       7       14284
-rwxr-xr-x      kill                    0       0       8       13804
-rwxr-xr-x      ln                      0       0       9       13704
-rwxr-xr-x      ls                      0       0       10      17576
-rwxr-xr-x      mkdir                   0       0       11      13856
-rwxr-xr-x      rm                      0       0       12      13836
-rwxr-xr-x      sh                      0       0       13      24692
-rwxr-xr-x      stressfs                0       0       14      14424
-rwxr-xr-x      usertests               0       0       15      59612
-rwxr-xr-x      wc                      0       0       16      15024
-rwxr-xr-x      zombie                  0       0       17      13504
-rwxr-xr-x      halt                    0       0       18      13460
-rwxr-xr-x      date                    0       0       19      14992
-rwxr-xr-x      test                    0       0       20      23520
-rwxr-xr-x      ps                      0       0       21      15240
-rwxr-xr-x      time                    0       0       22      14788
-rwxr-xr-x      chmod                   0       0       23      13852
-rwxr-xr-x      chown                   0       0       24      13748
-rwxr-xr-x      chgrp                   0       0       25      13752
-rwxr-xr-x      p5-test                 0       0       26      28784
-rwxr-xr-x      testsetuid              0       0       27      13624
crwxr-xr-x      console                 0       0       28      0
$ chmod 0777 INVALID-TEST
Failed to update INVALID-TEST
$ chown 19 INVALID-TEST
Failed to update INVALID-TEST
$ chgrp 18 INVALID-TEST
Failed to update INVALID-TEST
$ 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
-rwxrwxrwx      cat                     19      88      3       14512
-rwxr-xr-x      echo                    0       0       4       13728
-rwxr-xr-x      forktest                0       0       5       9444
-rwxr-xr-x      grep                    0       0       6       16156
-rwxr-xr-x      init                    0       0       7       14284
-rwxr-xr-x      kill                    0       0       8       13804
-rwxr-xr-x      ln                      0       0       9       13704
-rwxr-xr-x      ls                      0       0       10      17576
-rwxr-xr-x      mkdir                   0       0       11      13856
-rwxr-xr-x      rm                      0       0       12      13836
-rwxr-xr-x      sh                      0       0       13      24692
-rwxr-xr-x      stressfs                0       0       14      14424
-rwxr-xr-x      usertests               0       0       15      59612
-rwxr-xr-x      wc                      0       0       16      15024
-rwxr-xr-x      zombie                  0       0       17      13504
-rwxr-xr-x      halt                    0       0       18      13460
-rwxr-xr-x      date                    0       0       19      14992
-rwxr-xr-x      test                    0       0       20      23520
-rwxr-xr-x      ps                      0       0       21      15240
-rwxr-xr-x      time                    0       0       22      14788
-rwxr-xr-x      chmod                   0       0       23      13852
-rwxr-xr-x      chown                   0       0       24      13748
-rwxr-xr-x      chgrp                   0       0       25      13752
-rwxr-xr-x      p5-test                 0       0       26      28784
-rwxr-xr-x      testsetuid              0       0       27      13624
crwxr-xr-x      console                 0       0       28      0
$
```

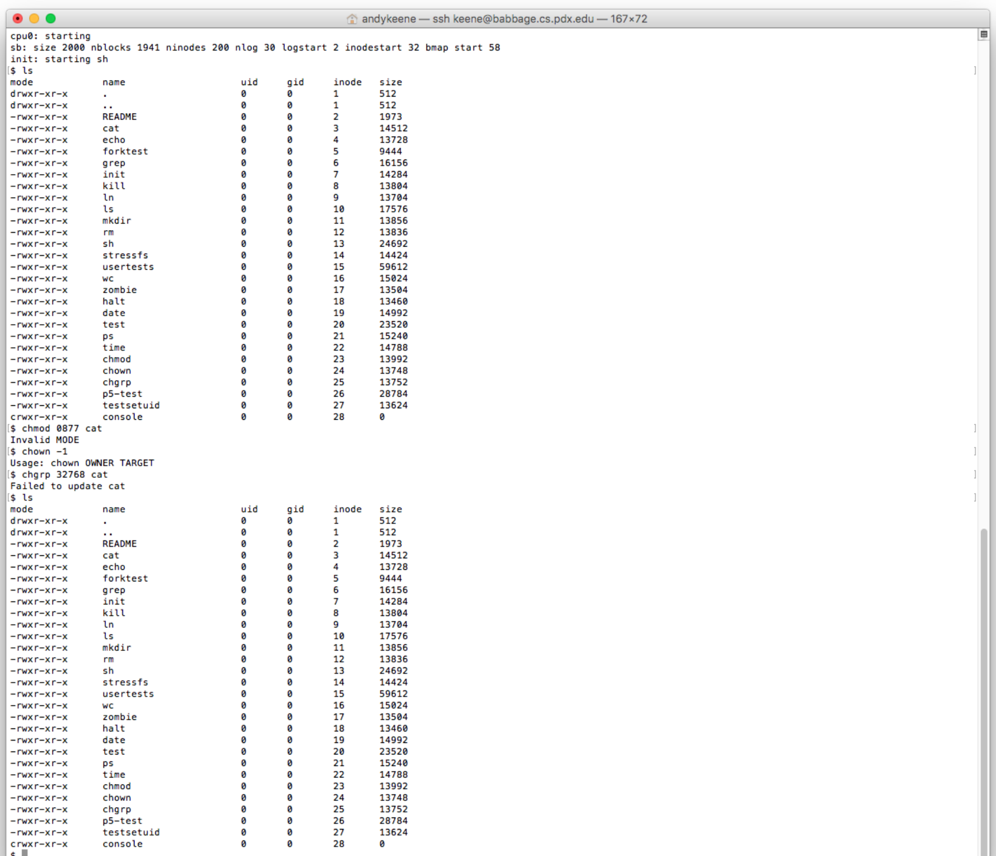
Figure 15: Invalid Filename Parameter

Here we see that the output from `ls` immediately after xv6 booted is identical to its output after the attempted user commands and that the file INVALID-TEST did not exist; further we see that each attempted user command with the invalid filename parameter printed the error message *"Failed to update INVALID-TEST"*. Thus, since our expectations for this sub-test are met we can conclude that this sub-test **PASSES**.

0.0.3 Invalid numeric parameter for chmod, chown and chrp

This test will demonstrate that the user commands `chmod`, `chown`, and `chgrp` do not modify any file when given an invalid numeric parameter. To do this we will first boot xv6 and execute `ls` to show the current file directory - note `ls` was demonstrated as correct previously in the test *ls*. We will then choose a filename that exists in the directory, such as cat, and execute the user command `chmod`, `chown`, and `chgrp` with the invalid numeric parameters 0877, -1, and 32768 respectively, but with the *valid* filename cat. The behavior for these user commands with valid parameters was demonstrated correct in the previous test *Valid Parameters for chmod, chown and chrp* which will demonstrate that any failure here is due strictly to an invalid numeric parameter. Since, by the construction of these user commands an error message will be printed upon an unsuccessful return code of the system call, we expect after the execution of each command `chmod`, `chown`, and `chgrp` to print an error message; specifically we expect `chown`, and `chgrp` to print *"Failed to update cat"* and `chmod` to print *"Invalid MODE"*.

We will then execute `ls` again, and expect the output to be identical to the the first.



```
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    cat           0      0      3       14512
-rwxr-xr-x    echo          0      0      4       13728
-rwxr-xr-x    forktest      0      0      5       9444
-rwxr-xr-x    grep          0      0      6       16156
-rwxr-xr-x    init          0      0      7       14284
-rwxr-xr-x    kill          0      0      8       13804
-rwxr-xr-x    ln            0      0      9       13704
-rwxr-xr-x    ls            0      0      10      17576
-rwxr-xr-x    mkdir         0      0      11      13856
-rwxr-xr-x    rm            0      0      12      13836
-rwxr-xr-x    sh            0      0      13      24692
-rwxr-xr-x    stressfs      0      0      14      14424
-rwxr-xr-x    usertests     0      0      15      59612
-rwxr-xr-x    wc            0      0      16      15024
-rwxr-xr-x    zombie        0      0      17      13504
-rwxr-xr-x    halt          0      0      18      13460
-rwxr-xr-x    date          0      0      19      14992
-rwxr-xr-x    test          0      0      20      23520
-rwxr-xr-x    ps            0      0      21      15240
-rwxr-xr-x    time          0      0      22      14788
-rwxr-xr-x    chmod         0      0      23      13992
-rwxr-xr-x    chown         0      0      24      13748
-rwxr-xr-x    chgrp         0      0      25      13752
-rwxr-xr-x    p5-test       0      0      26      28784
-rwxr-xr-x    testsetuid    0      0      27      13624
crwxr-xr-x    console       0      0      28      0
$ chmod 0877 cat
Invalid MODE
$ chown -1
Usage: chown OWNER TARGET
$ chgrp 32768 cat
Failed to update cat
$ 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    cat           0      0      3       14512
-rwxr-xr-x    echo          0      0      4       13728
-rwxr-xr-x    forktest      0      0      5       9444
-rwxr-xr-x    grep          0      0      6       16156
-rwxr-xr-x    init          0      0      7       14284
-rwxr-xr-x    kill          0      0      8       13804
-rwxr-xr-x    ln            0      0      9       13704
-rwxr-xr-x    ls            0      0      10      17576
-rwxr-xr-x    mkdir         0      0      11      13856
-rwxr-xr-x    rm            0      0      12      13836
-rwxr-xr-x    sh            0      0      13      24692
-rwxr-xr-x    stressfs      0      0      14      14424
-rwxr-xr-x    usertests     0      0      15      59612
-rwxr-xr-x    wc            0      0      16      15024
-rwxr-xr-x    zombie        0      0      17      13504
-rwxr-xr-x    halt          0      0      18      13460
-rwxr-xr-x    date          0      0      19      14992
-rwxr-xr-x    test          0      0      20      23520
-rwxr-xr-x    ps            0      0      21      15240
-rwxr-xr-x    time          0      0      22      14788
-rwxr-xr-x    chmod         0      0      23      13992
-rwxr-xr-x    chown         0      0      24      13748
-rwxr-xr-x    chgrp         0      0      25      13752
-rwxr-xr-x    p5-test       0      0      26      28784
-rwxr-xr-x    testsetuid    0      0      27      13624
crwxr-xr-x    console       0      0      28      0
$
```

Figure 16: Invalid Numeric Parameter

In the figure above we see that immediately after xv6 is booted that `ls` shows cat is in the current directory with default the permissions, GID, and UID. Next we see that as expected `chmod`, `chown`, and `chgrp` respond to our invalid numeric parameters with the appropriate messages *"Failed to update cat"* and *"Invalid MODE"*; where further we see the second `ls` output is identical to the first, showing that no attempts to change the files meta information were successful. Since all of our expectations for this sub-test were met we can conclude that this sub-test **PASSES**.

This all sub-tests for *User Commands* passed, we can conclude that this test **PASSES**